Capítulo 4

DESARROLLO DE LA APLICACIÓN DE TIERRA.

Desde el punto de vista del software, la aplicación C residente en el computador de vuelo se comunicaba originalmente con otro programa en C también alojado en la placa Hércules. Este programa hacía las funciones del ordenador de tierra, y estaba estructurado en una función central ***control.c*** donde, al igual que en ***main.c*** se encuentra un bucle principal `switch (n_control)` a partir del cual o bien se llama a una función para el envío de *n_control* o de la trama de inicialización (***envia.c***), o bien se llama a una función para la recepción del archivo de datos muestreados(***rec_archivo.c***). Aparte se tienen dos hilos: uno para la recepción de los datos en tiempo real y otro para la recepción de los códigos de error.

Este programa se ejecutaba en línea de comandos y fue indispensable para probar la eficiencia del software de recogida y envío de datos en el helicóptero en fases anteriores del proyecto Hermes. Ahora, el objetivo principal de este proyecto es el diseño de una aplicación en un entorno gráfico alojada en un PC de tierra que permita la construcción de las tramas, el envío de ordenes y la recepción de datos desde el helicóptero de forma rápida y cómoda. Además, se pretende implementar un sistema de monitorización del estado del sistema, con vistas a las pruebas reales en vuelo en las que el helicóptero no tendrá monitor alguno conectado y será el control de tierra el que tenga que informar al usuario de situaciones anómalas.

A la hora de elegir una plataforma adecuada para nuestros objetivos, se ha tenido en cuenta que ha de tratarse de una herramienta robusta, capaz de procesar y representar grandes cantidades de datos y que ofrezca la posibilidad del diseño de un entorno gráfico. Así pues se decide utilizar MATLAB.

El nombre de MATLAB proviene de la contracción de los términos 'MATrix LABoratory', y es un entorno de computación y desarrollo de aplicaciones totalmente integrado, orientado para llevar a cabo proyectos en donde se encuentren implicados elevados cálculos matemáticos y la representación gráfica de los mismos; integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica. Los usos más característicos de MATLAB los encontramos en áreas de computación y cálculo numérico tradicional, prototipaje algorítmico, teoría de control automático, estadística y análisis de series temporales para el proceso digital de señal.

La ventaja principal de MATLAB es el uso de familias de comandos de áreas específicas llamadas toolboxes, grupos de comandos de MATLAB (archivos .M) que extienden el ambiente de MATLAB para resolver problemas de áreas específicas como las que se detallan a continuación:

- Procesamiento de Señal.

- The MATLAB C Math Library.
- Matemáticas Simbólicas.
- Procesamiento de Imagen.
- The MATLAB Compiler.
- Redes Neuronales.
- Estadística.
- Splines.
- Diseño de Sistemas de Control.
- Control Robusto.
- Identificación de Sistemas.
- Optimización.
- Simulación.
- Diseño de Control no Lineal.
- Lógica Difusa.
- NAG Fundation Toolbox.

Vemos pues que las herramientas de procesamiento, de sistemas de control y de simulación serán de enorme utilidad para la identificación de parámetros del modelo físico del sistema y la simulación e implementación de los sistemas de control.

4.1. MATLAB: Graphical User Interfaces GUIs.

MATLAB también posee un conjunto de herramientas para el desarrollo de aplicaciones que requieran de una interfaz gráfica de usuario (GUI,Graphical User Interface). Ofrece así un entorno de diseño en el que se pueden desarrollar fácilmente un conjunto de pantallas (paneles) con botones, menús, ventanas, etc., que permiten utilizar de manera muy simple programas realizados dentro de este entorno. Las posibilidades que ofrece MATLAB en entornos gráficos no son quizás muy amplias, en comparación a otras aplicaciones de Windows como Visual Basic, Visual C. Sin embargo, para los objetivos a cubrir en el control de tierra son muy aceptables.

La elaboración de GUIs puede llevarse a cabo de dos formas, la primera de ellas consiste en escribir un programa que genere la GUI (script), la segunda opción consiste en utilizar la herramienta de diseño de GUIs, incluida en el Matlab, llamada GUIDE. Para el proyecto utilizaremos esta última.

El panel GUI se crea en una ventana, identificada como figura y está formada por los siguientes elementos:

-Menú de interfaz con el usuario.

-Dispositivos de control de la interfaz con el usuario.

-Ejes para desplegar las gráficas o imágenes.

Mediante la GUI, el flujo de información está controlado por las acciones (eventos) que sucedan en la interfaz. Comparando con los scripts, en éstos los comandos están en un orden preestablecido, mientras que en la GUI no lo están. Los comandos para crear una GUI se escriben en un script, pero una vez que se ejecuta la GUI, ésta permanece en la pantalla aunque se haya terminado la ejecución del script y la interacción con el usuario continúa hasta que se cierra.

A la hora de afrontar el diseño de la aplicación, lo más coherente es distribuir la construcción por bloques funcionales lógicos en los que se puede dividir el programa en el helicóptero. Éstos son:

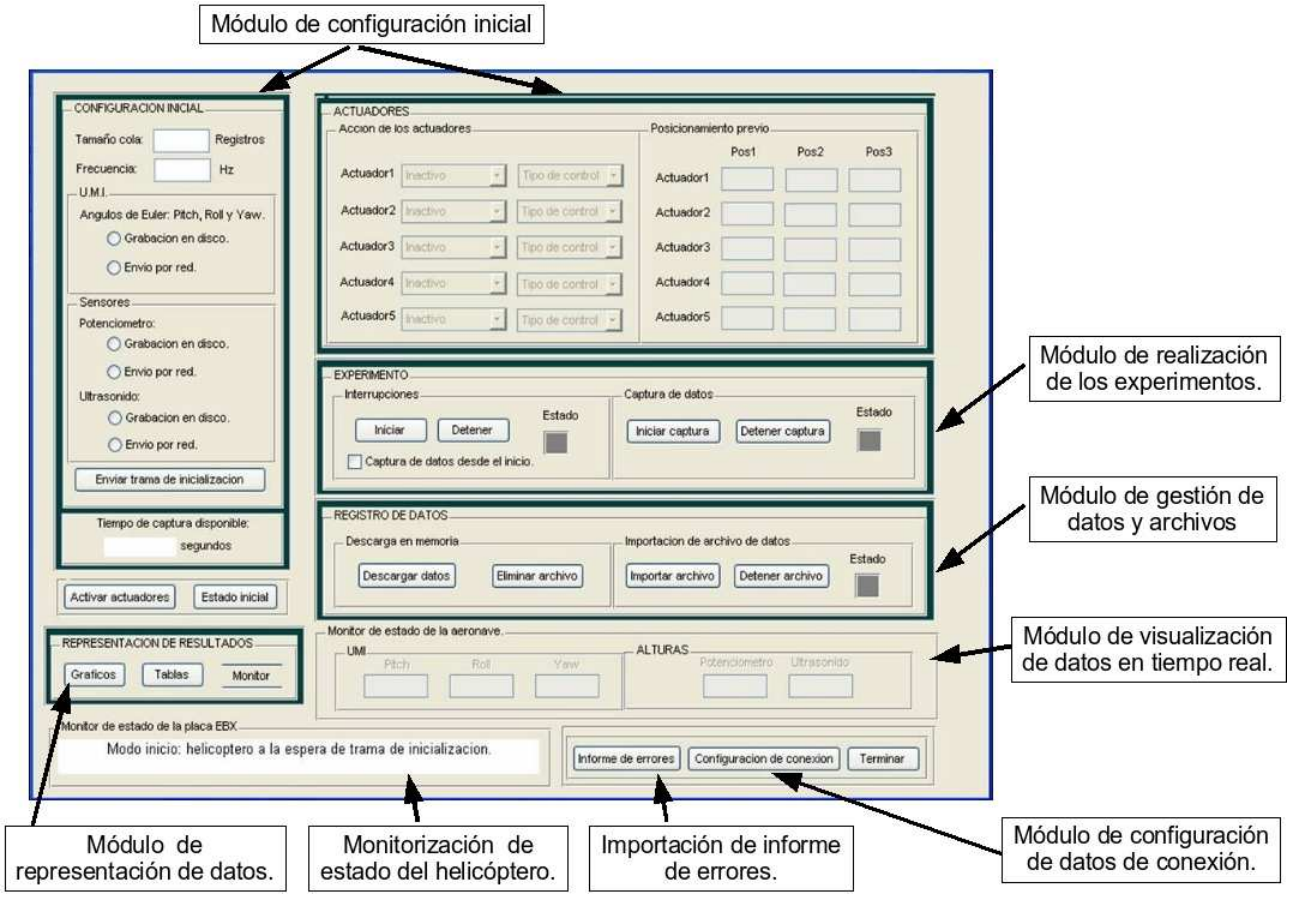


Figura 4.1: Módulos que conforman la aplicación gráfica de tierra.

- Módulo de inicialización.

- Módulo de configuración de datos de red.

- Módulo de realización de experimentos.

- Módulo de manipulación de datos y archivos registrados.

- Módulo de representación de los datos importados.

- Módulo de recepción de datos en tiempo real.

- Otras funcionalidades.

4.2. Módulo de inicialización.

4.2.1. Código C en computador de vuelo.

Para definir correctamente las funcionalidades a cubrir por este módulo, estudiamos cómo se lleva a cabo el proceso de inicialización en la aplicación del ordenador de abordo.

Estructura de la trama de inicialización:

Los parámetros necesarios para inicializar los experimentos se aunan en una estructura llamada *datos* que se define como:

```
struct tierra{
    int d0;
    int t_cola;
    int frec;
    struct  umi umi[M];
    struct ad ad[N];
    struct act a[4];
}datos;
```

Código fuente 4.1: Estructura principal de parámetros de inicialización en C.

datos.d0: valor entero que será utilizado en el bucle principal de las funciones de tierra en la aplicación C. Ya no será necesaria en la aplicación Matlab.

datos.t_cola:valor entero en el que se especifica el tamaño de la cola.

datos.frec: valor entero en el que se especifica la frecuencia de interrupción. Habrá de estar comprendido entre 1 y 1000Hz.

datos.umi, datos.ad y datos.act son a su vez estructuras que albergan parámetros de configuración relativos a los sensores y a los actuadores:

```
struct umi{
    unsigned int umi:1;
    unsigned int umi_disco:1;
    unsigned int umi_red:1;
}umi[M];

struct ad{
    unsigned int ad:1;
    unsigned int ad_disco:1;
    unsigned int ad_red:1;
}ad[N];
```

```
struct act{
        int actuador;
        int control;
        float p1;
        float p2;
        float p3;
}a[4];
```

Código fuente 4.2: Estructuras de sensores y actuadores en C.

M será 3 para albergar datos del pitch, el roll y el yaw. Cada estructura *datos.umi[i]* estará constituida por tres bits, el bit 'umi' para indicar si dicha entrada va a ser escaneada, el bit 'umi_disco' para introducir dicho dato en la cola tras escanearlo y el bit 'umi_red' para enviarlo directamente por red en tiempo real. Una estructura similar tendrá la correspondiente a los sensores de altura, en la que N es igual a 2 para albergar datos del potenciómetro y el sensor de ultrasonidos.

Por último tenemos la estructura donde se recogen los datos sobre la acción de control en los 5 actuadores y su posicionamiento previo. El entero *datos.a[i].actuador* guardará un código que indique si el actuador *i* está inactivo o si está en modo control o excitación. El entero *datos.a[i].control* indica el tipo de control aplicado en el caso de que el actuador esté en modo control. Finalmente, los tres datos flotantes serán las coordenadas en los tres ejes de cada actuador en el caso de decidir hacer posicionamiento previo de éstos.

Recepción de la trama de inicialización:

La trama de inicio creada por el usuario es recibida por el Hilo21

```
void* Hilo21(void*P)//hilo que recibe datos de tierra (config),
    abre semaforo cuando llegan
{
    while (1)
    {
        int sockfd;
        struct sockaddr_in my_addr; // información sobre mi
            dirección
        struct sockaddr_in their_addr; // información sobre la
            dirección del cliente
        int addr_len, numbytes1;

        if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        {
            perror("socket");
            env_errores(26);
            //exit(1);
        }

        my_addr.sin_family = AF_INET; // Ordenación de bytes
        my_addr.sin_port = htons(MYPORT1); // short, Ordenación de
             bytes de la red
        my_addr.sin_addr.s_addr = inet_addr(MYDIR); // rellenar
            con mi dirección IP//my_addr.sin_addr.s_addr =
            inet_addr("10.12.110.57");
        memset(&(my_addr.sin_zero), '\0', 8); // poner a cero el
            resto de la estructura

        if (bind(sockfd, (struct sockaddr *)&my_addr,sizeof(struct
             sockaddr)) == -1)
        {
            perror("bind");
            env_errores(27);
            //exit(1);
        }
...
```

```
...
        addr_len = sizeof(struct sockaddr);

    //Introduce la trama recibida en la estructura 'datos'.
        if ((numbytes1=recvfrom(sockfd,&datos, sizeof(datos), 0,
            (struct sockaddr *)&their_addr, &addr_len)) == -1)
        {
            perror("recvfrom");
            env_errores(28);
            //exit(1);
        }

        n_control=1;//ya que no ha sido modificado (n_control=1
            indica que ha llegado config)

        sem_post(&Semaforo1);//abro el semaforo
        close(sockfd);

    }//fin while
}//fin hilolector
```

Código fuente 4.3: Código del hilo de recepción de trama de inicialización en C.

4.2.2. Código Matlab en ordenador de tierra.

Variables de estado: en la aplicación de tierra es necesario aplicar un cierto control de ejecución para proteger al sistema de un uso incorrecto o a destiempo de sus funciones, como por ejemplo enviar una nueva trama de inicialización cuando los experimentos ya se han iniciado, o intentar comenzar a grabar datos en cola cuando las interrupciones no han sido arrancadas. Uno de los elementos utilizados para implementar esta protección será la definición de ciertas variables globales de estado que serán chequeadas por el sistema antes de ejecutar cualquier función. Estas son:

Estado: variable global que proporciona información sobre el estado en el que se encuentran los experimentos.	
Estado=0	Ninguna configuración de inicialización enviada. Aún no se pueden iniciar las interrupciones del experimento.
Estado=1	Trama de inicialización ya enviada. Se pueden arrancar las interrupciones.
Estado=2 .	Se han comenzado las interrupciones. No se pueden enviar más tramas de inicialización. Los datos de los sensores no se están guardando en cola.
Estado=3	Interrupciones del experimento corriendo y registrando datos en cola.

Import: variable global sobre el estado en el que se encuentra la importación de un archivo de datos generado tras un experimento.	
Import=0	No se ha puesto en marcha ningún proceso de importación de archivo.
Import=1	Se ha abortado el proceso de importación.
Import=2	Importación de archivo de datos en marcha.

Variable globales de control: para este módulo se inicializan diversas variables globales para el control de los experimentos:

'ndatoscola'	Se almacena el número de datos que se tendrán que introducir en la cola según cómo lo haya configurado el usuario.
'ndatosred'	Almacena el número de datos de sensores que se enviarán en tiempo real
'mascara' .	Vector de (1x7) formado por ceros y unos. Sirve de máscara en la recepción y representación de los datos enviados en un archivo tras los experimentos.
'red' .	Vector de (1x7) formado por ceros y unos servirá de máscara en la recepción y y representación de los datos enviados en tiempo real.

mascara=[tiempo de captura, número de interrupción, pitch, roll, yaw, potenciómetro, ultrasonidos]

red=[tiempo de captura, número de interrupción, pitch, roll, yaw, potenciómetro, ultrasonidos]

Para cada sensor cuyo valor vaya a ser capturado y enviado, bien a la cola cíclica bien al ordenador de tierra, se pondrá el valor de su fila correspondiente a '1'. La utilización de los elementos de estos dos vectores hará más fácil la recepción y correcto almacenamiento y representación de los datos de los sensores desde tierra. Esto se detallará en los módulos de manipulación de datos y archivos, y de representación de datos.

Tanto estas variables de estado globales como las tramas para la inicialización serán definidas dentro de la función ***OpeningFcn*** de la ventana principal de la aplicación. Esta función es la primera en ejecutarse justo después de que todos los componentes de la figura hayan sido creados y justo antes de que la ventana se haga visible al usuario, con lo cual suele ser en ***OpeningFcn*** donde se definen las variables globales y se importan datos de otros archivos.

Estructura de la trama de inicialización:

En Matlab es posible definir estructuras de forma muy similar al C, sin embargo hay dos peculiaridades a tener en cuenta:

- No se puede hacer definición por bits en las estructuras. La unidad mínima es el octeto.

- Es complejo trabajar con estructuras heterogéneas e incluir, por ejemplo, enteros y flotantes en una misma estructura genera diversos errores en la manipulación de los datos.

Para salvar el primer punto, vemos que la utilización de los campos de bit en C se hace dentro de una estructura entera (cuatro octetos), de manera que en el caso de *datos.umi[i]* y *datos.ad[i]* lo que se manipula son los tres primeros bits del octeto menos significativo del entero reservado para el elemento *i*. Así pues optamos por definir las estructuras de la UMI y de los sensores AD directamente como enteros de 32 bits, asignándole directamente valores enteros comprendidos entre 0 y 7.

Para solucionar el problema de las estructuras heterogéneas en Matlab, se decide definir para los actuadores una estructura de datos flotantes. Será en la placa Hércules tras la recepción de la trama, donde se realizará una conversión de tipos sobre *datos.a[i].actuador* y *datos.a[i].control.*

```
...
% Se crea la estructura completa de datos de configuración y de los sensores.
% del experimento en la Hércules y se inicializa a 0.
handles.datos=struct('do',int32(0),'t_cola', int32(0),'frec', int32(0),'umi',...
...struct('umi',uint32(0)),'ad',struct('ad',uint32(0)));
handles.datos.umi(2).umi=uint32(0);
handles.datos.umi(3).umi=uint32(0);
```

```
handles.datos.ad(2).ad=uint32(0);
handles.mascara=[1,1,0,0,0,0,0];
handles.red=[1,1,0,0,0,0,0];
handles.ndatoscola=2;
handles.disponible=0;
handles.ndatosred=0;
setappdata(gcf,'stopred',0);
setappdata(gcf,'mascarared',handles.red);
setappdata(gcf,'datosred',0);
handles.monitor=0;
 % Se crea una estructura de actuadores homogénea de flotantes para hacer
 % más fácil su envío, y se inicializa a 0.
handles.a=struct('actuador',single(0),'control',single(0),'p1',single(0),'p2',single(0),'p3',single(0));
i=2;
while i<6
        handles.a(i).actuador=single(0);
        handles.a(i).control=single(0);
        handles.a(i).p1=single(0);
        handles.a(i).p2=single(0);
        handles.a(i).p3=single(0);
        i=i+1;
end
...
```

Código fuente 4.4: Código Matlab de definición de estructuras de inicialización.

Para la modificación de los valores de las tramas utilizamos campos de edición de texto para el tamaño de la cola, la frecuencia de interrupciones y las coordenadas de posicionamiento previo de los actuadores. En el código de estos elementos es donde se implementan filtros para evitar que el usuario introduzca datos erróneos o fuera de rango.

Para programar el muestreo de los distintos sensores se utilizan 'RadioButtons' que proporcionan una forma muy visual de modificar dichos campos teniendo en cuenta las siguientes condiciones:

- Los datos de un sensor pueden ser muestreados sin que estos valores sean guardados en cola o enviados a tierra.

- La selección de las opciones de guardar en cola y enviar a tierra son excluyentes entre sí, y cualquiera de ellas implican que se seleccione la opción de muestreo de los datos.

Cuando cualquier 'RadioButton' es modificado se llama a la función ***actualizatrama()***, que se encarga de actualizar los valores pertinentes en la trama *handles.datos*:

```
function varargout=actualiza_trama(varargin)
    disp('Actualizando trama de inicialización...');
    euler=varargin2;
    pitch=euler(1);
    roll=euler(2);
    yaw=euler(3);
    estado=varargin1;
```

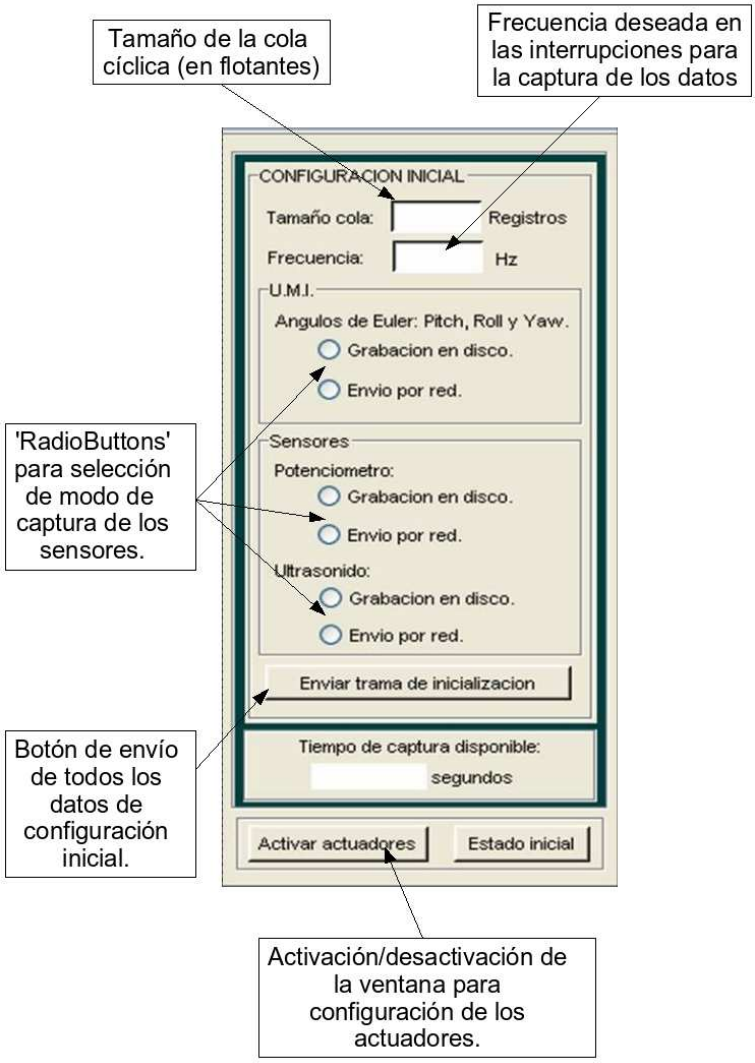


Figura 4.2: Ventana para configuración de parámetros de inicio en la captura de datos de los sensores.

```
disco=estado(1);
red=estado(2);
if (disco==1)|(red==1)
        % Pone muestreo a 1.
        pitch=bitset(pitch,1);
        roll=bitset(roll,1);
        yaw=bitset(yaw,1);
        if disco==1
            pitch=bitset(pitch,2);
            roll=bitset(roll,2);
            yaw=bitset(yaw,2);
            if red==2
                pitch=bitset(pitch,3,0);
                roll=bitset(roll,3,0);
                yaw=bitset(yaw,3,0);
```

```
                end
            end
            if red==1
                pitch=bitset(pitch,3);
                roll=bitset(roll,3);
                yaw=bitset(yaw,3);
                if disco==2
                    pitch=bitset(pitch,2,0);
                    roll=bitset(roll,2,0);
                    yaw=bitset(yaw,2,0);
                end
            end
        end
        if estado==[0,2]|estado==[2,0]
            % Pone muestreo a 0.
            pitch=bitset(pitch,1,0);
            roll=bitset(roll,1,0);
            yaw=bitset(yaw,1,0);
            if disco==0
                pitch=bitset(pitch,2,0);
                roll=bitset(roll,2,0);
                yaw=bitset(yaw,2,0);
            end
            if red==0
                pitch=bitset(pitch,3,0);
                roll=bitset(roll,3,0);
                yaw=bitset(yaw,3,0);
            end
    end
```

Código fuente 4.5: Función Matlab actualizatrama() para construcción de la trama de inicialización.

Para la creación de la trama de actuadores utilizamos una ventana específica que por defecto estará desactivada. En ella, a través de 'ListBoxes' se podrá seleccionar si el actuador permanece inactivo, si está en modo 'Control' o si se aplica una excitación. Para la opción de control, cada actuador puede optar por varios modos: P, PI, PID, etc. La opción de excitación queda planteada para completar su implementación en el futuro.

Envío de las tramas de inicialización:

Cuando se tienen configurados todos los parámetros que conforman las tramas de inicio, en el botón 'Enviar trama de inicialización' se ejecuta un código que lleva a cabo los siguientes pasos:

1. Introduce en dos vectores distintos (A y B) la trama de parámetros del experimento y la trama de los actuadores.

2. Actualiza los valores de las variables globales *ndatoscola*, *ndatosred*, *red* y *mascara*.

3. Se realiza el envío consecutivo de ambas tramas por el puerto 4951 para que sea recepcionada por el Hilo21 en el computador de vuelo. Ambas tramas se han de enviar por separado porque en Matlab la función de envío a través de un objeto udp ha de especificar el formato de los

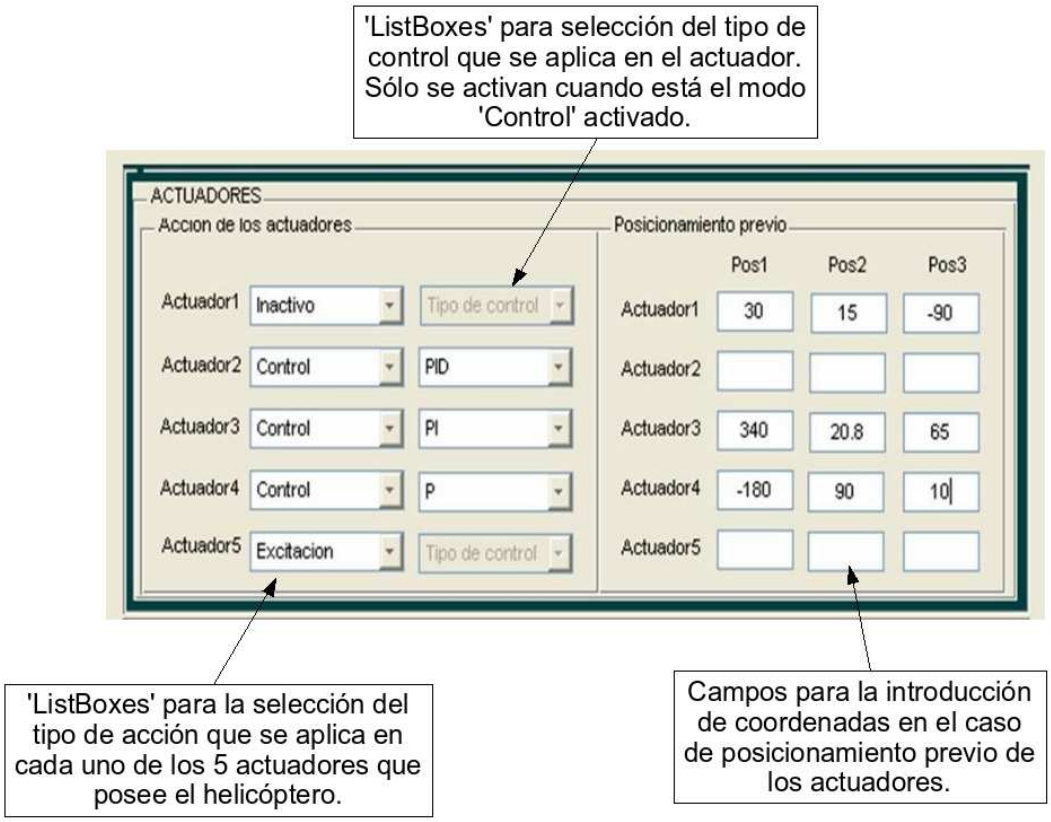


Figura 4.3: Ventana para configuración de parámetros de los actuadores del sistema.

elementos enviados, y la opción más sencilla es enviar primero la trama homogénea de enteros de 32 bits diseñada para los datos de sensores y después la trama de flotantes simples construida para los actuadores.

```
...
% Hacemos dos envíos antes de considerar que hay un fallo
% importante en la línea.
n_envio=0;
while n_envio<2
        % Se abre el socket UDP por puerto 4951.
        t=udp(getappdata(gcf,'ipserver'),getappdata(gcf,'p1'));
        set(t,'ByteOrder','littleEndian');
        fopen(t);
        % Se envía la trama de datos de sensores A.
        fwrite(t,A,'int32');
        % Se envía la trama de parametros de actuadores B.
        fwrite(t,B,'float32');
        n_envio=n_envio+1;
```

```
            disp('Recibo confirmación de trama de configuración:');
            [A,count]=fread(t, 1,'int32')
            fclose(t);
            delete(t);
            clear t;
    ...
```

Código fuente 4.6: Envío en Matlab de las estructuras para inicialización de experimentos en el helicóptero.

v

4. Recepción de un código de confirmación desde el computador de vuelo. Este código informará de la recepción correcta de las tramas o de diversas situaciones de error que han llevado a la no inicialización de la placa.

```
    ...
    elseif (A==1)
            n_envio=2;
            % Se pone 'Estado'=1 si se ha enviado la trama de inicio con éxito.
            setappdata(gcf,'Estado',1);
            frec=handles.datos.frec;
            ndatos=handles.ndatoscola;
            tam=handles.datos.t_cola;
            dint=frec*ndatos;
            tdisp=single(tam)/single(dint);
            % Se representa el tiempo disponible para captura en cola.
            set(handles.text12,'string',num2str(tdisp));
            guidata(hObject,handles);
            set(handles.text34,'string','Placa inicializada y a la espera de instrucciones para
experimentos.');
    ...
```

Código fuente 4.7: Indicación a la aplicación de tierra de recepción correcta de la trama de inicialización.

5. Si la placa ha sido inicializada con éxito, se modifica *Estado* y se calcula el tiempo de registro disponible para los experimentos antes de que la cola cíclica se llene y comience a sobreescribir los datos.

4.2.3. Modificaciones en el código del computador de vuelo.

El hecho de que la aplicación de tierra envíe la trama de inicialización dividida en dos subtramas conlleva una primera modificación en el código C, el cual ha de recibir la trama de flotantes correspondiente a los actuadores e introducir sus valores en la estructura global *datos.a[i]* haciendo las conversiones de tipo pertinentes. Ésto se realiza en el código del Hilo21.

```
...
        if ((numbytes1=recvfrom(sockfd,&datos, sizeof(datos), 0,
            (struct sockaddr *)&their_addr, &addr_len)) == -1)
        {
            perror("recvfrom");
            env_errores(28);
            //exit(1);
        }
        if ((numbytes1=recvfrom(sockfd,&actuadores, sizeof(
            actuadores), 0,
            (struct sockaddr *)&their_addr, &addr_len)) == -1)
        {
            perror("recvfrom");
            env_errores(28);
            //exit(1);
        }

        printf("Recibimos trama de actuadores.\n");
        for (i=0,j=0;i<=5;i++,j+=5)
        {
            datos.a[i].actuador=(int)(actuadores[j]);
            datos.a[i].control=(int)(actuadores[j+1]);
            datos.a[i].p1=actuadores[j+2];
            datos.a[i].p2=actuadores[j+3];
            datos.a[i].p3=actuadores[j+4];
        }
        for (i=0;i<5;i++)
        {
            printf("datos.a(%d).actuador=%d,",i,datos.a[i].actuador
                );
            printf("datos.a(%d).control=%d,",i,datos.a[i].control);
            printf("datos.a(%d).p1=%f,",i,datos.a[i].p1);
            printf("datos.a(%d).p2=%f,",i,datos.a[i].p2);
            printf("datos.a(%d).p3=%f\n\n",i,datos.a[i].p3);
        }
...
```

Código fuente 4.8: Código adicional C en Hilo21 para la recepción de la trama de actuadores e introducción en estructura 'datos'.

Por otra parte, uno de los objetivos de este proyecto es conseguir que la aplicación de tierra sea consciente en la medida de lo posible del estado del ordenador de vuelo. Así pues, conocer si una orden de tierra ha sido llevada a cabo con éxito o no implica una comunicación bidireccional en cada

directiva, esto es, que la placa Hércules habrá de enviar un código de confirmación o información cada vez que los hilos Hilo21 e Hilo22 reciban algo en sus puertos.

En el caso del módulo de inicialización, cuando el Hilo21 recibe la trama abre el semáforo para que el bucle `switch(ncontrol)` llame a la función ***ini()***. Sin embargo el hilo se queda esperando hasta que ***ini()*** active una nueva variable global llamada *piloto0* o bien hasta que se agote un tiempo de espera dado por la constante MARGENT. Si ***ini()*** asigna un valor a *piloto0* mayor que 0, el hilo envía dicho valor como confirmación al ordenador de tierra. Si se ha agotado el tiempo MARGENT sin que se modifique *piloto0*, se envía un valor muy elevado que el control de tierra interpretará como un error de recepción.

```
...
   sem_post(&Semaforo1);//abro el semaforo
//Indica a tierra si ha habido errores.
   j=0;
   while (j<=MARGENT)//MARGENT nos da un margen de tiempo
      ponderado de espera.
   {
      j++;
      if (piloto0>0)
      {
         j=MARGENT+100;
         if ((numbytes1=sendto(sockfd,&piloto0, sizeof(int), 0,(
            struct sockaddr *)&their_addr, addr_len)) == -1)
         {
            perror("sendto");
            env_errores(38);
            printf("Hay un error en sendto");
         }
      }
   }
   if (j==MARGENT)//Si se agota el tiempo, indicaremos a tierra
      que no se ha recibido la trama de inicialización.
   {
      if ((numbytes1=sendto(sockfd,&j, sizeof(int), 0,
         (struct sockaddr *)&their_addr, addr_len)) == -1)
      {
         perror("recvfrom");
         env_errores(39);
         printf("Hay un error en sendto");
      }
   }
   close(sockfd);
...
```

Código fuente 4.9: Código adicional C en Hilo21 para el envío de una confirmación de recepción de trama de inicio.

4.3. Módulo de configuración de datos de red.

Los datos referentes a la dirección IP del computador de vuelo y los puertos relativos a cada canal de comunicación con él pueden ser modificados por el usuario si se da la necesidad. La aplicación de tierra tiene que estar diseñada para proporcionar una forma sencilla de acceder a estos datos y modificarlos en un sólo sitio, sin tener que cambiar dichos datos en cada uno de los 'callbacks' * que los utilicen.

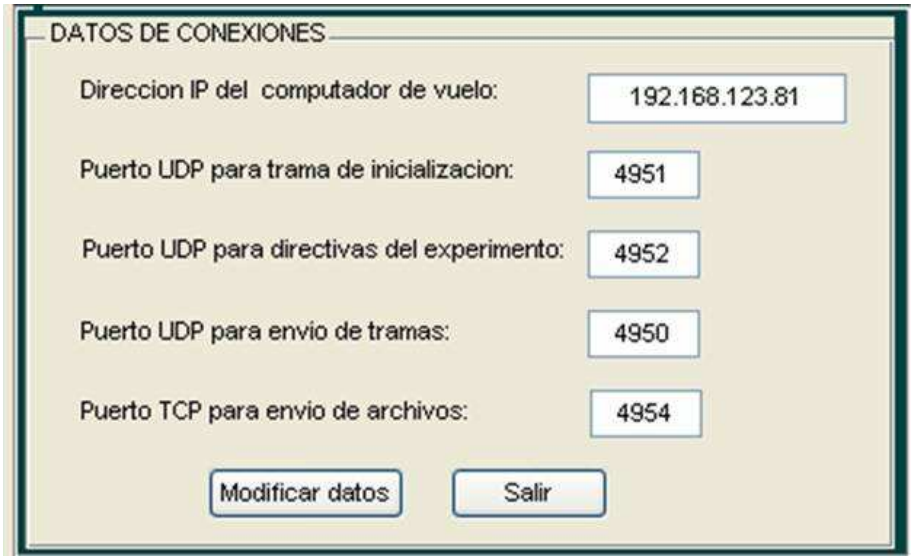


Figura 4.4: Ventana gráfica para visualización y modificación de parámetros de conexión (Conf.conexion.fig).

La visualización y modificación de los parámetros de red se hará en una ventana específica para ello. Para que estos datos sean accesibles desde distintas ventanas de la aplicación, deben ser guardadas en unas variables globales denominadas 'application data' (appdata). Estos appdata serán inicializados también en la función de carga de la ventana principal ***OpeningFcn***.

```
% Conexiones.mat será el archivo donde se guardan datos básicos de configuración de
%la aplicación: puertos, IP, etc.
if exist('conexiones.mat')=2
        data=load('conexiones.mat');
        setappdata(gcf,'ipserver',data.servidor);
        setappdata(gcf,'p1',data.puerto1);
        setappdata(gcf,'p2',data.puerto2);
        setappdata(gcf,'p3',data.puerto3);
        setappdata(gcf,'p5',data.puerto5);
        setappdata(gcf,'lastfile',data.file);
        setappdata(gcf,'pathfile',data.path);
else
% Datos de aplicación para la configuración de la conexión con la Hércules.
        setappdata(gcf,'ipserver','192.168.32.1');
        setappdata(gcf,'p1',4951);
```

*'Callback': función que se ejecuta cuando se activa un objeto de la ventana, como por ejemplo un botón.

```
        setappdata(gcf,'p2',4952);
        setappdata(gcf,'p3',4950);
        setappdata(gcf,'p5',4954);
        setappdata(gcf,'lastfile',0);
        setappdata(gcf,'pathfile',0);
    end
```

Código fuente 4.10: Código Matlab para la carga inicial de los parámetros principales de conexión.

Para que las modificaciones en estas variables de red sean permanentes y los cambios se vuelvan a cargar al volver a encender la aplicación, se define un archivo matlab ***conexiones.mat*** donde se pueden almacenar todas las variables del espacio de trabajo que se consideren necesarias. Así pues, al abrir la aplicación se carga su contenido mediante la orden `load` en la estructura *data* que será utilizada para inicializar los distintos appdata que servirán, entre otras cosas, para establecer las conexiones de red.

Cuando se carga la ventana 'Confconexion.fig', se almacenan los appdata en variables locales a dicha figura para poder representarlos:

```
% Se extraen los datos de la ip y los puertos de los appdata.
% Se convierten a strings y se representan en los campos de texto.
handles.ip =getappdata(Vprincipal,'ipserver');
set(handles.edit1,'string',handles.ip);
handles.port21 = getappdata(Vprincipal,'p1')
handles.port21=num2str(handles.port21)
set(handles.edit2,'string',handles.port21);
...
```

En el 'callback' del botón para modificar los datos, se comprueba si ha habido modificaciones y se cargan los cambios de nuevo en los appdata:

```
function pushbutton1_Callback(hObject, eventdata, handles)
    if (handles.modificado==1)
        if (handles.mascara(1)==1)
            setappdata(V_principal,'ipserver',handles.ip);
        end
        if (handles.mascara(2)==1)
            setappdata(V_principal,'p1',str2num(handles.port21));
        end
        if (handles.mascara(3)==1)
            setappdata(V_principal,'p2',str2num(handles.port22));
        end
        if (handles.mascara(4)==1)
            setappdata(V_principal,'p3',str2num(handles.port23));
        end
        if (handles.mascara(5)==1)
            setappdata(V_principal,'p5',str2num(handles.port24));
        end
        h = guidata(V_principal);
```

```
            handles.mascara=[0,0,0,0,0];
    end
guidata(hObject, handles);
close(Conf_conexion);
```

Código fuente 4.11: Código Matlab para guardar modificaciones en los parámetros de conexión.

Una vez que se han guardado los cambios en los appdata de datos de red, la aplicación podrá seguir operando perfectamente con la nueva configuración de canales de comunicación. Sin embargo, antes de salir del programa se han de almacenar estos cambios en el archivo externo ***conexiones.mat*** mediante la orden `save`. Vemos a continuación la parte del código que realiza estas operaciones dentro del 'callback' de finalización de aplicación:

```
function pushbutton18_Callback(hObject, eventdata, handles)
    %Cargar en 'conexiones.mat' los valores antes de salir:'
    servidor=getappdata(gcf,'ipserver');
    puerto1= getappdata(gcf,'p1');
    puerto2= getappdata(gcf,'p2');
    puerto3= getappdata(gcf,'p3');
    puerto5= getappdata(gcf,'p5');
    file=getappdata(gcf,'lastfile')
    path=getappdata(gcf,'pathfile')
    save('conexiones.mat','servidor','puerto1','puerto2','puerto3',...
    'puerto5','file','path');
    ...
```

Código fuente 4.12: Carga en Matlab de los parámetros de conexión antes de salir de la aplicación.

4.4. Módulo de realización de experimentos.

4.4.1. Código C en computador de vuelo

Recepción de n_control y bucle central de ejecución.

Como se ha visto en el capítulo anterior, la gestión de los experimentos se centra en las siguientes opciones del bucle central `switch(n_control)`:

```
    printf("----------espero ordenes desde tierra
       ----------\n");
    while (!ter)
    {
       sem_wait(&Semaforo1);//espero al semaforo que recibe desde
           tierra
       switch (n_control)
       {
...
          case 2: printf("llega (arranca)\n"); parado=0; arranca
             (); pthread_cancel(Hilo_t_h1);break;//no permito la
             llegada de mas configuraciones una vez comenzado el
             proceso.
          case 3: printf("llega 3 (arranca+datos)\n"); parado=0;
             col=1; arranca(); pthread_cancel(Hilo_t_h1);break;
          case 4: printf("llega 4 (datos a cola)\n");col=1;break;
          case 5: printf("llega 5 (parar int)\n");parar();parado
             =1;col=0;break;
          case 6: printf("llega 6 (parar datos a cola)\n");col=0;
             break;
...
```

Código fuente 4.13: Código adicional C en Hilo21 para el envío de una confirmación de recepción de trama de inicio.

Tenemos seis combinaciones posibles: si *n_control* es 2 o 3 se arrancan las interrupciones mediante la función ***arranca()***. La diferencia entre introducir datos en la cola o no hacerlo es el valor de la variable global *col*. Las interrupciones serán detenidas si se recibe *n_control* igual a 6, que conlleva a la ejecución de la función ***parar()***.

Observamos que para las dos opciones en las cuales se inician las interrupciones en la placa, se desactiva el hilo21 mediante la orden `pthread_cancel`. Esto se hace precisamente para evitar que se envíen nuevas tramas de configuración una vez que ya se han comenzado los experimentos. Sin embargo, las consecuencias de ésto son que si al finalizar las pruebas en un experimento determinado, e importados ya los datos pertinentes, se desea realizar otro experimento con una nueva configuración de captura de datos o frecuencia de interrupciones se ha de reiniciar la aplicación del computador de vuelo para que se reactiven correctamente todos los hilos, y no resulta muy cómodo de solucionar.

Los distintos valores de *n_control* son recepcionados a través del hilo22 cuyo código se muestra a continuación:

```
void* Hilo22(void*P)//hilo que recibe datos de tierra (n_control
    ), abre semaforo cuando llegan
{
    while (1)
    {
        int sockfd;
        struct sockaddr_in my_addr; // información sobre mi
            dirección
        struct sockaddr_in their_addr; // información sobre la
            dirección del cliente
        int addr_len, numbytes2;

        if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        {
            perror("socket");
            env_errores(29);
            //exit(1);
        }
        my_addr.sin_family = AF_INET; // Ordenación de bytes
        my_addr.sin_port = htons(MYPORT2); // short, Ordenación de
             bytes de la red
        my_addr.sin_addr.s_addr = inet_addr(MYDIR); // rellenar
            con mi dirección IP//my_addr.sin_addr.s_addr =
            inet_addr("10.12.110.57");
        memset(&(my_addr.sin_zero), '\0', 8); // poner a cero el
            resto de la estructura
        if(bind(sockfd,(struct sockaddr *)&my_addr,sizeof(struct
            sockaddr)) == -1)
        {
            perror("bind");
            env_errores(30);
        }
        addr_len = sizeof(struct sockaddr);

        //Recibe un dato entero desde tierra que almacena en
            n_control.
        if ((numbytes2=recvfrom(sockfd,&n_control, sizeof(
            n_control), 0,
            (struct sockaddr *)&their_addr, &addr_len)) == -1)
        {
            perror("recvfrom");
            env_errores(31);
            //exit(1);
        }
        sem_post(&Semaforo1);//abro el semaforo
...
```

```
...
        close(sockfd);
    }//fin while
}//fin hilolector
```

Código fuente 4.14: Código del Hilo22 para recepción en C de directivas mediante n_control.

Código de adquisición de datos de altura.

Cuando se arrancan las interrupciones, con la frecuencia programada se invoca a la función ***sec()***, que se encargará de llamar secuencialmente a una función para aplicar la acción de control sobre los actuadores (***accion()***), a una función para capturar los datos de la UMI (***umi1()***), a una función para capturar el dato de la altura (***ad1()***) y por último a una función que envía datos en tiempo real (***env_d_red()***).

En la función de captura ***ad1()***sólo se realiza la captura de datos desde la entrada analógica, ya que en las condiciones en las que se desarrolló dicho software aún no se había contemplado la opción de añadir otro sensor para la altura. Existía sin embargo un nueva aplicación en C, desarrollada por un proyecto posterior, cuya misión específica era la captura de las alturas desde el potenciómetro y el sensor de ultrasonidos, en sus respectivas posiciones dentro de la plataforma de vuelo, y su representación en Matlab. El siguiente paso sería integrar este código de captura de datos del sensor de ultrasonidos desde la entrada digital y realizar las modificaciones necesarias en el código de captura desde la entrada analógica.

Para la aplicación original, en la función principal ***main()*** se hace una configuración inicial de los parámetros de captura por la entrada A/D:

```
...
    //============================================================
    //a-d
    //============================================================
    memset(&dscadsettings, 0, sizeof(DSCADSETTINGS));
    dscadsettings.range = RANGE_10; //Rango de la conversión A/D.
    dscadsettings.polarity = BIPOLAR; //Polaridad.
    dscadsettings.gain = GAIN_1; //Ganancia.
    dscadsettings.load_cal = (BYTE)TRUE; // La placa tomará de la
    // EEPROM los parámetros de calibración para el rango
    // seleccionado antes de comenzar la conversión A/D.
    dscadsettings.current_channel = 0; //El canal en el cual se
    // realiza la conversión.
    dscadsettings.scan_interval = SCAN_INTERVAL_9; //Tiempo entre
    //dos conversiones consecutivas.
    dscadsettings.addiff = (BYTE) intBuff;
...
```

```
...
    if( dscADSetSettings( dscb, &dscadsettings ) != DE_NONE )
    {
        dscGetLastError(&errorParams);
        fprintf( stderr, "dscADSetSettings error: %s %s\n",
            dscGetErrorString(errorParams.ErrCode), errorParams.
            errstring );
        env_errores(16);
        return 0;
    }

    //============================================================
    //a-d scan
    //============================================================
    dscadscan.low_channel = 0; //Primer canal dentro del rango de
        escaneo A/D.
    dscadscan.high_channel = N; //El último canal dentro del
        rango de escaneo.
    dscadscan.gain = GAIN_1; //Ganancia.

    //Reserva de memoria para muestras de cada canal.
    samples = (DSCSAMPLE*)malloc( sizeof(DSCSAMPLE) * ( dscadscan
        .high_channel - dscadscan.low_channel + 1 ) );

    if ( samples == NULL ) {
        fprintf( stderr, "Unable to allocate memory\n");
        env_errores(17);
        return 0;
    }
...
```

Código fuente 4.15: Código C al inicio de main() en el cual se configuran los parámetros para la captura A/D en la placa.

Las conversiones A/D están originalmente configuradas para que sean bipolares y se realicen a través del número de canales especificados en la variable global N. Dado que el objetivo es que únicamente uno de los sensores de altura sea leído por la entrada analógica esta parte del código habrá de ser modificada, al igual que la función ***ad1()*** que se muestra a continuación:

```
ad1 (void)
{
    int i;
    DFLOAT voltage;
    //============================================================
    //a-d scan
    //============================================================
...
```

```
ad1 (void)
{
    int i;
    DFLOAT voltage;
    //============================================================
    //a-d scan
    //============================================================
    //Introduce en 'samples' lo leído por cada canal.
    if( dscADScan( dscb, &dscadscan, samples ) != DE_NONE )
       {
          dscGetLastError(&errorParams);
          fprintf( stderr, "dscADScan error: %s %s\n",
             dscGetErrorString(errorParams.ErrCode), errorParams.
             errstring );
          free( samples ); // remember to deallocate malloc()
             memory
          //return 0;
          env_errores(1);
       }

       //printf( "\nSample readouts:" );
       for( i = 0; i < (dscadscan.high_channel - dscadscan.
          low_channel)+ 1; i++)
          printf( " %d", dscadscan.sample_values[i] );

       //printf( "\nActual voltages:" );
       for( i = 0; i < (dscadscan.high_channel - dscadscan.
          low_channel)+ 1; i++)
       {
          //Convierte a voltaje los datos leídos.
          if( dscADCodeToVoltage(dscb, dscadsettings, dscadscan.
             sample_values[i], &voltage) != DE_NONE)
          {
             dscGetLastError(&errorParams);
             fprintf( stderr, "dscADCodeToVoltage error: %s %s\n"
                , dscGetErrorString(errorParams.ErrCode),
                errorParams.errstring );
             //return 0;
             env_errores(2);
          }
          printf( "%5.3lfV", voltage);
       }
    printf("\n");
    //Codigo que escribe en la cola:
...
```

Código fuente 4.16: Código C de la función ad1() de captura de datos por la entrada analógica (potenciómetro).

En la función se muestrean los datos por cada canal y se hace la conversión a voltaje de cada uno de ellos.

4.4.2. Código Matlab en ordenador de tierra.

Se diseña un cuadro gráfico que ha de cubrir las cinco posibilidades existentes para la gestión de los experimentos, esto es *n_control* entre 2 y 6.

Figura 4.5: Cuadro para gestión de la realización de los experimentos.

El código perteneciente al botón de iniciar interrupciones sería:

```
function pushbutton2_Callback(hObject, eventdata, handles)
    if ((getappdata(gcf,'Estado'))==1)
            if(get(handles.checkbox1,'Value'))
             % Comenzamos a grabar datos desde el principio.
                 n_control=3;
            else
             % Interrupciones sin grabar datos.
                 n_control=2;
            end
    else
            errordlg('Trama de inicialización sin enviar al servidor','Error');
            return
    end
    t=udp(getappdata(gcf,'ipserver'),getappdata(gcf,'p2'));
    set(t,'ByteOrder','littleEndian');
    set(t,'DatagramTerminateMode','off');
    set(t,'Terminator',0)
    fopen(t);
    fwrite(t,n_control,'int32');
    disp('Recibo confirmación de inicio de interrupciones:');
    [A,count]=fread(t, 1,'int32')
     % Confirmación de inicio de interrupciones con éxito.
    if A==1
            % Se enciende el piloto de interrupciones.
            set(handles.text15,'string','En marcha');
            set(handles.uipanel16,'BackgroundColor',[0.545,0.804,0.196]);
            if (n_control==2)
                set(handles.text34,'string','Interrupciones en proceso en la placa.');
            % Cambiamos el estado del sistema.
```

```
                setappdata(gcf,'Estado',2);
            elseif (n_control==3)
                set(handles.text16,'string','En marcha');
                set(handles.uipanel17,'BackgroundColor',[0.545,0.804,0.196]);
                 set(handles.text34,'string','Interrupciones en proceso y guardando datos en
cola.');
                setappdata(gcf,'Estado',3);
            end
             % Desactivamos el cuadro de envío de tramas de inicialización.
            set(handles.edit2,'Enable','off');
            set(handles.edit3,'Enable','off');
            set(handles.radiobutton1,'Enable','off');
            set(handles.radiobutton2,'Enable','off');
            set(handles.radiobutton3,'Enable','off');
            set(handles.radiobutton4,'Enable','off');
            set(handles.radiobutton5,'Enable','off');
            set(handles.radiobutton6,'Enable','off');
            set(handles.pushbutton1,'Enable','off');
            guidata(hObject,handles);
        else
             set(handles.text34,'string','ERROR: las interrupciones de usuario no se han podido
iniciar en la placa.');
            set(handles.text15,'string','Detenido');
            set(handles.uipanel16,'BackgroundColor',[1,1,0.502]);
        end
        fclose(t);
        delete(t);
        clear t;
```

Código fuente 4.17: Código Matlab para inicio de interrupciones en la placa Hércules.

En este callback se envía por el socket (puerto udp 4952) *n_control* igual a 2 o a 3 según si el 'CheckBox' de grabación de datos desde el principio se activa o no. Espera una confirmación desde la placa Hércules, y si es correcta activa los pilotos correspondientes, modifica *Estado* y deshabilita el cuadro de envío de tramas de inicialización. Así pues, protegemos desde tierra al sistema de un envío a destiempo de este tipo de tramas, pudiendo evitar la desactivación del Hilo21 en el código del computador de vuelo.

En el código perteneciente al botón de iniciar introducción de datos en cola se envía *n_control*=4 y se espera confirmación del computador de vuelo. Si la confirmación es positiva se cambia *Estado* y se enciende el piloto:

```
function pushbutton4_Callback(hObject, eventdata, handles)
    if ((getappdata(gcf,'Estado'))==2)
            n_control=4;
            t=udp(getappdata(gcf,'ipserver'),getappdata(gcf,'p2'));
            set(t,'ByteOrder','littleEndian');
            fopen(t);
```

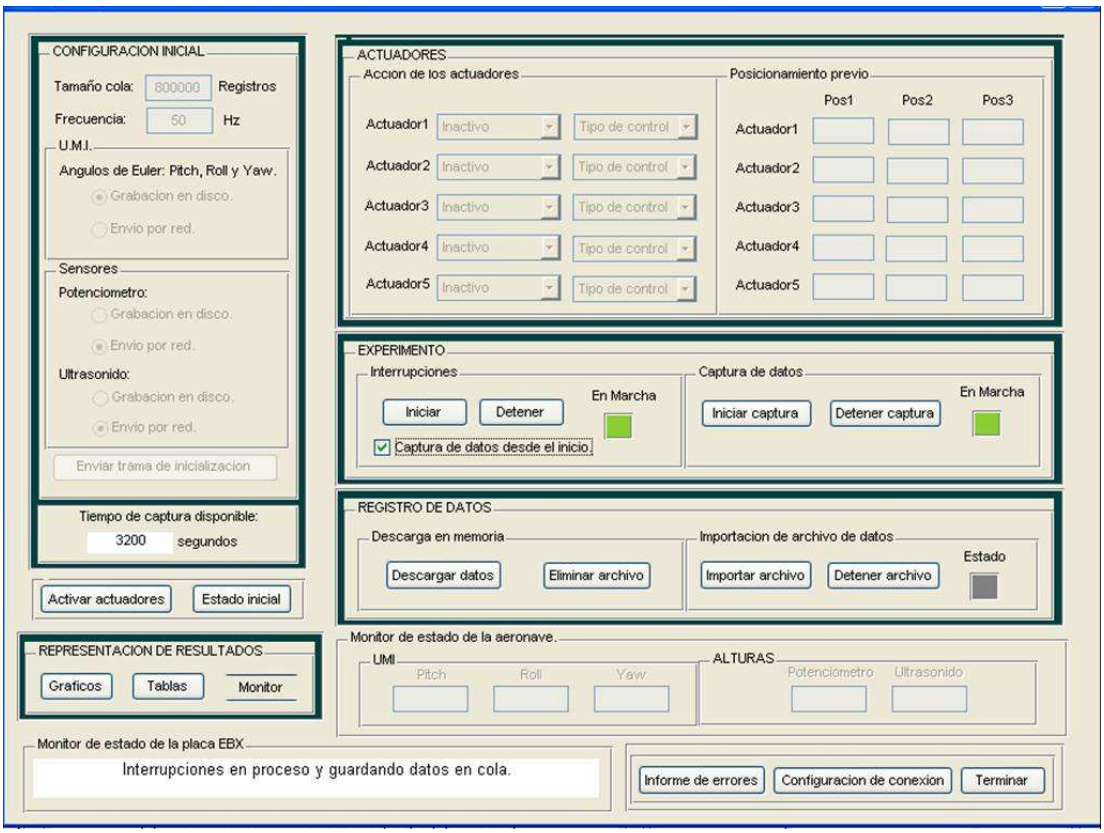


Figura 4.6: Al iniciar las interrupciones con éxito en el computador de vuelo se desactiva el cuadro de envío de tramas de inicialización.

```
        fwrite(t,n_control,'int32');
        disp('Recibo confirmación de captura de datos en cola:');
        [A,count]=fread(t, 1,'int32')
        fclose(t);
        delete(t);
        clear t;
        % Confirmación de grabación de datos correcta.
        if (A==1)
            setappdata(gcf,'Estado',3);
            set(handles.text16,'string','En marcha');
            set(handles.uipanel17,'BackgroundColor',[0.545,0.804,0.196]);
            set(handles.text34,'string','Captura de datos de sensores en proceso.');
        else
            set(handles.uipanel17,'BackgroundColor',[1,1,0.502]);
            set(handles.text16,'string','Detenido');
            set(handles.text34,'string','Error: imposible iniciar captura de datos.');
        end
elseif ((getappdata(gcf,'Estado'))==3)
        errordlg('Captura de datos ya en proceso','Error');
elseif ((getappdata(gcf,'Estado'))<=1)
        errordlg('Iniciar primero interrupciones en servidor','Error');
end
```

```
guidata(hObject,handles);
```

Código fuente 4.18: Código Matlab para inicio de introducción de datos de sensores en cola.

En la función correspondiente al botón para detener las interrupciones, la confirmación esperada es una cantidad distinta de -1, que dará información sobre el número de registros aún disponibles en la cola cíclica tras detener la introducción de los datos. Con este valor hará los cálculos para representar los segundos aún disponibles para grabar datos de sensores en las condiciones actuales del experimento:

```
function pushbutton3_Callback(hObject, eventdata, handles)
    if ((getappdata(gcf,'Estado'))>=2)
            n_control=5;
            t=udp(getappdata(gcf,'ipserver'),getappdata(gcf,'p2'));
            set(t,'ByteOrder','littleEndian');
            set(t,'DatagramTerminateMode','off');
            set(t,'Timeout',20);
            set(t,'Terminator',0)
            fopen(t);
            fwrite(t,n_control,'int32');
            disp('Recibo confirmación de detención de interrupciones:');
            [A,count]=fread(t, 1,'int32');
             % Si se ha recibido algún elemento por el puerto.
            if (count>0)
                if (A==-1)
                    set(handles.text34,'string','Error: las interrupciones no se han detenido con
éxito.');
                else
                    handles.disponible=single(A);
                    set(handles.uipanel16,'BackgroundColor',[1,1,0.502]);
                    set(handles.text15,'string','Detenido');
                    if((get(handles.uipanel17,'BackgroundColor'))==[0.545,0.804,0.196])
                        set(handles.uipanel17,'BackgroundColor',[1,1,0.502]);
                        set(handles.text16,'string','Detenido');
                    end
                     set(handles.text34,'string','Interrupciones detenidas. Helicóptero en espe-
ra');
                    setappdata(gcf,'stopred',1);
                 % Se calcula el nuevo tiempo disponible de registro de datos de sensores.
                    frec=handles.datos.frec;
                    ndatos=handles.ndatoscola;
                    dint=frec*ndatos;
                    tdisp=single(A)/single(dint);
                    set(handles.text12,'string',num2str(tdisp));
                    setappdata(gcf,'Estado',1);
                    guidata(hObject,handles);
                end
            else
```

```
                set(handles.text34,'string','Error: sobrepasado el tiempo de confirmación. No hay
información');
            end
            fclose(t);
            delete(t);
            clear t;
        else
            errordlg('Ningún experimento corriendo.','Error');
        end
```

Código fuente 4.19: Código Matlab para detención de interrupciones en la placa Hércules.

Por último tenemos el código del botón donde únicamente se detiene la grabación de los datos de los sensores en la cola. Éste callback es idéntico al anterior en el que se envía *n_control*=6 y se espera la cantidad de posiciones disponibles en cola para registros como confirmación desde tierra.

4.4.3. Modificaciones en el código del computador de vuelo.

Recepción de n_control y bucle central de ejecución.

Tenemos que en el bucle central de ***main()*** se elimina la cancelación del Hilo21 para *n_control* igual a 2 o a 3. A su vez, al Hilo22 se le añade el código necesario para poder mandar las confirmaciones a tierra en cada caso:

```
...
    if ((numbytes2=recvfrom(sockfd,&n_control, sizeof(n_control),
        0,
        (struct sockaddr *)&their_addr, &addr_len)) == -1)
    {
        perror("recvfrom");
        env_errores(31);
    }
...
    sem_post(&Semaforo1);//abro el semáforo
//En el caso de iniciar interrupciones, si se hace con éxito se
    indica a Matlab para encender piloto1.
    if ((n_control==2)||(n_control==3))
    {
        j=0;
...
```

```
...
        while (j<=MARGENT)//MARGENT nos da un margen de tiempo
           ponderado de espera.
        {
           j++;
           if (piloto1)//Si la función arranca() tiene éxito
              activa piloto1.
           {
              j=MARGENT+100;
              if ((numbytes2=sendto(sockfd,&piloto1, sizeof(int),
                 0,
                 (struct sockaddr *)&their_addr, addr_len)) == -1)
              {
                 perror("sendto");
                 env_errores(48);
              }
           }
        }
        if (j==MARGENT)//Si se agota el tiempo, indicaremos a
           Matlab que no se han iniciado las interrupciones.
        {
           printf("Se ha sobrepasado el tiempo\n");
           if ((numbytes2=sendto(sockfd,&j, sizeof(int), 0,
              (struct sockaddr *)&their_addr, addr_len)) == -1)
           {
              perror("recvfrom");
              env_errores(49);
              printf("Hay un error en sendto");
           }
        }
     }
  //En el caso de iniciar el registro de datos en cola, se indica
     a Matlab si debe encender piloto.
     if ((n_control==3)||(n_control==4))
     {
        j=0;
        while (j<=MARGENT)//MARGENT nos da un margen de tiempo
           ponderado de espera.
        {
           j=j+1;
           if (col)//Se activa el flag de introducir datos en cola
              .
           {
              printf("Enviamos col en %d iteracciones\n",j);
...
```

```
...
            if ((numbytes2=sendto(sockfd,&col, sizeof(int), 0,
               (struct sockaddr *)&their_addr, addr_len)) == -1)
            {
               perror("recvfrom");
               env_errores(50);
               printf("Hay un error en sendto\n");
               //exit(1);
            }
         j=MARGENT+100;
         }
      }
      if (j==MARGENT)//Si se agota el tiempo, indicaremos a
         Matlab que no se han detenido las interrupciones.
      {
         printf("Se ha sobrepasado el tiempo\n");
         if ((numbytes2=sendto(sockfd,&j, sizeof(int), 0,
            (struct sockaddr *)&their_addr, addr_len)) == -1)
         {
            perror("recvfrom");
            env_errores(49);
            printf("Hay un error en sendto");
         }
      }
   }
//Se envía al ordenador de tierra la memoria disponible antes de
    machacar datos en cola.
   if ((n_control==5)||(n_control==6))
   {
      j=0;
      while (j<=MARGENT)//MARGENT nos da un margen de tiempo
         ponderado de espera.
      {
         j=j+1;
         if (!col)//Ya no van a variar los índices de la cola.
         {
            mem_chec();
            printf("enviamos la memoria disponible por la red:%d
               \n",mem_disp);
            if ((numbytes2=sendto(sockfd,&mem_disp, sizeof(int),
                0,
               (struct sockaddr *)&their_addr, addr_len)) == -1)
            {
               perror("recvfrom");
               env_errores(51);
               printf("Hay un error en sendto\n");
            }
         j=MARGENT+100;
         }
      }
...
```

```
...
        if (j==MARGENT) //Si se agota el tiempo, indicaremos a
            Matlab que no se han detenido las interrupciones.
        {
            printf("Se ha sobrepasado el tiempo\n");
            j=-1;
            if ((numbytes2=sendto(sockfd,&j, sizeof(int), 0,
                (struct sockaddr *)&their_addr, addr_len)) == -1)
            {
                perror("recvfrom");
                env_errores(49);
                printf("Hay un error en sendto");
            }
        }
    }
...
```

Código fuente 4.20: Código C adicional en Hilo22 para envío de confirmaciones en la gestión de los experimentos.

Para el caso de inicio de interrupciones se espera un tiempo dado por la constante MARGENT. Si dentro de éste la función ***arranca()*** inicia las interrupciones con éxito y activa la variable global *piloto1*=1, se envía éste como confirmación a tierra. Si se sobrepasa el tiempo se envía MARGENT, que es un valor muy elevado. Para el caso de iniciar registro de datos en cola se sigue el mismo principio esperando ésta vez a que ***main()*** active el flag *col*=1 que permite la grabación de estos datos.

Por último, tanto para el caso de detener los datos a cola como para la detención de las interrupciones, lo que se espera es a que el flag *col* sea puesto a cero por ***main()*** de manera que los índices de gestión de la cola cíclica ya no varían. En estas condiciones se puede hacer una llamada a la nueva función ***mem_chec()*** cuya misión es calcular a partir de estos índices las posiciones que aún quedan libres para introducción de datos antes de empezar a sobreescribirlos. Este valor es introducido en la variable global *mem_disp* que será enviada como confirmación por el Hilo22 a tierra. Si el tiempo MARGENT es sobrepasado se envía como confirmación negativa un -1.

Código de adquisición de datos de altura.

Al principio de ***main()*** se integra parte del código del proyecto de Adrián, en el cual se alimenta el potenciómetro a 10V, sólo se selecciona un canal de escaneo y la polaridad de la conversión se selecciona como unipolar:

```
...

//=========================================================
//d-a  Alimentamos con 10 Voltios el potenciómetro.
//=========================================================
    //Configuración e inicialización de la salida analógica.
   intBuff=1;
...
```

```
...
    dasettings.polarity = (BYTE) intBuff;       /*UNIPOLAR*/
    dasettings.load_cal = (BYTE) intBuff;

    if( dscDASetSettings(dscb, &dasettings) != DE_NONE)
    {
        dscGetLastError(&errorParams);
        fprintf( stderr, "dscDASetSettings error: %s %s\n",
            dscGetErrorString(errorParams.ErrCode), errorParams.
            errstring );
        return 0;
    }

    channel = 0; // El canal de salida será el 0
    output_code = 4095;  // 9.98 voltios.

    printf( "\nAlimentando potenciómetro con 10v para medir
        altura (code, %u, to channel %d)\n", output_code, (int)
        channel );
    if( dscDAConvert( dscb, channel, output_code ) != DE_NONE )
    {
        dscGetLastError(&errorParams);
        fprintf( stderr, "dscDAConvert error: %s %s\n",
            dscGetErrorString(errorParams.ErrCode), errorParams.
            errstring );
        return 0;
    }
//============================================================
//a-d Configuración de los parámetros de conversión AD para
la entrada analógica.
//============================================================

    memset(&dscadsettings, 0, sizeof(DSCADSETTINGS));
    dscadsettings.range = RANGE_10;
    dscadsettings.polarity = UNIPOLAR;
    dscadsettings.gain = GAIN_1;
    dscadsettings.load_cal = (BYTE)TRUE;
    dscadsettings.current_channel = 0;
    dscadsettings.scan_interval = SCAN_INTERVAL_5;
    dscadsettings.addiff = (BYTE) intBuff;

    if( dscADSetSettings( dscb, &dscadsettings ) != DE_NONE )
    {
        dscGetLastError(&errorParams);
        fprintf( stderr, "dscADSetSettings error: %s %s\n",
            dscGetErrorString(errorParams.ErrCode), errorParams.
            errstring);
        env_errores(16);
        return 0;
    }
...
```

Código fuente 4.21: Cambios en el código C de main() para configuración de los parámetros de conversión A/D en la placa Hércules.

Ya que el nuevo código a integrar tiene en cuenta el vuelo del helicóptero en su plataforma, éste realiza un paso previo al inicio del registro de datos de sensores en el cual se tomará una muestra inicial del potenciómetro y se almacenará en la variable global *voltagezero*. Así pues, este código se integra al comienzo de ***main()***, tras la configuración de los parámetros de conversión A/D. El valor *voltagezero* actuará como referencia de altura cero del helicóptero y se utilizará en la función ***ad1()*** para el cálculo de las alturas durante los experimentos.

```
...
//=============================================================
//a_d scan
//=============================================================
     //Tomamos una primera muestra para determinar la tensión de
         referencia o punto de altura inicial.
     //Suponemos que en el momento inicial la plataforma está
        con la rueda apoyada en el suelo.
     //Altura = 0.16 m, Ángulo = 6.5º

   if( dscADSample( dscb, &sample ) != DE_NONE )
   {
      dscGetLastError(&errorParams);
      fprintf( stderr, "dscADSample error: %s %s\n",
         dscGetErrorString(errorParams.ErrCode), errorParams.
         errstring );
      //free( samples ); // remember to deallocate malloc()
         memory
      //return 0;
      env_errores(1);
   }

   if( dscADCodeToVoltage(dscb, dscadsettings, sample, &
      voltagezero) != DE_NONE)
   {
      dscGetLastError(&errorParams);
      fprintf( stderr, "dscADCodeToVoltage error: %s %s\n",
         dscGetErrorString(errorParams.ErrCode), errorParams.
         errstring );
      //return 0;
      env_errores(2);
   }

   printf( "\nTensión de altura cero: %5.3lfV\n\n", voltagezero)
      ;
...
```

Código fuente 4.22: Código adicional C en main() para establecer el punto de altura inicial del potenciómetro en la plataforma.

Todo el código referente al sensor de ultrasonidos se ha añadido tal cual, teniendo cuidado de definir correctamente todas las variables necesarias en distintas partes de la aplicación residente en el

computador de vuelo. También como parte de las configuraciones iniciales en la placa cuando arranca la aplicación se introduce el código de autocalibración del sensor de ultrasonidos, justo después del cálculo de *voltagezero*:

```
...
/////// AUTO–CALIBRACIÓN DEL SENSOR DE ULTRASONIDO ////////

    calibración = 20.0;  // Parámetro de calibración del sensor.
       Se corresponde con la temperatura del medio. Inicialmente
       20ºC
    altura_ini = 0;      // Variable que almacenará la altura
       inicial, que se tendrá que parecer lo más posible a
       ALTURA0


    while ( (ALTURA0 - altura_ini) >= 0.001 ) // Repetimos la
       operación de medir la altura hasta asegurarnos de que la
       altura inicial medida es muy parecida a la ALTURA0.
    {
       dscGetTime(&t_0); // Tomamos el tiempo inicial para
          asegurarnos de que la medida del sensor no dure más de
          17 ms
       altura_ini = medida_us(); // Medimos y almacenamos la
          altura en el Búffer.
       printf("altura_ini %f\n",altura_ini);
    }

    calibracion = (((200000.0*ALTURA0/contador)-331.0)/0.6); //
       Despejamos el término de la temperatura de la fórmula
       original de la velocidad del sonido. Contador almacena el
       número de us que ha estado a nivel alto la señal del
       sensor en la última medida

    printf("calibracion %f\n",calibracion);

    printf("contador %d\n",contador);

    printf("t_0 %u\n",t_0);
...
```

Código fuente 4.23: Código C adicional en main() para realización de la autocalibración del sensor de ultrasonidos.

Finalmente, la función de registro de datos de alturas ***ad1()*** queda como se muestra a continuación:

```
ad1 (void)
{
   int i;
   BYTE port;      // port used for digital I/O
   BYTE bit;       // BYTE sent to digital output
   BYTE input_b;   // BYTE receiving digital input
   BYTE input_timer;
   BYTE config_byte;        // DIO configuration byte

//=============================================================
//ADscan
//=============================================================

   dscGetTime(&t_ini);
   printf("Llamamos␣a␣medida_potenc\n");
   medida_potenc();  // II. Medida de la altura a partir del
      potenciómetro
   printf("Altura_pot:␣%f\n",altura_pot);

   printf("Llamamos␣a␣medida_us\n");
   medida_us();  // III. Medida de la altura a partir del Sensor
       de Ultrasonidos
   printf("Altura_us:␣%f\n",altura_us);

   printf("%1.5lf␣m\t%1.5lf␣m\t%d␣ms\n",altura_pot,altura_us,
      t_ini-t_0);

      //Código que escribe en la cola
      if (col1)
      {
         if (datos.ad[0].ad_disco)
            {
               printf("Introducimos␣Potenciómetro\n");
               *(p+ind_alm)=altura_pot;
               ind_cola();
            }
         if (datos.ad[1].ad_disco)
            {
               printf("Introducimos␣Ultrasonido\n");
               *(p+ind_alm)=altura_us;
               ind_cola();
            }

      }
}//fin ad1
```

Código fuente 4.24: Código C de la función ad1() de adquisición de alturas desde el potenciómetro y el sensor de ultrasonidos.

Las funciones ***medida_pot()*** y ***medida_us()*** serán las encargadas de escanear los datos de los sensores y de convertirlos a valores válidos de altura teniendo en cuenta la posición de éstos en la plataforma de vuelo y el modo de funcionamiento de cada sensor:

```
//=========================================================
// medida_potenc()
//  Función que toma la tensión de la entrada analógica
//  conectada al Potenciómetro y calcula la altura a partir
//  de ella.
//=========================================================

DFLOAT medida_potenc()
{
    DFLOAT altura = 0;
// Leemos la entrada analógica obteniendo un valor digital de
   ésta

   if( dscADSample( dscb, &sample ) != DE_NONE )
   {
         dscGetLastError(&errorParams);
      fprintf( stderr, "dscADSample error: %s %s\n",
         dscGetErrorString(errorParams.ErrCode), errorParams.
         errstring );
      env_errores(1);
   }

// Transformamos el dato obtenido de la entrada analógica en un
   dato de tensión que tomaremos como la tensión inicial.
   if( dscADCodeToVoltage(dscb, dscadsettings, sample, &voltage)
       != DE_NONE)
   {
      dscGetLastError(&errorParams);
      fprintf( stderr, "dscADCodeToVoltage error: %s %s\n",
         dscGetErrorString(errorParams.ErrCode), errorParams.
         errstring );
      env_errores(2);
   }

// Cálculo simplificado de la altura a partir de la medida del
   potenciómetro.
// Cada tensión la transformamos en un ángulo:
// angulo=34.3434*(voltage-voltagezero)
// A partir del ángulo, calculamos la ALTURA (m):
// altura = ALTURA0 + 1.08063*sin(angulo*3.1416/180)
// Hemos asumido que el ángulo es pequeño-> sin(angulo)=angulo
       altura = ALTURA0 + 0.654545454*(voltage-voltagezero);
       altura_pot=altura+ALTURACTE;
...
```

```
...
    printf("ALTURA0 %f\n", ALTURA0);
    printf("Voltage %f, Voltagezero %f\n",voltage,voltagezero);
    printf("Altura %f\n",altura);
    printf("Altura_pot %f\n",altura_pot);
return;
} // Fin de medida_potenc()
```

Código fuente 4.25: Función 'medida_potenc()' en C para el cálculo de la altura medida por el potenciómetro.

```
//=========================================================
// medida_us()
//  Función que calcula la altura a partir de la medida del
//  sensor de ultrasonidos.
//          I. Activación el sensor
//         II. Lectura de la salida del sensor
//        III. Calculo la altura
//=========================================================

DFLOAT medida_us (void)
{
   int i;
   DFLOAT altura=0;// Variable que almacenará la altura
   BYTE port;  // Puerto E/S Digital usado
   BYTE bit;   // Bit enviado a la salida digital
   BYTE input_b;   // Byte recibido por la entrada digital
   BYTE input_timer; // Entrada del temporizador auxiliar
   BYTE config_byte; // Byte de configuración de las E/S
      Digitales
   float aux1=0.0;
   float aux2=0.0;

   if (calibracion == 20.0)
//Si estamos en la autocalibración, medimos el tiempo.
      dscGetTime(&t_ini);

   //=====================================================
   // I. ACTIVACIÓN DEL SENSOR
   // Configuramos el Puerto D como salida, y lo ponemos a
   // nivel alto durante 12us para activar el Disparo del
   // sensor.
   //=====================================================
   printf("ACTIVACIÓN DEL SENSOR\n\n");
...
```

```
...
// Configuramos el puerto D (E/S Digital) como SALIDA
   config_byte = 0x08;
   // Aplicamos la configuración (Puerto D como salida y el
      resto como entrada)
   if ( dscDIOSetConfig(dscb, &config_byte ) != DE_NONE )
   {
      dscGetLastError(&errorParams);
      fprintf( stderr, "dscDIOSetConfig error: %s %s\n",
         dscGetErrorString(errorParams.ErrCode), errorParams.
         errstring );
      env_errores(40);
   }

   port = 3;         // Puerto D
   bit = 6;          // bit 6 del Puerto D
// Repetimos 3 veces la operación para asegurarnos de que se
   mantenga
// al menos durante 12 us a nivel alto.
      for(i=0;i<2;i++)
   // Ponemos el BIT 6 del puerto D a 1 durante 12 us para
      iniciar el US
        if ((result = dscDIOSetBit(dscb, port, bit)) != DE_NONE)
   {
      dscGetLastError(&errorParams);
      fprintf( stderr, "dscDIOSetBit error: %s %s\n",
         dscGetErrorString(errorParams.ErrCode), errorParams.
         errstring );
      env_errores(43);
   }
//Pasados los 12u ponemos el BIT 6 del puerto D a 0
   if ((result = dscDIOClearBit(dscb, port, bit)) != DE_NONE)
   {
      dscGetLastError(&errorParams);
      fprintf( stderr, "dscDIOClearBit error: %s %s\n",
         dscGetErrorString(errorParams.ErrCode), errorParams.
         errstring );
      env_errores(44);
   }

   //=======================================================
   // II. LECTURA DE LA SALIDA DEL SENSOR
   // Configuramos el Puerto D como entrada, y esperamos a
   // detectar un nivel alto. Una vez lo hemos detectado,
   // calculamos el tiempo que se mantiene a dicho nivel
   // apoyándonos en el reloj auxiliar emulado con una señal
   // PWM.
   //=======================================================
   printf("LECTURA DE LA SALIDA DEL SENSOR\n\n");
// Configuramos todos los puertos como entradas
   config_byte = 0x00;
...
```

```
...
// Aplicamos la configuración en los puertos
   if ( dscDIOSetConfig(dscb, &config_byte ) != DE_NONE )
   {
      dscGetLastError(&errorParams);
      fprintf( stderr, "dscDIOSetConfig error: %s %s\n",
         dscGetErrorString(errorParams.ErrCode), errorParams.
         errstring );
      env_errores(40);
   }
   do
   {
//Leemos el BIT 6 del puerto D en espera de que se ponga a nivel
    alto
      if ((result = dscDIOInputBit(dscb, port, bit, &input_b))
         != DE_NONE)
         {
            dscGetLastError(&errorParams);
            fprintf( stderr, "dscDIOInputByte error: %s %s\n",
               dscGetErrorString(errorParams.ErrCode),
               errorParams.errstring );
            env_errores(45);
         }
// Tomamos el instante en que termina la operación
         dscGetTime(&t_act);
   }
// Cuando se detecta el nivel alto o sobrepasamos el tiempo
//establecido (17 ms), continuamos
   while (input_b == 0 && (t_act-t_ini) <= 17);
//Cuando se detecta un nivel alto, se pone el contador de
//tiempo a 0. El contador cuenta cada 10 us
   contador=0;
// Vamos a utilizar para emular un contador/temporizador una
// señal PWM a 100000KHZ.
   do
   {
      port=3;
// La entrada de nuestro contador/temporizador será el bit 7
// del Puerto D
      bit=7;
      do
      {
         if ((result = dscDIOInputBit(dscb, port, bit, &
            input_timer)) != DE_NONE)
         {
            dscGetLastError(&errorParams);
            fprintf( stderr, "dscDIOInputByte error: %s %s\n",
               dscGetErrorString(errorParams.ErrCode),
               errorParams.errstring );
            env_errores(45);
         }
      }
...
```

```
...
 //Leemos el BIT 7 del puerto D (entrada del contador/
    temporizador)
// mientras esté a nivel alto
      while (input_timer == 1);
// Cada vez que leemos un pulso del PWM, incrementamos el
// contador (ha pasado 10 us)
      contador++;
      port=3;
      bit=6;

// Leemos el bit 6 del Puerto D (señal del sensor) esperando a
// que pase a nivel bajo
      if ((result = dscDIOInputBit(dscb, port, bit, &input_b))
         != DE_NONE)
      {
         dscGetLastError(&errorParams);
         fprintf( stderr, "dscDIOInputByte error: %s %s\n",
            dscGetErrorString(errorParams.ErrCode), errorParams.
            errstring );
         env_errores(45);
      }
   }

// Contamos mientras la señal del sensor esté a nivel alto
// y el contador sea menor que 17ms
   while (input_b != 0 && contador <= 1700);
   //=======================================================
   // III. CÁLCULO DE LA ALTURA
   // Una vez que la señal del sensor pasa a estar a nivel
   // bajo de nuevo y a partir del tiempo que ésta ha
   // permanecido a nivel alto, calculamos la altura a la que
   // está el objeto. Además, comprobaremos que no hayamos
   // sobrepasado el límite de tiempo en le proceso.
   //=======================================================

   printf("CALCULO DE LA ALTURA\n\n");
   aux1=(331.0+(calibracion)*0.6);
   aux2=((contador)/200000.0);

   // ALTURA(m) = (TIEMPO(s)/2)*Vsonido (m/s)
   // Vsonido(m/s) = 331.0 + T(ºC)*0.6
   altura = aux1 * aux2;
   // Resolución de 1.715 mm (contador a 100kHz -> 10us)

// III. Medida de la altura a partir del Sensor de Ultrasonidos
   altura_us= altura + ALTURACTE;

// Si en algún momento hemos sobrepasado el tiempo límite
// (17 ms), lanzamos un mensaje de error
...
```

```
...
    if(contador > 1700 || t_act-t_ini > 17)
        {
// Tomaremos la altura nula como dato erróneo
            altura = 0;
            printf("\n ALTURA ERRÓNEA. Timeout.\n");
            printf("contador:%d, t_act: %u, t_ini: %u\n",contador,
                t_act,t_ini);
            env_errores(46);
        }

    return altura;
} // Fin de la función medida_us()
```

Código fuente 4.26: Función 'medida_us()' en C para el cálculo de la altura medida por el sensor de ultrasonidos.

4.5. Módulo de gestión de datos y archivos registrados.

4.5.1. Código C en computador de vuelo

En el bucle principal `switch(n_control)` de la aplicación C se contemplan las siguientes opciones para la manipulación de los datos registrados:

```
    sem_wait(&Semaforo1);//espero al semaforo que recibe desde
        tierra
    switch (n_control)
    {
    ...
        //graba datos a disco ;sem_post(&Semaforo)
        case 7: printf("llega 7 (datos a disco)\n");grabar=1;break
            ;
        case 8: printf("llega 8 (parar datos a disco)\n");grabar
            =0;break;
        case 9: printf("llega 9 (env archivo)\n");
            if (parado)//necesito que este parado para mandar
                archivo
            {
                terg=1;//cierro el archivo desde d_disco
                DSCSleep(200);
                env_archivo();
                break;
            }
        case 10: printf("llega 10 (fin)\n");ter=1;terg=1; break;
        case 11: printf("llega 11 (borrado archivo)\n");
            borrado_archivo(); break;
...
```

Código fuente 4.27: Opciones en el bucle principal para la gestión de los datos registrados.

Paso de datos al disco duro.

Con la opción *n_control*=7, la variable global *grabar* se activa a 1 y el hilo ***d_disco*** comienza a pasar los datos de la cola cíclica a un archivo **datos.dat**, en el mismo orden en el cual fueron introducidos en ella.

```
void* Hilo(void*P) //hilo que se escribe los datos de la cola al
    archivo
{
   int i;
   //abro el archivo para escritura
   FILE *fo;
   fo = fopen("datos.dat","a+b");
   if(fo == NULL)
   {
      printf("Error al abrir el archivo \n");
      env_errores(3);
   }
   while (1)
   {
      while (grabar)
      {
         dscGetTime(&t[5]);
      //para que sec continúe con las interrupciones
         dscSleep(1);
//=============================================================
// los datos que hayan sido seleccionados estarán en la cola en
// el orden apropiado luego al sacarlos tendrán ese orden,
// sacandolos en múltiplos de los introducidos. El dato del
// numero de elementos estará en la variable n_datos_cola.
//=============================================================
         if (ind_rec==ind_alm)//la cola esta vacía
         {
            printf("cola vacía\n");
            //dscSleep(500);
            if (terg)//termina el programa
            {
               printf("cola finalizada\n");
               break;
            }
            else
            dscSleep(2000);
         }
         else //la cola no esta vacía
...
```

```
...
            {
                for (i=0;i<n_datos_cola;i++)
                {
                        fwrite((p+ind_rec),4,1,fo);
                    ind_rec++;
                //actualización y salida en caso limite
                        if (ind_rec==NU)
                        ind_rec=0;
                        s++;//datos a disco
                }
                printf("cola no vacía s %d, ind_rec %d, ind_alm %d\n",
                    s,ind_rec,ind_alm);
                dscGetTime(&t[6]);
                printf("tiempo de grabar = %u\n",t[6]-t[5]);
            }
        }
        if (terg)
            break;
        else
            dscSleep(2000);
    }//fin while, acabo la cola y termino programa
    printf("d_disco: cierro el archivo\n");
    fclose(fo);//cierro el archivo
}//fin de d_disco
```

Código fuente 4.28: Hilo 'd_disco.c' para paso de datos desde la cola al dico duro.

Vemos que con *n_control*=8 se detiene el paso de datos al archivo y que únicamente se sale del bucle principal cuando se finaliza la el programa completamente (*terg*=1). Al principio se estableció que los datos sólo podían ser pasados una vez al archivo por cada experimento, además al recibir la orden de envío de archivo el hilo era finalizado, lo cual obliga a reiniciar la aplicación para volver a activarlo.

Borrado del archivo de datos.

Entre una experimento y otro es posible que el número y tipo de sensores que se decide registrar en cola sean distintos, así pues debe existir la opción de borrar el archivo **datos.dat** cuando estas condiciones de muestreo cambien, ya que en el hilo ***d_disco*** se abre el archivo para escritura a continuación de la información preexistente, y se estarían introduciendo tramas de distinta naturaleza. Así pues, con *n_control*=11 se ejecuta un código sencillo de borrado del contenido 'datos.dat', y que no sufrirá modificación alguna.

Envío a tierra del archivo de datos registrados.

Cuando se recibe la opción *n_control*=9, se ejecuta una función denominada ***env_archivo()*** cuya misión es ir leyendo bloques de **datos.dat** e ir mandándolos por TCP/IP a través de puerto 4954. Esta función ha sido desarrollada completamente en este proyecto.

Figura 4.7: Elementos del módulo de gestión de datos y archivos.

4.5.2. Código Matlab en ordenador de tierra

Paso de datos al disco duro.

Vemos el callback perteneciente al botón para iniciar el paso de datos a disco:

```
if ((getappdata(gcf,'Estado'))==2)|((getappdata(gcf,'Estado'))==1)
        n_control=7;
else
        if ((getappdata(gcf,'Estado'))==3)
            errordlg('Captura de datos en proceso.','Error');
            return
        elseif ((getappdata(gcf,'Estado'))==0)
            errordlg('Sistema no inicializado.','Error');
            return
        end
end
t=udp(getappdata(gcf,'ipserver'),getappdata(gcf,'p2'));
set(t,'ByteOrder','littleEndian');
fopen(t);
fwrite(t,n_control,'int32');
fclose(t);
delete(t);
clear t;
 % Calcula un tiempo proporcional a la cantidad de datos registrados en cola para dar un margen antes de poder dar la orden de importar el archivo.
ocupado=single(handles.datos.t_cola)-single(handles.disponible);
espera=single(ocupado)/10000
pause(espera);
set(handles.text34,'string','Datos registrados descargados a disco duro.');
```

Código fuente 4.29: Código Matlab para paso de los datos registrados en cola al disco duro del computador de vuelo.

El paso de datos en la placa Hércules, al ser de memoria a disco es una operación enormemente rápida. En las pruebas realizadas en laboratorio se observa que deben ser cantidades de mínimo decenas de Mbs para que el tiempo empleado en este trasvase sea apreciable. Ante este hecho nos damos cuenta de que no es demasiado operativa la opción de abortar el paso de los datos a disco, sin embargo, el envío de datos al ordenador de tierra a través de la red sí resulta ser una operación bastante más lenta, debido sobre todo a las operaciones de confirmación de recepción que como veremos se

realizan por cada paquete de datos recibido. Así pues, la opción *n_control*=8 pasará a materializar la interrupción del envío de **datos.dat**.

Borrado del archivo de datos.

El código para envío de la directiva de borrado del archivo de datos es:

```
function pushbutton16_Callback(hObject, eventdata, handles)
    n_control=11;
    disp('Borrado del archivo 'datos.dat');
    t=udp(getappdata(gcf,'ipserver'),getappdata(gcf,'p2'));
    set(t,'ByteOrder','littleEndian');
    set(t,'DatagramTerminateMode','off');
    set(t,'Timeout',20);
    set(t,'Terminator',0)
    fopen(t);
    fwrite(t,n_control,'int32');
    fclose(t);
    delete(t);
    clear t;
```

Código fuente 4.30: Código Matlab para borrado del archivo de datos registrados en la placa Hércules.

Envío a tierra del archivo de datos registrados.

Acometemos el diseño de la función de recepción del archivo de datos teniendo en cuenta varios puntos:

- En las operaciones de recepción por el socket, Matlab necesita conocer la cantidad de elementos y el tipo de datos que se van a recibir.

- Las operaciones de envío en la aplicación C en la placa son procesadas más rápido que las operaciones de recepción de los paquetes por parte del Matlab, con lo cual, tras un número determinado de paquetes se llena la cola de recepción y comienza a dar errores de red que conllevan a pérdida de datos.

- En las pruebas en laboratorio se han detectado las situaciones anómalas más habituales: que suceda un error al abrir el archivo de datos en el ordenador de vuelo, que las cantidades de datos que se envían a tierra lleguen incorrectas o que directamente no llegue ningún dato. Se habrá de preparar al código para que sepa identificar estas situaciones y pueda tomar las medidas adecuadas, informando siempre al usuario del error.

Para solucionar el primer punto, diseñaremos el código en C de manera que tras una operación de lectura del archivo, envíe en primer lugar la cantidad que se a podido extraer de él. La aplicación Matlab guarda este entero en la variable local *nrecepcion*, y si es correcto lo utilizará en la operación de lectura del bloque de datos del archivo que se envía a continuación. Si *nrecepción* está vacío o no se corresponde con un valor lógico, informará del error al usuario, mandará una confirmación negativa al ordenador de vuelo y cerrará el socket.

En el caso de que *nrecepcion* sea correcto se realiza la operación de recepción de la cantidad de datos flotantes especificada en dicha variable, Estos valores flotantes se introducen en el vector *datos*. Si esta recepción falla, se envía una confirmación negativa al helicóptero. Si por el contrario se reciben los flotantes correctamente se procede a completar una matriz A que consta de 7 columnas

para albergar los datos 'Número de interrupción', 'Tiempo', 'Pitch', 'Roll', 'Yaw', 'Potenciómetro' y 'Ultrasonido'. En las columnas correspondientes a los sensores que no han sido configurados para ser guardados en cola se habrá de poner siempre un cero. Para facilitar esta identificación columna-dato se utilizan los valores de la máscara, *mascara[i]*.

La aplicación en el helicóptero se queda esperando confirmaciones tras enviar la cantidad que conforma cada paquete extraído del archivo, y tras enviar el paquete de datos en sí. De esta forma el envío se ralentiza bastante, pero nos aseguramos de que no se dan errores de red por desbordamiento de la cola de recepción.

Cuando se finaliza la recepción del archivo de forma satisfactoria, la matriz A se introduce en un archivo cuyo nombre es dado por el usuario. El nombre y la ruta del último archivo importado con éxito es guardado en los appdata *lastfile* y *pathfile*. Esta información será utilizada en el módulo de representación de los datos, y también se guarda en el archivo **conexiones.mat** al salir de la aplicación.

Se utiliza además en este código la variable global de estado *Import* para impedir operaciones sobre **datos.dat** mientras está siendo importado. Vemos como queda el callback de importar archivo completo:

```
function pushbutton15_Callback(hObject, eventdata, handles)
    if ((getappdata(gcf,'Import'))<=1)
            conexion=1;
            % Se solicita al usuario que defina el archivo para guardar los datos
            [nombreArchivo,ruta] = uiputfile('*.mat','Guardar datos de muestreo');
            if (ruta==0)
                set(handles.text34,'string','Envío de archivo cancelado por el usuario.');
            else
                disp(['User selected',fullfile(ruta,nombreArchivo)]);
                %Actualizamos el archivo .mat que por defecto se va a representar.
                set(handles.text34,'string','Selección del archivo .mat de almacenamiento de da-
tos.');
                setappdata(gcf,'lastfile',nombreArchivo);
                setappdata(gcf,'pathfile',ruta);
                n_control=9;
                %Enviamos n_control por el puerto udp 4952.
                t=udp(getappdata(gcf,'ipserver'),getappdata(gcf,'p2'));
                set(t,'ByteOrder','littleEndian');
                fopen(t);
                fwrite(t,n_control,'int32');
                fclose(t);
                delete(t);
                clear t;
                %Abrimos el puerto tcp para la recepción del archivo.
                t=tcpip(getappdata(gcf,'ipserver'),getappdata(gcf,'p5'));
                set(t,'ByteOrder','littleEndian');
                set(t,'Timeout',6);
                fopen(t); Paso de datos al disco duro.          %Recibimos del helicóp-
tero la cantidad de elementos tipo flotante que se van a recibir en el siguiente envío.
                disp('Elementos tipo flotante que se van a recibir del servidor:')
                nrecepcion=fread(t, 1, 'int32')
```

```
            if nrecepcion>0
                setappdata(gcf,'Import',2);
                 %Si se recibe cantidad errónea en el primer envío.
                if nrecepcion>(get(t,'InputBufferSize'))
                    disp('Problemas de conexion');
                    conexion=0;
                    aux=4;
                    Tbuffer=aux*handles.ndatoscola;
                else
                     %Recepción correcta de nrecepcion.
                    Tbuffer=nrecepcion;
                    aux=nrecepcion/handles.ndatoscola;
            end
            j=1;
            set(handles.text20,'string','En marcha');
            set(handles.uipanel31,'BackgroundColor',[0.545,0.804,0.196]);
         %Error al abrir el fichero en el servidor.
        elseif nrecepcion==-1
            errordlg('Problemas al abrir el fichero en servidor','Error');
             set(handles.text34,'string','Error: no ha sido posible abrir el fichero en la placa
Hércules.');
         %No se ha recibido nada del helicóptero.
        elseif (isempty(nrecepcion))
            set(handles.text34,'string','Error: no ha sido posible conectar con el helicoptero.');
            fclose(t);
            delete(t);
            clear t;
            return;
        end
        A=[];
        while (nrecepcion>0)
             %vaciamos el búffer de datos.
            datos=[];
             %Se chequea la cola de eventos para comprobar si se ha ordenado cancelar el
envío del archivo.
            drawnow;
            set(handles.text34,'string','Recibiendo el archivo de datos del helicóptero.');
            disp('Recibiendo datos...')
             %Se reciben lo 'nrecepcion' flotantes del archivo.
            datos=fread(t,Tbuffer,'float32');
            if isempty(datos)==1|nrecepcion>(get(t,'InputBufferSize'))
             %En el caso de recepción de datos errónea o recepción nula se envía confirmación
negativa a ordenador de vuelo.
                disp('ERROR DE RECEPCIÓN');
                fwrite(t,-1,'int32');
                disp('Se han perdido datos en el envío!');
                 set(handles.text34,'string','Error de recepción: se han perdido datos en el
envío.');
                set(handles.text20,'string','Abortado');
```

```
                    set(handles.uipanel31,'BackgroundColor',[1,0.392,0.392]);
                    setappdata(gcf,'Import',1);
                else
                    % Recepción correcta. Confirmación 1.
                    fwrite(t,1,'int32');
                    % Construcción de la matriz de datos A a partir de la trama de datos recibida
y la máscara.
                    p=1;
                    for i=1:aux
                        for k=1:7
                            if p<=nrecepcion
                                if handles.mascara(k)==1
                                    A(j,k)=datos(p);
                                    p=p+1;
                                else
                                    A(j,k)=0;
                                end
                            end
                        end
                        j=j+1;
                    end
                end
                nrecepcion=fread(t, 1, 'int32')
                if nrecepcion>(get(t,'InputBufferSize'))
                    nrecepcion=0;
                end
            end
            %Si el helicóptero indica que ya no se leen más datos del archivo se detiene el proceso
de recepción.
            if nrecepcion==0
                j=j-1;
                numdatos=num2str(j);
                mensaje1='Envío de archivo de datos completo. Enviados ';
                mensaje2=' registros.';
                mensaje=[mensaje1 numdatos mensaje2];
            %Se introducen todos los datos estructurados en A dentro del archivo creado por el
usuario.
                save(fullfile(ruta,nombreArchivo),'A');
                set(handles.text20,'string','Terminado');
                set(handles.uipanel31,'BackgroundColor',[1,1,0.502]);
            %Se indica si la recepción ha sido correcta o si se han perdido paquetes.
            if (conexion==1)
                    set(handles.text34,'string',mensaje);
                else
                    set(handles.text34,'string','Error de recepción: se han perdido datos en el
envío.');
                end
                setappdata(gcf,'Import',0);
            %Recepción abortada por problemas con el archivo en el helicóptero.
```

```
            elseif nrecepcion==-1
                set(handles.text20,'string','Abortado');
                set(handles.uipanel31,'BackgroundColor',[1,0.392,0.392]);
                setappdata(gcf,'Import',1);
            end
            fclose(t);
            delete(t);
            clear t;
    end
```

Código fuente 4.31: Código Matlab para recepción del archivo de datos registrados de los sensores.

El botón para interrumpir el envío del archivo tiene un callback bastante sencillo de envío de *n_control*=8 a través del puerto udp 4952. Cada elemento de la ventana genera un evento al ser modificado ó accionado, y al accionar este botón el callback de recepción del archivo de datos está en proceso, con lo cual el envío de la solicitud de interrupción espera a que la función de recepción termine de ejecutarse.

Para resolver ésto vemos que todos los objetos gráficos poseen una serie de propiedades programables por el usuario que determinarán su comportamiento o apariencia bajo ciertas circunstancias. Una de ellas es la propiedad **'Interrumpible'**: si está 'on' especifica que el callback de dicho objeto puede ser interrumpido en su ejecución. Así pues activamos esta propiedad en el botón de importar el archivo, sin embargo, Matlab procesa la cola de eventos sólo cuando el callback en ejecución finaliza o cuando encuentra dentro de ésta función ciertos comandos especiales que le obligan a suspender la ejecución y procesar el siguiente evento en la cola (siempre que su propiedad 'Interrumpible'='on').

El comando especial que utilizamos en nuestro código para que el envío de un archivo de datos pueda ser suspendido es `drawnow`, que limpia la cola de eventos y refresca la ventana gráfica.

Aparte de todo ésto, hemos de hacer que el evento generado por el botón de cancelar el envío quede bloqueado en la cola de eventos, para ello programamos su propiedad **'Busy Action'**: si **'Busy Action'**='queue' será el evento introducido en la cola, mientras que si **'Busy Action'**='cancel' será descartado.

4.5.3. Modificaciones en el código del computador de vuelo.

Paso de datos al disco duro.

Se hacen diversas modificaciones en el código del hilo ***d_disco*** de manera que cuando *grabar*=1 se abra el archivo **datos.dat** y se graben todos los datos.

```
void* Hilo(void*P) //hilo que se escribe los datos de la cola al
    archivo
{
    int i;
    int iteraccion=1;
    //abro el archivo para escritura por primera vez.
    FILE *fo;
    fo = fopen("datos.dat","a+b");
    printf("Abierto archivo datos.dat\n");
...
```

```
...
   if(fo == NULL)
   {
      printf("Error al abrir el archivo \n");
      env_errores(3);
   }
   while (1)
   {
      if (grabar)
      {
//Si es la primera iteracción no hace falta que abra el archivo
      if (iteraccion>=2)
      {
         fo = fopen("datos.dat","a+b");
         if(fo == NULL)
         {
            printf("Error al abrir el archivo \n");
            exit(1);
            env_errores(3);
         }
      }
      iteraccion++;
      }
      while (grabar)
      {
         dscGetTime(&t[5]);
      //para que sec continue con las interrupciones
         dscSleep(1);
//=============================================================
// los datos que hayan sido seleccionados estarán en la cola en
// el orden apropiado luego al sacarlos tendrán ese orden,
// sacandolos en múltiplos de los introducidos. El dato del
// numero de elementos estará en la variable n_datos_cola.
//=============================================================
         if (ind_rec==ind_alm)//la cola esta vacia
         {
            printf("cola vacia\n");
            grabar=0;
         //cierro el archivo si se vacía la cola
            fclose(fo);
//termina el programa (Matlab no lo permite, así que es poco
   probable)
            if (terg)
            {
               printf("cola finalizada\n");
               break;
            }
            else
            {
               dscSleep(2000);
            }
         }
...
```

```
...
            else //la cola no está vacía
            {
                for (i=0;i<n_datos_cola;i++)
                {
                    fwrite((p+ind_rec),4,1,fo);
                    ind_rec++;
                    if (ind_rec==NU)//actualización y salida en caso
                        limite
                    ind_rec=0;
                    s++;//datos a disco
                }
                printf("cola no vacia s %d, ind_rec %d, ind_alm %d\n",
                    s,ind_rec,ind_alm);
                dscGetTime(&t[6]);
                printf("tiempo de grabar = %u\n",t[6]-t[5]);
            }
        }
        if (terg)
        {
            fclose(fo);//cierro el archivo
            break;
        }
        else
            dscSleep(2000);
    }//fin while
    printf("terg=%d\n",terg);
}//fin de d_disco
```

Código fuente 4.32: Modificaciones en el hilo 'd_disco.c' para paso de datos desde la cola al dico duro.

Cuando la cola se vacía se vuelve a cerrar el archivo y se desactiva *grabar*=0. El hilo sigue activo esperando un nuevo cambio en la variable global, con lo cual se pueden hacer todas las grabaciones en el archivo que se deseen, y el envío de un archivo a tierra no cierra la función. Esto implica mucha más rapidez y flexibilidad en los experimentos, pero se ha de tener la precaución de borrar el archivo antes de grabar datos de un experimento en el cual las condiciones de registro de sensores han cambiado respecto a las pruebas anteriores. Una forma de evitar errores en este sentido es incluir en la función Matlab de envío de trama de inicialización un código de envío previo de *n_control*=11, de manera que nos aseguramos de que antes de modificar las condiciones de los experimentos se borra el contenido del archivo de datos.

Envío a tierra del archivo de datos registrados.

Vemos con detalle el código de la función ***env_archivo.c*** diseñada para el envío de **datos.dat**:

```
int env_archivo(void)
{
    int i=0,j=0;
...
```

```
...
    int datos_leidos;
    int numbytes;
    int conf;
// listen on sock_fd, new connection on new_fd
    int sockfd, new_fd;
// my address information
    struct sockaddr_in my_addr;
// connector's address information
    struct sockaddr_in their_addr;
    socklen_t sin_size;
    int yes=1;
    int tam=n_datos_cola*4;
    float leido[tam];

    //abro el archivo
    FILE *fo;
    printf("Vamos a abrir datos.dat\n");
    if((fo = fopen("datos.dat","rb"))==NULL)
    {
        printf("Error al abrir el archivo \n");
        env_errores(4);
    }
    printf("Después de fopen");

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror("socket");
        printf("Error en socket");
        //exit(1);
        env_errores(41);
    }
    printf("Socket definido\n");
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof
        (int)) == -1)
    {
        perror("setsockopt");
        printf("Error en setsockopt");
        //exit(1);
        env_errores(42);
    }

    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(MYPORT5);
    my_addr.sin_addr.s_addr = INADDR_ANY;
    memset(&(my_addr.sin_zero), '\0', 8);

    printf("Vamos a entrar en bind\n");

    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct
        sockaddr)) == -1)
...
```

```
...
    {
        perror("bind");
        printf("Error en bind");
        env_errores(5);
        return 0;
    }
    if (listen(sockfd, 1) == -1)
    {
        perror("listen");
        env_errores(6);
    }
    printf("Abierto Socket Servidor\n");
    sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr,
       &sin_size)) == -1)
    {
        perror("accept");
        printf("Error en new_fd");
        env_errores(47);
    }
    printf("server: got connection from %s\n",inet_ntoa(
        their_addr.sin_addr));

//Si no se pudo abrir el archivo se envía un -1.
     if (fo==NULL)
         datos_leidos=-1;
     else
     {
         datos_leidos=fread(&leido,sizeof(float),tam,fo);
         printf("Datos leidos: %d",datos_leidos);
     }
//Si se pudo abrir el archivo y se han leído datos con éxito,
//se envía la cantidad a mandar a tierra.
    if ((numbytes=send(new_fd,&datos_leidos,sizeof(int) , 0)) ==
        -1)
    {
        printf("Error al enviar la cantidad a mandar\n");
        env_errores(61);
    }
     while(datos_leidos>0)
    {
        conf=-1;
        printf("tamaño datos leidos: %d\n",sizeof(leido));
        for(i=0;i<tam;i++)
        {
            j++;
            printf("leido[%d]=%f\n",j,leido[i]);
        }
        //Se manda una cadena de datos a tierra.
        if ((numbytes=send(new_fd,leido,sizeof(leido) , 0)) == -1)
...
```

```
            {
                perror("send");
                env_errores(60);
            }
    //Se espera una confirmación desde tierra.
        numbytes=recv(new_fd,&conf,sizeof(int) , 0);
        if (numbytes== -1)
        {
            perror("send");
            env_errores(62);
    }
     //printf("datos leidos %d sent %d bytes to %s\n",
         datos_leidos, numbytes,inet_ntoa(their_addr.sin_addr));
//Se comprueba si se ha cancelado el envío del archivo desde
    tierra.
        if (archivo==0)
        {
          printf("Envio de archivo interrumpido");
          datos_leidos=-1;
          if ((numbytes=send(new_fd,&datos_leidos,sizeof(int) , 0)
              ) == -1)
                  {
                  perror("send");
              env_errores(63);
                  }
          close(new_fd);
          fclose(fo);
          return;
        }
    //Si se recibe confirmación negativa desde tierra.
        if (conf==-1)//Ha habido problemas con la conexión
        {
              printf("Paquete %f no recibido por el cliente.\n",
                  leido[0]);
              datos_leidos=0;
        }
        else datos_leidos=fread(&leido,sizeof(float),tam,fo);
        printf("Datos leidos: %d",datos_leidos);
        if ((numbytes=send(new_fd,&datos_leidos,sizeof(int) , 0))
           == -1)
        {
            printf("Error al enviar la cantidad a mandar");
            env_errores(61);
        }
    }
    printf("terminado el envio, enviados %d datos\n",j);
    close(new_fd);
    close(sockfd);
    if (fo!=NULL) fclose(fo);
    return 0;
}
```

Código fuente 4.33: Función env_archivo() en C para el envío a tierra de 'datos.dat' a través del puerto tcp/ip 4954.

Para la interrupción del envío del archivo se utiliza la variable global *archivo*, que será modificada por el Hilo22 y chequeada por ***env_archivo.c*** en cada iteración:

```
void* Hilo22(void*P)//hilo que recibe datos de tierra (n_control
    ), abre semáforo cuando llegan
{
    while (1)
    {
...
        if ((numbytes2=recvfrom(sockfd,&n_control, sizeof(
            n_control), 0,
            (struct sockaddr *)&their_addr, &addr_len)) == -1)
        {
            perror("recvfrom");
            env_errores(31);
            //exit(1);
        }
        if (n_control==8)
        {
            archivo=0;
        }
        sem_post(&Semaforo1);//abro el semáforo
...
```

Código fuente 4.34: Código C adicional en Hilo22 para la cancelación del envío del archivo de datos registrados.

4.6. Módulo de representación de datos importados.

Dentro del cuadro de representación de datos encontramos tres botones: uno que da acceso a la ventana para la representación gráfica de los datos, otro que exportará los datos a una hoja de cálculo en excel y el último botón ('monitor') que activa el panel de monitorización de los datos de sensores que se envían en tiempo real desde el helicóptero. Este último lo veremos en el siguiente módulo.

Figura 4.8: Elementos del módulo de representación de datos.

4.6.1. Representación gráfica de los resultados.

Con la activación del primer elemento se abre una nueva ventana de representación de datos, denominada **graficas_archivo.fig**. En la función constructora ***OpeningFcn*** de esta figura se procede a cargar en una matriz los valores almacenados en el último archivo importado. Para ello se valdrá de los appdata *lastfile* y *pathfile*. Acto seguido separa en distintos vectores las columnas que conforman la matriz de datos de manera que agrupamos los datos por los tipos que interesan para la representación gráfica: 'tiempo', 'pitch', 'roll', 'yaw', 'potenciometro' y 'ultrasonido'.

```
handles.lastfile = getappdata(V_principal,'lastfile');
handles.lastpath = getappdata(V_principal,'pathfile');
 %Carga los datos almacenados en el último archivo importado del helicóptero
load(fullfile(handles.lastpath,handles.lastfile),'A');
[m,n]=size(A);
handles.filas=m;
 %Se carga en vectores los distintos tipos de datos.
handles.tiempo(1:m)=A(1:m,2);
handles.pitch(1:m)=A(1:m,3)
handles.roll(1:m)=A(1:m,4);
handles.yaw(1:m)=A(1:m,5);
handles.potenc(1:m)=A(1:m,6);
handles.us(1:m)=A(1:m,7);
 %Vectores para guardar los límites originales de la representación.
handles.eje1=0;
handles.eje2=0;
 %Vectores para guardar los límites del zoom de la representación.
handles.eje1zoom=0;
handles.eje2zoom=0;
guidata(hObject,handles);
 %Se muestra al usuario el archivo de datos que está cargado para la representación.
set(handles.edit2,'string',handles.lastfile)
...
```

Código fuente 4.35: Código Matlab de carga e inicialización de variables al abrir la ventana graficas_archivo.fig.

Tras estas operaciones, se construye la ventana de representación (ver figura 4.9). En ella estarán listos para ser representados los datos correspondientes al último archivo importado desde el helicóptero, sin embargo se proporciona al usuario la opción de seleccionar otro archivo de experimentos anteriores. Así pues, en el botón 'Seleccionar Archivo' se vuelve a realizar la operación de carga de los datos desde el archivo seleccionado.

En la representación se ha de seleccionar mediante 'CheckBoxes' los sensores que se desea sean representados. El código que refresca la representación según los datos seleccionados es:

```
function pushbutton1_Callback(hObject, eventdata, handles)
    %Se hacen invisibles las barras de desplazamiento del Zoom.
    set(handles.slider3,'Visible','off');
    set(handles.slider1,'Visible','off');
    set(handles.slider4,'Visible','off');
    set(handles.slider2,'Visible','off');
```

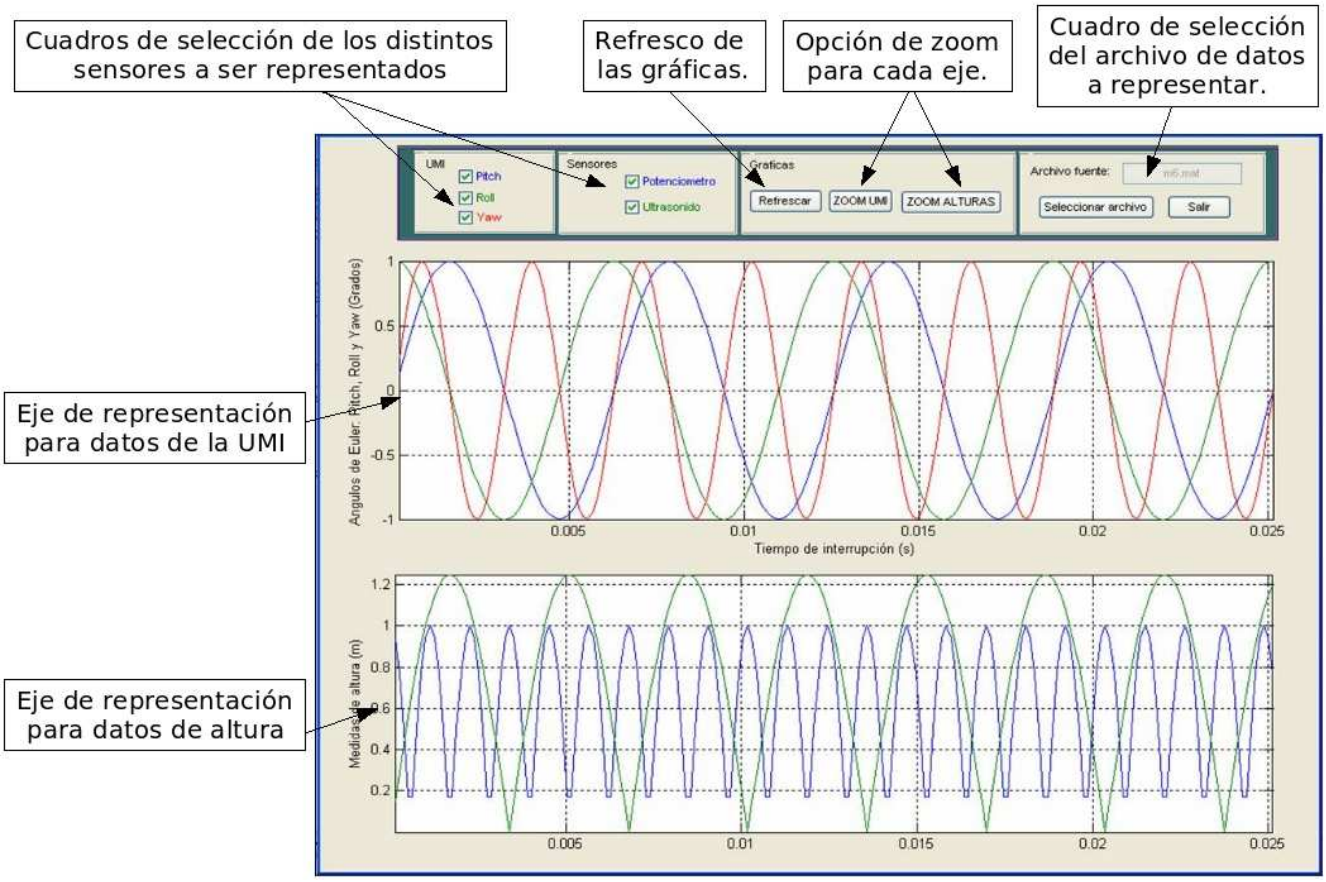


Figura 4.9: Ventana para representación gráfica de resultados de los experimentos.

```
x=handles.tiempo/1000;
m=handles.filas;
G1=zeros(m,1);
G2=zeros(m,1);
i=1;
 %Eje de la UMI.
axes(handles.axes1);
newplot;
 %Pitch.
if(get(handles.checkbox1,'Value'))
        G1(1:m,i)=handles.pitch;
        set(handles.checkbox1,'ForegroundColor',[0,0,1]);
        i=i+1;
else
        set(handles.checkbox1,'ForegroundColor',[0.251,0,0.251]);
end
 %Roll.
if(get(handles.checkbox2,'Value'))
        G1(1:m,i)=handles.roll;
        if(get(handles.checkbox1,'Value'))
            set(handles.checkbox2,'ForegroundColor',[0,0.502,0]);
```

```
        else
            set(handles.checkbox2,'ForegroundColor',[0,0,1]);
        end
        i=i+1;
else
        set(handles.checkbox2,'ForegroundColor',[0.251,0,0.251]);
end
 %Yaw.
if(get(handles.checkbox3,'Value'))
        G1(1:m,i)=handles.yaw;
        if (get(handles.checkbox1,'Value'))&&(get(handles.checkbox2,'Value'))
            set(handles.checkbox3,'ForegroundColor',[1,0,0]);
        elseif (get(handles.checkbox1,'Value'))||(get(handles.checkbox2,'Value'))
            set(handles.checkbox3,'ForegroundColor',[0,0.502,0]);
        else
            set(handles.checkbox3,'ForegroundColor',[0,0,1]);
        end
        i=i+1;
else
        set(handles.checkbox3,'ForegroundColor',[0.251,0,0.251]);
end
plot(x,G1);
grid on;
xlabel('Tiempo de interrupción (s)');
ylabel('Angulos de Euler: Pitch, Roll y Yaw (Grados)');
axis tight;
handles.eje1=axis;
i=1;
 %Eje de alturas.
axes(handles.axes2);
newplot;
 %Potenciómetro.
if(get(handles.checkbox4,'Value'))
        G2(1:m,i)=handles.potenc;
        set(handles.checkbox4,'ForegroundColor',[0,0,1]);
        i=i+1;
else
        set(handles.checkbox4,'ForegroundColor',[0.251,0,0.251]);
end
 %Ultrasonidos.
if(get(handles.checkbox5,'Value'))
        G2(1:m,i)=handles.us;
        if(get(handles.checkbox4,'Value'))
            set(handles.checkbox5,'ForegroundColor',[0,0.502,0]);
        else
            set(handles.checkbox5,'ForegroundColor',[0,0,1]);
        end
        i=i+1;
else
```

```
            set(handles.checkbox5,'ForegroundColor',[0.251,0,0.251]);
        end
        plot(x,G2);
        grid on;
        ylabel('Medidas de altura (m)');
        axis tight;
        handles.eje2=axis;
        guidata(hObject,handles);
```

Código fuente 4.36: Código Matlab de representación de los datos en sus ejes.

Para cada uno de los ejes, cada vez que se pulse el botón de ZOOM el rango de representación tanto en el eje x como en el eje y se verá dividido por dos, y se harán visibles sus respectivas barras de desplazamiento para recorrer el rango completo de la gráfica con su nueva resolución:

```
    function pushbutton2_Callback(hObject, eventdata, handles)
        axes(handles.axes1);
        v=handles.eje1;
        v1=axis
        % Se redefinen los límites del eje x con la nueva resolución.
        rangox=v1(2)-v1(1);
        maxx=(rangox/2)+v1(1);
        % Se redefinen los límites del eje y con la nueva resolución.
        rangoy=v1(4)-v1(3);
        maxy=(rangoy/2)+v1(3);
        % Se refresca la figura con los nuevos límites.
        axis([v1(1) maxx v1(3) maxy]);
        % Se activan las barras de desplazamiento.
        set(handles.slider3,'Visible','on');
        set(handles.slider1,'Visible','on');
        % Se detecta la posición de la barra y posiciona automáticamente el foco sobre el cuadrante
correspondiente.
        handles.eje1zoom=axis;
        slide=(v1(1)-v(1))/(v(2)-v(1)+rangox/2);
        set(handles.slider3,'value',slide);
        slide=(v1(3)-v(3))/(v(4)-v(3)+rangoy/2);
        set(handles.slider1,'value',slide);
        guidata(hObject,handles);
```

Así pues, el código de las barras de desplazamiento queda:

```
    function slider1_Callback(hObject, eventdata, handles)
        % Hints: get(hObject,'Value') returns position of slider
        axes(handles.axes1);
        v=handles.eje1;
        % Se recalcula el 'paso' de la barra para hacer el desplazamiento correcto.
        vzoom=handles.eje1zoom;
        ventana=vzoom(4)-vzoom(3);
        zoomax=v(3)+ventana;
        rangoy=v(4)-zoomax;
```

```
miny=(get(hObject,'Value')*rangoy)+v(3);
maxy=(get(hObject,'Value')*rangoy)+zoomax;
axis([ vzoom(1) vzoom(2) miny maxy ]);
handles.eje1zoom=axis
guidata(hObject,handles);
```

Código fuente 4.37: Código Matlab para ejecutar un zoom sobre las gráficas de datos.

Este es el código correspondiente a los ejes de la UMI. El código correspondiente a los ejes correspondientes a los sensores de altura es muy similar.

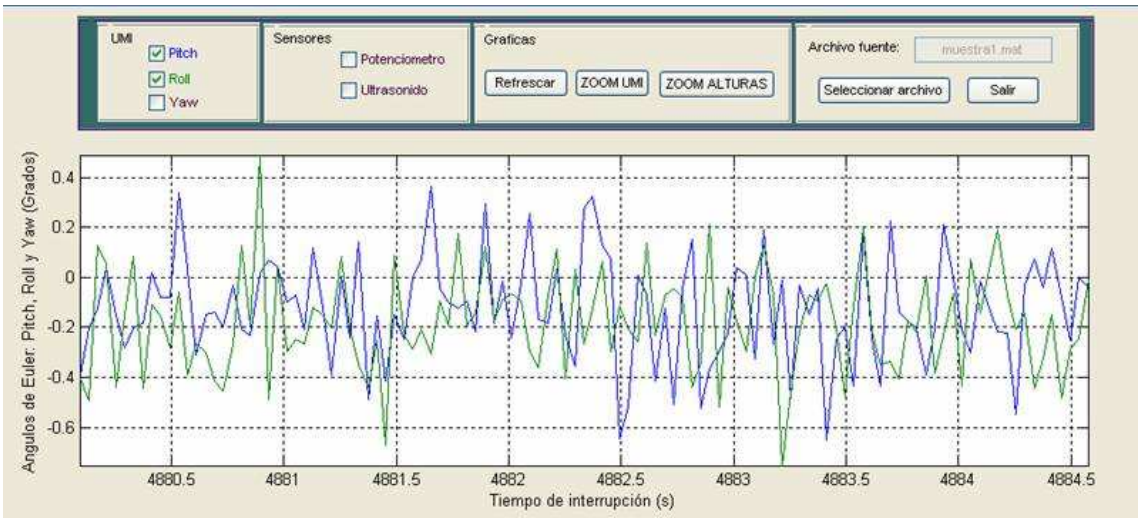


Figura 4.10: Representación de datos a escala 1:1, en el rango de valores de los datos importados.

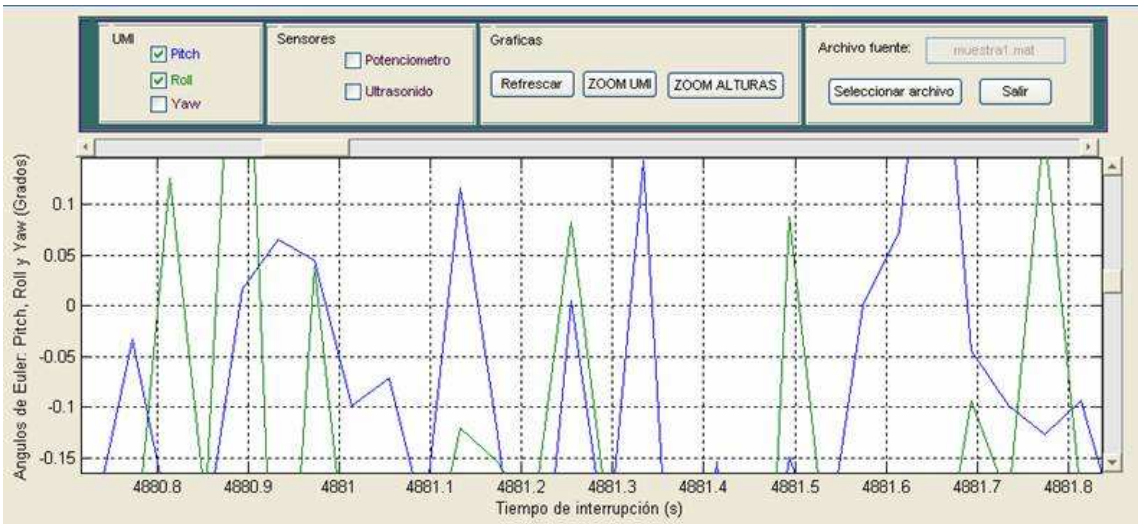
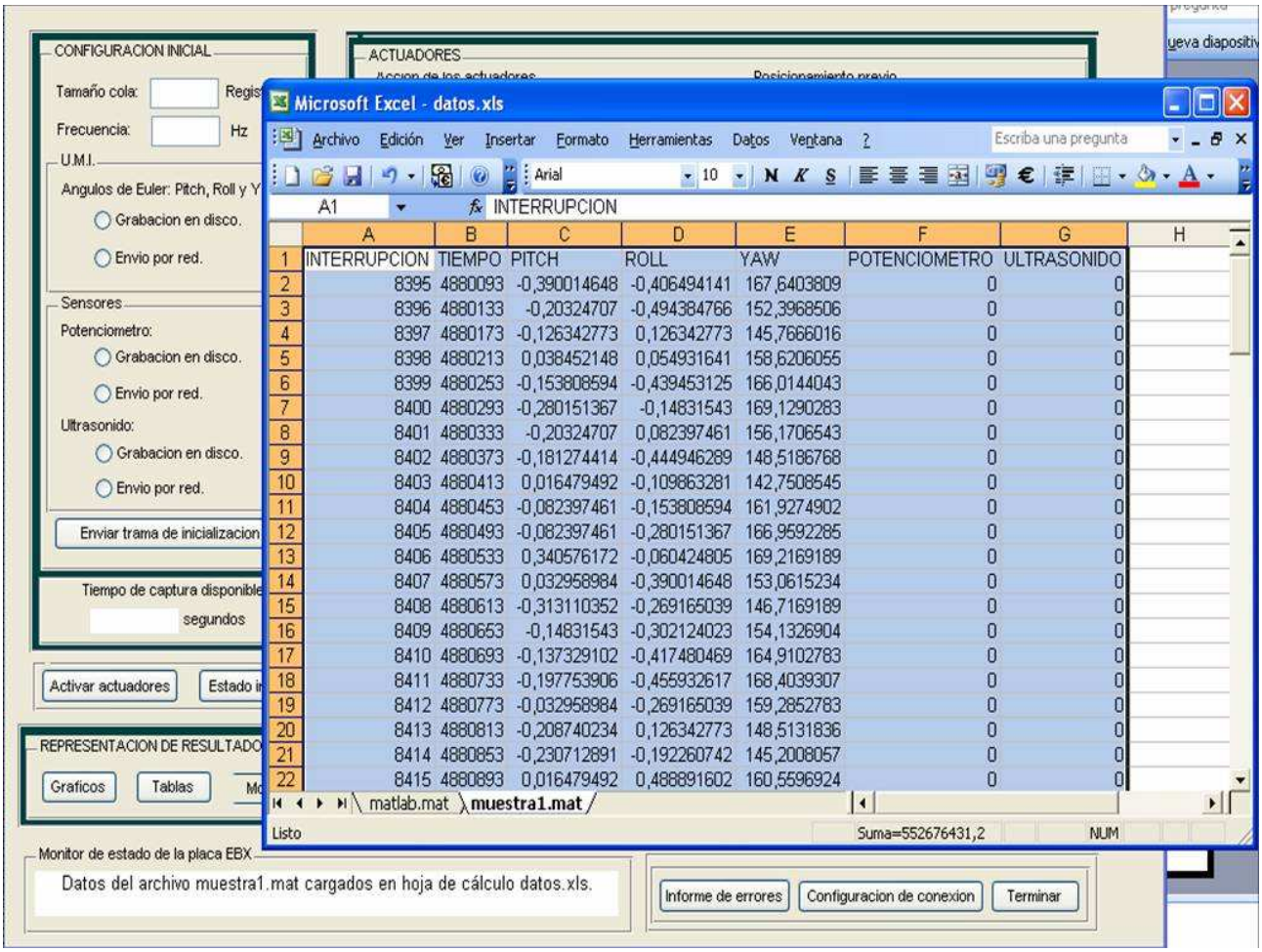


Figura 4.11: Representación de datos a escala 4:1. Aparecen las barras de desplazamiento que nos permiten desplazarnos a lo largo de todo el rango de valores de los datos importados.

4.6.2. Exportación de datos a una hoja de cálculo.

Todos los datos correspondientes a los distintos experimentos realizados se encuentran en archivos matlab (.mat), clasificados como el usuario considere oportuno. Para trabajar con esos datos podemos visualizarlos mediante la ventana de representación gráfica. Si bien deseamos trabajar con los valores en modo numérico podríamos cargar fácilmente la matriz completa de valores A mediante un `load` en el espacio de trabajo de Matlab.

Otra opción es exportar dichos valores del archivo .mat a una hoja de cálculo, para así tener los valores en tablas. Matlab permite hacer esto de forma sencilla con la orden `xlswrite`, mediante la cual, los datos contenidos en una matriz A podrán ser escritos en una hoja de cálculo 'file' dentro del archivo de excel **datos.xls**. De esta forma tendremos unificadas dentro de un único excel todas las tablas de datos que deseemos exportar. Cada una de estas tablas de datos se encuentra dentro de una hoja cuyo nombre es igual al nombre del archivo .mat exportado.



	A	B	C	D	E	F	G	H
1	INTERRUPCION	TIEMPO	PITCH	ROLL	YAW	POTENCIOMETRO	ULTRASONIDO	
2	8395	4880093	-0,390014648	-0,406494141	167,6403809	0	0	
3	8396	4880133	-0,20324707	-0,494384766	152,3968506	0	0	
4	8397	4880173	-0,126342773	0,126342773	145,7666016	0	0	
5	8398	4880213	0,038452148	0,054931641	158,6206055	0	0	
6	8399	4880253	-0,153808594	-0,439453125	166,0144043	0	0	
7	8400	4880293	-0,280151367	-0,14831543	169,1290283	0	0	
8	8401	4880333	-0,20324707	0,082397461	156,1706543	0	0	
9	8402	4880373	-0,181274414	-0,444946289	148,5186768	0	0	
10	8403	4880413	0,016479492	-0,109863281	142,7508545	0	0	
11	8404	4880453	-0,082397461	-0,153808594	161,9274902	0	0	
12	8405	4880493	-0,082397461	-0,280151367	166,9592285	0	0	
13	8406	4880533	0,340576172	-0,060424805	169,2169189	0	0	
14	8407	4880573	0,032958984	-0,390014648	153,0615234	0	0	
15	8408	4880613	-0,313110352	-0,269165039	146,7169189	0	0	
16	8409	4880653	-0,14831543	-0,302124023	154,1326904	0	0	
17	8410	4880693	-0,137329102	-0,417480469	164,9102783	0	0	
18	8411	4880733	-0,197753906	-0,455932617	168,4039307	0	0	
19	8412	4880773	-0,032958984	-0,269165039	159,2852783	0	0	
20	8413	4880813	-0,208740234	0,126342773	148,5131836	0	0	
21	8414	4880853	-0,230712891	-0,192260742	145,2008057	0	0	
22	8415	4880893	0,016479492	0,488891602	160,5596924	0	0	

function pushbutton20_Callback(hObject, eventdata, handles)
    file = getappdata(gcf,'lastfile');
    path = getappdata(gcf,'pathfile');

```
% Se cargan en A los valores del archivo .mat.
load(fullfile(path,file),'A');
d = 'INTERRUPCION', 'TIEMPO','PITCH','ROLL','YAW','POTENCIOMETRO','ULTRASONIDO';
xlswrite('datos', d,file);
mensaje1='Datos del archivo ';
mensaje2=' cargados en hoja de cálculo datos.xls.';
mensaje=[mensaje1 file mensaje2];
set(handles.text34,'string',mensaje);
% Se escriben los valores en la hoja 'file' dentro de datos.xls.
xlswrite('datos', A,file,'A2');
% Se abre el archivo para visualizarlo.
N = xlsread('datos',-1);
```

Código fuente 4.38: Código Matlab para exportación de datos a una hoja de cálculo.

4.7. Módulo de recepción de datos en tiempo real.

4.7.1. Código C en el computador de vuelo.

En cada interrupción en la placa Hércules la función ***sec()*** llama en último lugar a la función que se encarga de enviar los datos configurados para ello: ***env_d_red()***.

```
env_d_red (void)
{
    int i=0,tam=0,j=0;
    int sockfd;
// información sobre la dirección del servidor
    struct sockaddr_in their_addr;
    struct hostent *he;
    int numbytes;
    char *arg=DIR;
    float d_red[MAXRED];
// obtener información de máquina servidor (tierra)
    if ((he=gethostbyname(arg)) == NULL)
    {
        perror("gethostbyname");
        env_errores(7);
    }
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    {
        perror("socket");
        env_errores(8);
    }
...
```

```
...
    their_addr.sin_family = AF_INET; // Ordenación de bytes de
       máquina
    their_addr.sin_port = htons(MYPORT); // short, Ordenación de
       bytes de la red
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    memset(&(their_addr.sin_zero), '\0', 8); // poner a cero el
       resto de la estructura
    //guardamos datos en estructura de envío
    if (datos.umi[0].umi_red)
    {
        d_red[j]=pitch;
        printf("d_red[%d]=%f\n",j,d_red[j]);
        j++;
    }
    if (datos.umi[1].umi_red)
    {
        d_red[j]=roll;
        printf("d_red[%d]=%f\n",j,d_red[j]);
        j++;
    }
    if (datos.umi[2].umi_red)
    {
        d_red[j]=yaw;
        printf("d_red[%d]=%f\n",j,d_red[j]);
        j++;
    }
    if (datos.ad[0].ad_red)
    {
        d_red[j]=altura_pot;
        printf("d_red[%d]=%f\n",j,d_red[j]);
        j++;
    }
    if (datos.ad[1].ad_red)
    {
        d_red[j]=altura_us;
        printf("d_red[%d]=%f\n",j,d_red[j]);
        j++;
    }

//Se podría enviar la matriz de quaternions en el futuro
    //if (datos.umi[3].umi_red)
    //for(i=0;i<4;i++) d_red[j]=quat[i];j++;

    tam=j*4;
...
```

```
...
    //Enviamos los datos a tierra.
    if ((numbytes=sendto(sockfd,&d_red, tam, 0,
        (struct sockaddr *)&their_addr, sizeof(struct sockaddr)))
            == -1) {
        perror("sendto");
        env_errores(9);
    }

    close(sockfd);
}//fin env_d_red1c
```

Código fuente 4.39: Código C de la función env_d_red() encargada de enviar los datos registrados de los sensores en tiempo real.

Según la configuración inicial, los datos se van introduciendo dentro de la cadena de flotantes *d_red* y acto seguido se envía. Así pues, el tamaño de la cadena es variable y se especificará en la orden de envío mediante la variable local *tam*.

4.7.2. Código Matlab en ordenador de tierra.

El cuadro donde se monitoriza el estado del helicóptero en tiempo real (su actitud y altura) está desactivado por defecto. Cuando pulsamos el botón 'Monitor', el sistema utiliza la máscara *red* para habilitar los cuadros correspondientes a los datos de los sensores que se espera recibir en tiempo real.

```
function togglebutton2_Callback(hObject, eventdata, handles)
 % Hint: get(hObject,'Value') returns toggle state of togglebutton2
    if (get(hObject,'Value'))==1
     % Al presionar el botón se habilitan los cuadros de texto de los datos que se van a recibir por
red.
            if handles.red(3)==1
                set(handles.text35,'Enable','on');
                set(handles.text36,'Enable','on');
                set(handles.text37,'Enable','on');
                set(handles.edit19,'Enable','on');
                set(handles.edit20,'Enable','on');
                set(handles.edit21,'Enable','on');
            end
            if handles.red(6)==1
                set(handles.text38,'Enable','on');
                set(handles.edit22,'Enable','on');
            end
            if handles.red(7)==1
                set(handles.text39,'Enable','on');
                set(handles.edit23,'Enable','on');
            end
            frec=handles.datos.frec;
            contador=single(1)/single(frec)
             % Se abre el socket por el puerto udp 4950.
```

```
            red=udp(getappdata(gcf,'ipserver'));
            set(red,'LocalPort',4950);
            set(red,'ByteOrder','littleEndian');
            set(red,'DatagramTerminateMode','off');
            set(red,'Timeout',contador);
            fopen(red);
             % Se evalúa la variable global 'stopred'.
            while (getappdata(gcf,'stopred'))==0
                [A,count]=fread(red, handles.ndatosred,'float32')
                if (count>0)
                    i=1;
                    j=1;
                    while i<6
                        if handles.red(i+2)==1
                            B(i)=A(j);
                            j=j+1;
                        else
                            B(i)=0;
                        end
                        i=i+1;
                    end
                     % Utilizamos el vector auxiliar B(i) para representar los datos que se van
recibiendo por el socket.
                    if handles.red(3)==1
                        set(handles.edit19,'String',num2str(B(1),' %7.4f'));
                        set(handles.edit20,'String',num2str(B(2),' %7.4f'));
                        set(handles.edit21,'String',num2str(B(3),' %7.4f'));
                    end
                    if handles.red(6)==1
                        set(handles.edit22,'String',num2str(B(4),' %7.4f'));
                    end
                    if handles.red(7)==1
                        set(handles.edit23,'String',num2str(B(5),' %7.4f'));
                    end
                end
                 % Chequea la cola de eventos por si se ha activado 'stopred'=1.
                drawnow;
            end
            fclose(red);
            delete(red);
            clear red;
        elseif (get(hObject,'Value'))==0
            setappdata(gcf,'stopred',1);
            if(exist('red'))
                fclose(red);
                delete(red);
                clear red;
            end
            set(handles.text38,'Enable','off');
```

```
            set(handles.text39,'Enable','off');
            set(handles.text35,'Enable','off');
            set(handles.text36,'Enable','off');
            set(handles.text37,'Enable','off');
            set(handles.edit19,'Enable','off');
            set(handles.edit20,'Enable','off');
            set(handles.edit21,'Enable','off');
            set(handles.edit22,'Enable','off');
            set(handles.edit23,'Enable','off');
    end
```

Código fuente 4.40: Código Matlab para monitorización de datos de sensores en tiempo real.

En la apertura del socket, esta vez el ordenador de tierra hace las veces de servidor y el helicóptero se comporta como cliente, con lo cual se ha de especificar el puerto local. En la operación de recepción se especifica la cantidad a recibir mediante la variable global *ndatosred*, cuyo valor es fijado al enviar la trama de inicialización del experimento. Este vector recibido A, de tamaño variable, es transformado a un vector auxiliar B, de tamaño fijo 7, utilizando el patrón que nos ofrece la máscara *red*. Así podremos representar con más facilidad los valores recibidos en sus respectivos cuadros de texto.

La operación se detiene si el appdata *stopred*=1. Esta variable será activada, bien si se vuelve a pulsar el botón 'Monitor', bien si se detienen las interrupciones.

Se ha de tener en cuenta que si la frecuencia de interrupciones es muy alta el helicóptero enviará gran cantidad de datos por segundo y el Matlab refrescará los valores en la pantalla a gran velocidad, y podría resultar entonces algo difícil distinguir los valores con nitidez para el usuario.

En el código, para que el Matlab se adapte a la frecuencia de interrupciones en el computador de vuelo, se programa la propiedad 'Timeout' del objeto udp creado. Esta propiedad fija el tiempo de espera antes de que se dé por no recepcionado el paquete de datos. Aunque es inevitable que se pierdan paquetes en un canal como éste, disminuirá en gran medida la pérdida de información si 'Timeout' es proporcional a la frecuencia de interrupción. Esto tiene el inconveniente ya comentado de que a partir de ciertas frecuencias la monitorización va a demasiada velocidad.

4.8. Otras funcionalidades.

4.8.1. Vuelta al estado inicial del sistema.

Una utilidad que puede ahorrar mucho tiempo es la posibilidad de volver a un estado 'inicial' en la aplicación, en el cual se puedan enviar nuevas tramas de inicialización para comenzar experimentos distintos en la misma sesión. Ya que se eliminó del código C en la placa el código que inhabilitaba el Hilo21 en el bucle `switch(n_control)`, es bastante más fácil alcanzar un estado óptimo para volver a definir un experimento:

```
function pushbutton17_Callback(hObject, eventdata, handles)
    if ((getappdata(gcf,'Estado'))<=1)
            %Volvemos a habilitar la configuración de parámetros de trama inicial.
            set(handles.edit2,'Enable','on');
            set(handles.edit3,'Enable','on');
            set(handles.radiobutton1,'Enable','on');
```

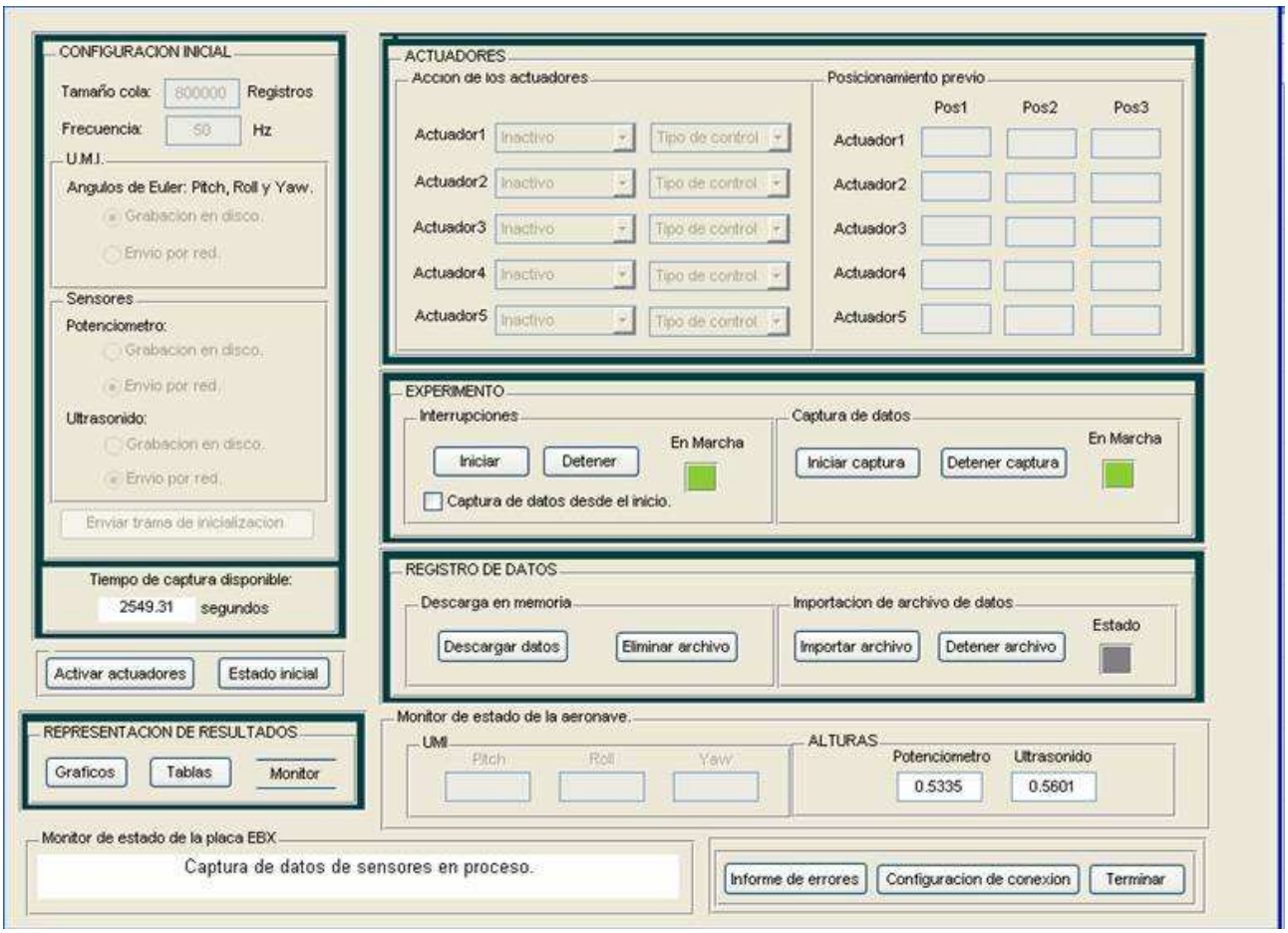


Figura 4.13: Monitorización de los datos del potenciómeto y sensor de ultrasonidos mientras los datos de la UMI son registrados en la cola.

```
set(handles.radiobutton2,'Enable','on');
set(handles.radiobutton3,'Enable','on');
set(handles.radiobutton4,'Enable','on');
set(handles.radiobutton5,'Enable','on');
set(handles.radiobutton6,'Enable','on');
set(handles.pushbutton1,'Enable','on');
%Apagamos los pilotos.
set(handles.text15,'string','Estado');
set(handles.uipanel16,'BackgroundColor',[0.502,0.502,0.502]);
set(handles.text16,'string','Estado');
set(handles.uipanel17,'BackgroundColor',[0.502,0.502,0.502]);
set(handles.text20,'string','Estado');
set(handles.uipanel31,'BackgroundColor',[0.502,0.502,0.502]);
set(handles.text34,'string','Modo inicio: helicoptero a la espera de trama de inicializa-
cion.');
%Desactivamos las ventanas de configuración de actuadores, y ponemos sus valores a
cero.
set(handles.popupmenu1,'Enable','off');
set(handles.popupmenu1,'Value',1);
```

```
set(handles.popupmenu2,'Enable','off');
set(handles.popupmenu2,'Value',1);
set(handles.popupmenu3,'Enable','off');
set(handles.popupmenu3,'Value',1);
set(handles.popupmenu4,'Enable','off');
set(handles.popupmenu4,'Value',1);
set(handles.popupmenu5,'Enable','off');
set(handles.popupmenu5,'Value',1);
set(handles.popupmenu6,'Enable','off');
set(handles.popupmenu6,'Value',1);
set(handles.popupmenu7,'Enable','off');
set(handles.popupmenu7,'Value',1);
set(handles.popupmenu8,'Enable','off');
set(handles.popupmenu8,'Value',1);
set(handles.popupmenu9,'Enable','off');
set(handles.popupmenu9,'Value',1);
set(handles.popupmenu10,'Enable','off');
set(handles.popupmenu10,'Value',1);
set(handles.edit24,'Enable','off');
set(handles.edit24,'String','');
set(handles.edit25,'Enable','off');
set(handles.edit25,'String','');
set(handles.edit26,'Enable','off');
set(handles.edit26,'String','');
set(handles.edit27,'Enable','off');
set(handles.edit27,'String','');
set(handles.edit28,'Enable','off');
set(handles.edit28,'String','');
set(handles.edit29,'Enable','off');
set(handles.edit29,'String','');
set(handles.edit30,'Enable','off');
set(handles.edit30,'String','');
set(handles.edit31,'Enable','off');
set(handles.edit31,'String','');
set(handles.edit32,'Enable','off');
set(handles.edit32,'String','');
set(handles.edit33,'Enable','off');
set(handles.edit33,'String','');
set(handles.edit34,'Enable','off');
set(handles.edit34,'String','');
set(handles.edit35,'Enable','off');
set(handles.edit35,'String','');
set(handles.edit36,'Enable','off');
set(handles.edit36,'String','');
set(handles.edit37,'Enable','off');
set(handles.edit37,'String','');
set(handles.edit38,'Enable','off');
set(handles.edit38,'String','');
i=1;
```

```
            while i<6
                handles.a(i).actuador=single(0);
                handles.a(i).control=single(0);
                handles.a(i).p1=single(0);
                handles.a(i).p2=single(0);
                handles.a(i).p3=single(0);
                i=i+1;
            end
             % Se informa al helicóptero de que se vuelve a un estado inicial y que debe poner
'piloto0' a 0.
            n_control=12;
            t=udp(getappdata(gcf,'ipserver'),getappdata(gcf,'p2'));
            set(t,'ByteOrder','littleEndian');
            fopen(t);
            fwrite(t,n_control,'int32');
            fclose(t);
            delete(t);
            clear t;
             % Se inicializan las variables locales más importantes.
            handles.mascara=[1,1,0,0,0,0,0];
            handles.ndatoscola=2;
            handles.red=[1,1,0,0,0,0,0];
            handles.ndatosred=0;
             % Se solicita a tierra que se borre el archivo 'datos.dat'.
            t=udp(getappdata(gcf,'ipserver'),getappdata(gcf,'p2'));
            set(t,'ByteOrder','littleEndian');
            fopen(t);
            n_control=11;
            fwrite(t,n_control,'int32');
            fclose(t);
            delete(t);
            clear t;
        else
            errordlg('Interrupciones en curso en la Hércules', 'Error');
        end
        guidata(hObject,handles);
```

Código fuente 4.41: Código Matlab para recuperación de condiciones iniciales en la aplicación.

Lo que se hace en este punto es volver a habilitar el módulo de envío de tramas de configuración, volver a sus valores iniciales las variables más importantes y deshabilitar de nuevo el módulo de los actuadores. Además se apagarán todos los pilotos del panel. Con todo esto se pueden volver a enviar tramas de inicio al helicóptero para nuevos experimentos.

4.8.2. Monitorización de eventos en el helicóptero.

En las pruebas en vuelo del helicóptero, la placa Hércules no tendrá conectado monitor alguno, con lo cual todos los comandos de impresión de mensajes de estado o error en el código C serán poco

operativos. Habilitamos así una ventana en la aplicación de tierra que de mensajes concisos sobre el estado en el que se encuentra la aplicación del computador de vuelo.

Esta pequeña pantalla informará al usuario de los hechos más relevantes: si la placa se ha inicializado con éxito o no, si se han iniciado las interrupciones correctamente, si se ha perdido la conexión, etc. Como hemos ido viendo a lo largo de todo este capítulo, los mensajes que serán representados en este monitor serán elaborados por cada callback de interacción con el helicóptero y su naturaleza dependerá directamente de la comunicación bidireccional desarrollada sobre todo a partir del sistema de envío de confirmaciones desde los hilos.

4.8.3. Envío de un informe de errores.

El sistema de monitor comentado nos muestra los hechos y situaciones más importantes para conocer el desarrollo del experimento, suficiente para que es usuario pueda trabajar con comodidad. Sin embargo, una determinada situación de error general reflejada en el monitor de tierra puede tener muchos matices que al usuario le pueden interesar para depuración del código, es decir, si por ejemplo en el envío del archivo de 'datos.dat' se pierde la conexión, la aplicación dará efectivamente un mensaje de pérdida de conexión, pero puede resultar interesante conocer en qué punto del código se produjo este error: al abrir el archivo o en la apertura del socket.

Así pues, nos fijamos en la función ***env_errores()*** definida en el computador de vuelo. Esta función, originalmente lo que hacía era simplemente imprimir un mensaje de error a partir de un código que se le pasaba como parámetro. Aprovechamos estos mensajes de error ya definidos y modificamos la función ligeramente para que cree un informe **errores.txt** sobre los motivos de una situación anómala y que proporciona datos mucho más pormenorizados sobre los errores acontecidos que el monitor de tierra.

```
env_errores (int error)
{
//Si es la primera vez que se llama a env_errores, se mete la
   fecha y la hora de comienzo del programa.
   if (archivoError==0)
   {
      time ( &horalocal );
      informe = fopen("errores.txt","w+");
      fprintf(informe,"INFORME␣DE␣ERRORES␣DE␣APLICACION␣SERVIDOR
         ␣EN␣PLACA␣HERCULES\n");
      fprintf(informe,"Creación␣del␣informe:␣%s", ctime (&
         horalocal));
   }
   else
   {
      informe=fopen("errores.txt","a+");
   }
   archivoError++;
   printf("Registrando␣error␣%d␣en␣informe\n",error);
...
```

```
...
    switch (error)
    {
        case 0: fprintf(informe,"Error␣0:␣dscPWMFunction␣error␣en␣
            accion.c␣\n");fprintf(informe,"dscPWMFunction␣error:␣%s
            ␣%s\n",dscGetErrorString(errorParams.ErrCode),
            errorParams.errstring);break;
        case 1: fprintf(informe,"Error␣1:␣dscADScan␣error␣en␣AD.c␣
            \n");fprintf(informe,"dscADSample␣error:␣%s␣%s\n",
            dscGetErrorString(errorParams.ErrCode),errorParams.
            errstring);break;
        case 2: fprintf(informe,"Error␣2:␣dscADCodeToVoltage␣error
            ␣en␣AD.c␣\n");fprintf(informe,"dscADCodeToVoltage␣error
            :␣%s␣%s\n",dscGetErrorString(errorParams.ErrCode),
            errorParams.errstring);break;
        case 3: fprintf(informe,"Error␣3:␣error␣al␣abrir␣archivo␣
            datos.dat␣para␣escritura␣en␣d_disco.c␣\n");break;
        case 4: fprintf(informe,"Error␣4:␣error␣al␣abrir␣archivo␣
            datos.dat␣para␣lectura␣en␣env_archivo.c␣\n");break;
        case 5: fprintf(informe,"Error␣5:␣error␣en␣bind␣en␣
            env_archivo.c␣\n");break;
        case 6: fprintf(informe,"Error␣6:␣error␣en␣listen␣en␣
            env_archivo.c␣\n");break;
        case 7: fprintf(informe,"Error␣7:␣error␣en␣gethostbyname␣
            en␣env_d_red.c␣\n");break;
        case 8: fprintf(informe,"Error␣8:␣socket␣en␣env_d_red.c␣\n
            ");break;
        case 9: fprintf(informe,"Error␣9:␣sendto␣en␣env_d_red.c␣\n
            ");break;
        case 10: fprintf(informe,"Error␣10:␣Port␣open␣failed␣en␣
            main.c␣\n");break;
        case 11: fprintf(informe,"Error␣11:␣Could␣not␣map␣the␣
            Device␣to␣a␣Port␣en␣main.c␣\n");fprintf(informe,"%s\n",
             explainError(deviceNum));break;
        case 12: fprintf(informe,"Error␣12:␣GetSerialNumber␣error␣
            en␣main.c␣\n");fprintf(informe,"S/N␣Error␣-␣:␣%s\n",
            explainError(errorCode));break;
        case 13: fprintf(informe,"Error␣13:␣Firmware␣error␣en␣main
            .c␣\n");printf("Error␣-␣:␣%s\n", explainError(errorCode
            ));break;
        case 14: fprintf(informe,"Error␣14:␣dscInit␣error␣en␣main.
            c␣\n");fprintf(informe,"dscInit␣error:␣%s␣%s\n",
            dscGetErrorString(errorParams.ErrCode), errorParams.
            errstring );break;
        case 15: fprintf(informe,"Error␣15:␣dscInitBoard␣error␣en␣
            main.c␣\n");fprintf(informe,"dscInitBoard␣error:␣%s␣%s\
            n",dscGetErrorString(errorParams.ErrCode),errorParams.
            errstring);break;
...
```

```
...
        case 16: fprintf(informe,"Error␣16:dscADSetSettings␣error␣
            en␣main.c␣\n");fprintf(informe,"dscADSetSettings␣error:
            ␣%s␣%s\n",dscGetErrorString(errorParams.ErrCode),
            errorParams.errstring);break;
        case 17: fprintf(informe,"error␣17:␣Unable␣to␣allocate␣
            memory␣AD␣en␣main.c␣\n");break;
        case 18: fprintf(informe,"Error␣18:␣dscCancelOp␣error␣en␣
            main.c␣\n");fprintf(informe,"dscCancelOp␣error:␣%s␣%s\n
            ",dscGetErrorString(errorParams.ErrCode),errorParams.
            errstring);break;
        case 19: fprintf(informe,"Error␣19:␣
            dscClearUserInterruptFunction␣error␣en␣main.c␣\n");
            fprintf(informe,"dscClearUserInterruptFunction␣error:␣%
            s␣%s\n",dscGetErrorString(errorParams.ErrCode),
            errorParams.errstring);break;
        case 20: fprintf(informe,"Error␣20:␣error␣al␣abrir␣datos.
            dat␣para␣borrado␣en␣borrado_archivo.c.\n");break;
        case 21: fprintf(informe,"error␣21:␣dscInitBoard␣error␣en␣
            (ini)␣en␣main.c\n");break;
        case 22: fprintf(informe,"Error␣22:␣memoria␣insuficiente␣
            para␣colas␣en␣(ini)␣main.c\n");break;
        case 23: fprintf(informe,"Error␣23:␣dscUserInt␣error␣en␣(
            arranca)␣main.c␣\n");fprintf(informe,"dscUserInt␣error:
            ␣%s␣%s\n",dscGetErrorString(errorParams.ErrCode),
            errorParams.errstring);break;
        case 24: fprintf(informe,"Error␣24:␣dscCancelOp␣error␣en␣(
            parar)␣en␣main.c␣\n");fprintf(informe,"dscCancelOp␣
            error:␣%s␣%s\n",dscGetErrorString(errorParams.ErrCode),
            errorParams.errstring);break;
        case 25: fprintf(informe,"Error␣25:␣
            dscClearUserInterruptFunction␣error␣en␣(parar)␣en␣main.
            c\n");fprintf(informe,"dscClearUserInterruptFunction␣
            error:␣%s␣%s\n",dscGetErrorString(errorParams.ErrCode),
            errorParams.errstring);break;
        case 26: fprintf(informe,"Error␣26:␣error␣socket␣en␣Hilo21
            ␣en␣rec_d_red.c\n");break;
        case 27: fprintf(informe,"Error␣27:␣error␣bind␣en␣Hilo21␣
            en␣rec_d_red.c\n");break;
        case 28: fprintf(informe,"Error␣28:␣error␣en␣recvfrom␣en␣
            Hilo21␣en␣rec_d_red.c\n");break;
        case 29: fprintf(informe,"Error␣29:␣error␣socket␣en␣Hilo22
            \n");break;
        case 30: fprintf(informe,"Error␣30:␣error␣bind␣en␣Hilo22\n
            ");break;
        case 31: fprintf(informe,"Error␣31:␣error␣recvfrom␣de␣
            n_control␣en␣Hilo22␣\n");break;
        case 32: fprintf(informe,"Error␣32:␣error␣socket␣en␣Hilo23
            \n");break;
        case 33: fprintf(informe,"Error␣33:␣error␣bind␣en␣Hilo23\n
            ");break;
...
```

```
...
        case 34: fprintf(informe,"Error␣34:␣error␣recvfrom␣de␣
            trama␣en␣Hilo23\n");break;
        case 35: fprintf(informe,"Error␣35:␣error␣leyendo␣angulos␣
            Euler␣en␣umi.c␣\n");fprintf(informe,"Error␣-␣:␣%s\n",
            explainError(errorCode));break;
        case 36: fprintf(informe,"error␣36:␣error␣leyendo␣m.␣
            quartenions␣en␣umi.c␣\n");break;
        case 37: fprintf(informe,"error␣37:␣error␣leyendo␣datos␣en
            ␣umi.c␣␣\n");break;
        case 38: fprintf(informe,"Error␣38:␣error␣en␣send␣del␣
            piloto0␣hilo21␣en␣rec_d_red.c\n");break;
        case 39: fprintf(informe,"Error␣39:␣error␣en␣send␣de␣
            indicacion␣de␣tiempo␣sobrepasado␣en␣hilo21␣en␣rec_d_red
            .c\n");break;
        case 40: fprintf(informe,"Error␣40:␣error␣en␣configuracion
            ␣puerto␣D␣en␣ad.c␣\n");fprintf(informe,"dscDIOSetConfig
            ␣error:␣%s␣%s\n",dscGetErrorString(errorParams.ErrCode)
            ,errorParams.errstring);break;
        case 41: fprintf(informe,"Error␣41:␣error␣de␣socket␣en␣
            env_archivo.c␣\n");break;
        case 42: fprintf(informe,"Error␣42:␣error␣de␣setsockopt␣en
            ␣env_archivo.c␣\n");break;
        case 43: fprintf(informe,"Error␣43:␣error␣en␣configuracion
            ␣bit␣6␣puerto␣D␣en␣ad.c␣\n");fprintf(informe,"
            dscDIOSetBit␣error:␣%s␣%s\n",dscGetErrorString(
            errorParams.ErrCode),errorParams.errstring);break;
        case 44: fprintf(informe,"Error␣44:␣error␣en␣set␣0␣bit␣6␣
            puerto␣D␣en␣ad.c␣\n");fprintf(informe,"dscDIOClearBit␣
            error:␣%s␣%s\n",dscGetErrorString(errorParams.ErrCode),
            errorParams.errstring);break;
        case 45: fprintf(informe,"Error␣45:␣error␣en␣lectura␣bit␣6
            ␣puerto␣D␣en␣ad.c␣\n");fprintf(informe,"dscDIOInputByte
            ␣error:␣%s␣%s\n",dscGetErrorString(errorParams.ErrCode)
            ,errorParams.errstring);break;
        case 46: fprintf(informe,"Error␣46:␣ALTURA␣ERRÓNEA.␣
            Timeout.\n");fprintf(informe,"contador:%d,␣t_act:␣%u,␣
            t_ini:␣%u\n",contador,t_act,t_ini);break;
        case 47: fprintf(informe,"Error␣47:␣error␣en␣accept␣new_fd
            ␣en␣env_archivo.c␣\n");break;
        case 48: fprintf(informe,"Error␣48:␣error␣en␣send␣del␣
            piloto1␣hilo22␣en␣rec_d_red.c\n");break;
        case 49: fprintf(informe,"Error␣49:␣error␣en␣send␣de␣
            indicacion␣de␣tiempo␣sobrepasado␣en␣hilo22␣en␣rec_d_red
            .c\n");break;
        case 50: fprintf(informe,"Error␣50:␣error␣en␣send␣de␣col1␣
            hilo22␣en␣rec_d_red.c\n");break;
        case 51: fprintf(informe,"Error␣51:␣error␣en␣send␣de␣
            mem_disp␣hilo22␣en␣rec_d_red.c\n");break;
        case 60: fprintf(informe,"Error␣60:␣error␣en␣send␣de␣los␣
            datos␣leidos␣en␣env_archivo.c␣\n");break;
...
```

```
...
        case 61: fprintf(informe,"Error␣61:␣error␣en␣send␣de␣la␣
            cantidad␣de␣datos␣a␣enviar␣a␣tierra␣en␣env_archivo.c␣\n
            ");break;
        case 62: fprintf(informe,"Error␣62:␣error␣en␣recv␣de␣la␣
            confirmacion␣de␣tierra␣en␣env_archivo.c␣\n");break;
        case 63: fprintf(informe,"Error␣63:␣error␣en␣send␣de␣la␣
            confirmacion␣envío␣interrumpido␣en␣env_archivo.c␣\n");
            break;
        case 100: fprintf(informe,"Error␣100:␣saltada␣interrucion␣
            %d␣\n",counter);fprintf(informe,"t[0]␣%u,␣t_int␣%u\n",t
            [0],t_int);break;
        default:fprintf(informe,"no␣llego␣opcion␣correcta\n");
            break;
    }
    fclose(informe);
}
```

Código fuente 4.42: Código C de la función env_errores() que crea un informe detallado de errores ocurridos durante un experimento.

Se diseña una función ***env_informe()*** para enviar el informe de errores a tierra a petición del usuario:

```
int env_informe(void)
{
    int i=0,j=0;
    int datos_leidos;
    int numbytes;
    int conf;
    int sockfd, new_fd;  // listen on sock_fd, new connection on
        new_fd
    struct sockaddr_in my_addr;    // my address information
    struct sockaddr_in their_addr; // connector's address
        information
    socklen_t sin_size;
    int yes=1;
    int tam=200;
    char leido[tam];
    char fin[3]="fin";
    //Se abre el archivo
    FILE *fo;
    printf("Vamos␣a␣abrir␣errores.txt\n");
    if((fo = fopen("errores.txt","r"))==NULL)
    {
        printf("Error␣al␣abrir␣el␣archivo␣\n");
    }
...
```

```
...
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror("socket");
        printf("Error en socket");
    }
    printf("Socket definido\n");
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof
        (int)) == -1)
    {
        perror("setsockopt");
        printf("Error en setsockopt");
    }

    my_addr.sin_family = AF_INET;      // host byte order
    my_addr.sin_port = htons(MYPORT5);    // short, network byte
        order
    my_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill
        with my IP
    memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the
         struct
    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct
        sockaddr)) == -1)
    {
        perror("bind");
        printf("Error en bind");
        return 0;
    }

    if (listen(sockfd, 1) == -1)
    {
        perror("listen");
    }
    printf("Abierto Socket Servidor\n");
    sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr,
        &sin_size)) == -1)
    {
        perror("accept");
        printf("Error en new_fd");
    }
    printf("server: got connection from %s\n",inet_ntoa(
        their_addr.sin_addr));

    //Si no se pudo abrir el archivo se envía un -1.
     if (fo!=NULL)
    {
        //Se leen los datos del archivo
        while(fgets(leido,tam,fo)!=NULL)
...
```

```
...
        {
            conf=-1;
            printf("Datos leidos: %s\n",leido);
            //Se manda una cadena de datos a tierra.
            printf("Tamaño leido: %d\n",strlen(leido));
            if ((numbytes=send(new_fd,leido,(strlen(leido)-1) , 0))
                == -1)
            {
                perror("send");
                printf("Error en send");
            }
            //Se espera una confirmación desde tierra.
            numbytes=recv(new_fd,&conf,sizeof(int) , 0);
            printf("Confirmacion:%d\n",conf);
            if (numbytes== -1)
            {
                perror("send");
                printf("Error en recv");
            }
            //Si se recibe confirmación negativa desde tierra.
            if (conf==-1)
            {
                if (fgets(leido,tam,fo)==NULL)
                {
                    printf("Fin de envío de archivo.\n");
                }
                else//Ha habido problemas con la conexión
                {
                    printf("Se han perdido datos en el envío.\n");
                    close(new_fd);
                    close(sockfd); carácter nulo es escrito
                    fclose(fo);
                    return;
                }
            }
        }
    }
    printf("Terminado el envio del informe de errores.\n");
    close(new_fd);
    close(sockfd);

    if (fo!=NULL) fclose(fo);
    return 0;
}
```

Código fuente 4.43: Código C de la función env_errores() que envía a tierra por el puerto tcp/ip 4954 el informe detallado de errores ocurridos durante un experimento.

El código Matlab para la recepción del informe de errores es el siguiente:

```
function pushbutton25_Callback(hObject, eventdata, handles)
if ((getappdata(gcf,'Estado'))<=1)
        fid=fopen('Errores.dat','w');
        if fid==-1
            set(handles.text34,'string','Error: no ha sido posible abrir Errores.txt en tierra.');
            return;
        end
        n_control=13;
        t=udp(getappdata(gcf,'ipserver'),getappdata(gcf,'p2'));
        set(t,'ByteOrder','littleEndian');
        fopen(t);
        fwrite(t,n_control,'int32');
        set(handles.text34,'string','Recibiendo el informe de errores del helicóptero.');
        fclose(t);
        delete(t);
        clear t;
        t=tcpip(getappdata(gcf,'ipserver'),getappdata(gcf,'p5'));
        set(t,'ByteOrder','littleEndian');
        set(t,'Timeout',2);
        fopen(t);
         % Se reciben las líneas del informe de errores.
        nrecepcion=200;
        [datos,count]=fscanf(t,' %c',nrecepcion);
        set(handles.text34,'string','Recibiendo el informe de errores del helicóptero.');
        i=-2;
        while (count>0)
            %Confirmación 1
            disp('Confirmacion positiva.');
            fwrite(t,1,'int32');
             %Se van introduciendo las líneas recibidas en el archivo destino seguidas de un
terminador de nueva línea.
            fprintf(fid,' %s \n',datos);
            i=i+1
            disp('Recibiendo datos...')
            [datos,count]=fscanf(t,' %c',nrecepcion)
        end
         %Confirmación -1
        disp('Confirmacion negativa.');
        fwrite(t,-1,'int32');
        if (i>0)
            mensaje1='Envío de informe de errores completo. Registrados ';
            mensaje2=' errores.';
            mensaje=[mensaje1 num2str(i) mensaje2];
            set(handles.text34,'string',mensaje);
        else
             set(handles.text34,'string','No se ha generado ningún informe de errores en el
helicóptero.');
```

```
            end
            status = fclose(fid)
            fclose(t);
            delete(t);
            clear t;
        end
```

Código fuente 4.44: Código Matlab para recepción de un informe de errores creado por la aplicación de vuelo.

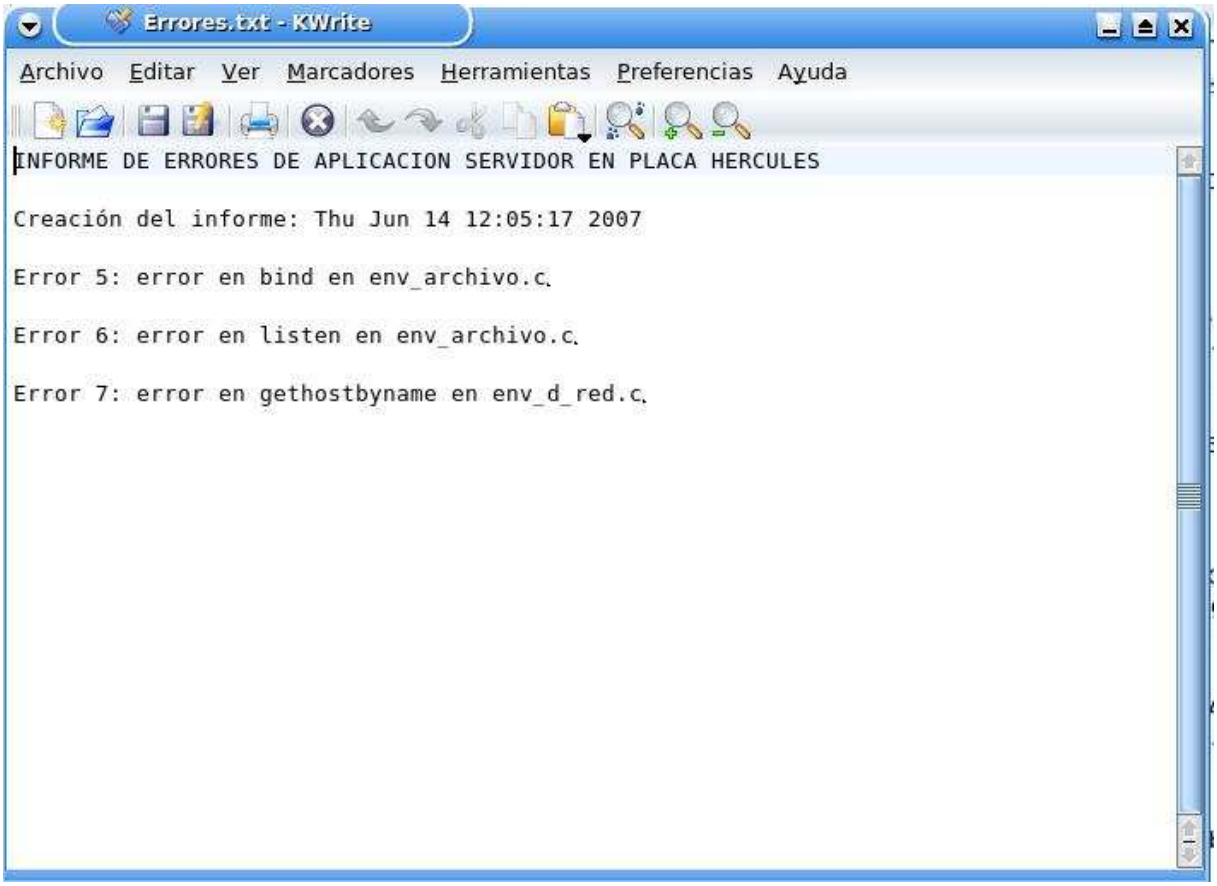


Figura 4.14: Ejemplo de un informe de errores importado desde el helicóptero.