4 Aplicación de tiempo real

4.1 Requisitos

Requisitos de memoria RAM

La memoria RAM de que disponemos es de 128Mb. De esta cantidad podemos utilizar hasta 50Mb para el almacenamiento de datos de vuelo, que posteriormente volcaremos a la flash disk.

Siendo los datos a almacenar floats (4 bytes) podremos almacenar:

50Mb/4=12.5M datos

si muestreamos a 50Hz (20ms) podremos hacer experimentos de

12.5M/50=250.000s para un solo canal.

Normalmente utilizaremos alrededor de 25 canales por lo tanto tendremos = 10.000s; 166.6minutos

Requisitos de disco duro

Disponemos de 128Mb en la flask disk utilizando:

90Mb el Sistema Operativo

5Mb los distintos programas y drivers situados en /

32Mb para los datos registrados en los experimentos

Requisitos de velocidad

El micro que tiene la Hercules funciona a 400MHz siendo esta velocidad suficiente para cumplir nuestros objetivos. Donde gastamos mas tiempo es en la lectura de los datos de la imu a través del puerto RS-232, lo que hace que el resto del tiempo de computo sea mínimo con respecto a este.

4.1.1 Control

El control deberá realizarse a una frecuencia fija. Esta frecuencia será función del numero de operaciones que tengamos que hacer a cada interrupción. Por lo tanto el control será la primera actividad que realizaremos en cada interrupción, para que la frecuencia no varie.

Con los datos obtenidos de los distintos sensores se ejecutara el programa de control, el cual calculará los valores para los cinco actuadores en función del tipo de control seleccionado y la entrada a seguir.
El tipo de control y sus parámetros será elegidos desde el control de tierra.
Los valores de los actuadores se los pasara a la función PWM la cual le dará la salida a los servomotores en función de un valor de pwm de 0 a 100.

4.1.2 Captura de datos

La captura de datos se realiza desde la IMU (a través del puerto RS-232), desde la entrada analógica y desde la entrada digital de la placa Hercules.
Los datos que van a ser capturados son seleccionados previamente desde el ordenador de tierra, que mandará un mensaje con esta información a través de la red al helicóptero. De un posible dato a capturar podremos seleccionar:

si es capturado.
si es guardado en la cola (para posterior grabado a disco).
si es mandado por red al ordenador de tierra.

Los datos que podemos capturar son:
En la IMU:
Pitch, roll y yaw
Matriz de quartenions
Entradas analógicas:
Altura
Nivel de la batería
Los datos serán almacenados en la cola circular de tipo FIFO (primer elemento en entrar primero en salir), esperando que desde el ordenador de tierra se de la orden de empezar a grabar esos datos a disco.

4.1.3 Comunicación con tierra

La comunicación del helicóptero con tierra solucionará las siguientes necesidades;
Envío de ordenes y datos de configuración desde tierra a helicóptero.

Envío continuo de datos de vuelo en tiempo real desde el helicóptero a tierra.
Este envío se hará a través de distintos canales, uno para cada tipo de dato diferente. Según la dirección de los datos;

Helicóptero a tierra;

Puerto 4950; para el envío de datos muestreados a tierra. Este envío se realizara en cada interrupción.

Puerto 4954; para el envío del archivo escrito por el helicóptero al finalizar el experimento.

Puerto 4955; para los mensajes de error.

Tierra a helicóptero;

Puerto 4951; para la recepción de los parámetros de inicio, con los que se configurara el experimento.

Puerto 4952; para la recepción de ordenes desde tierra.

Puerto 4953; para las tramas que se utilizarán para la excitación en alguno de los ejes en los experimentos de identificación.

4.2 Arquitectura

4.2.1 Software

4.2.1.1 Hercules

4.2.1.1.1 Estructura de datos

Debido a la estructura de programación concurrente introducida por los hilos y las interrupciones existen una serie de variables globales visibles por todos los programas. Estas pueden ser separadas en variables de control y variables de red
Las variables de control servirán para inicializar y terminar programas.
Las variables de red es la información enviada por tierra a través de la red, estas servirán para saber que datos son muestreados, la frecuencia de muestreo, el tipo de control etc.

4.2.1.1.1.1 Datos globales de control

El siguiente cuadro nos muestra las variables globales de control utilizadas en nuestro programa.

Var. g. control	Usada en:	Para:
Int terg	D_disco.c	Terminar una vez que la cola este vacía
Int col	Umi.c	Permitir que los datos se almacenen en la cola
Int col1	Umi.c y ad.c	Soluciona el problema de robustez con el cambio de la variable col.
Int ind_alm, ind_rec		Índices que gestionan la cola.
Int NU	Ind_alm() D_disco.c	Numero máximo de elementos que caben en la cola
Float *p;	Ind_alm()	Puntero de la cola

	D_disco.c main	
Int grabar	Main.c D_disco.c	Permite que podamos iniciar o parar de grabar los datos de la cola
Int n_datos_cola	Main.c D_disco.c Ind_cola()	Numero de datos que mandamos a la cola en cada interrupción

4.2.1.1.1.2 Datos modificables desde tierra: variables de red

Desde el control de tierra, al iniciar el experimento mandaremos la configuración de los siguientes parámetros;

<table>
<tr><td>Datos umi</td><td colspan="3">0=Pitch.1=roll. 2=yaw.
3=Quat[3]</td><td rowspan="2">Lectura
Escritura en disco
Envío por red</td></tr>
<tr><td>Datos AD</td><td colspan="3">0-5</td></tr>
<tr><td colspan="5">Tamaño de cola</td></tr>
<tr><td colspan="5">Frecuencia de interrupciones (hz.)</td></tr>
<tr><td rowspan="5">Actuadores</td><td rowspan="5">0-4</td><td colspan="3">0=Nada</td><td></td></tr>
<tr><td rowspan="3">1=Control</td><td rowspan="3">Elegir
Tipo</td><td>Parámetro 1</td><td></td></tr>
<tr><td>Parámetro 2</td><td></td></tr>
<tr><td>Parámetro 3</td><td></td></tr>
<tr><td colspan="3">2Excitación</td><td></td></tr>
</table>

4.2.1.1.1.3 Datos modificables antes de compilar

Hay una serie de datos que podremos modificar antes de compilar nuestros programas.

Estos datos han sido organizados en un archivo que se encuentra en la carpeta en la que están todos los programas y se llama define.h

En él podremos modificar los siguientes parámetros:

	Parámetro	Valor actual
Numero de datos umi	N	3
Numero de datos AD	M	1
Numero de datos enviados al mismo tiempo en env_archivo.c	NDATOS	10
Numero máximo de variables a enviar por red en env_red.c	MAX	10
Dirección a la que mandamos los datos. Usada por env_d_red.c	DIR	"127.0.0.1"
Dirección de nuestra máquina. Usada por rec_d_red.c	MYDIR	"127.0.0.1"

4.2.1.1.2 Protocolo de comunicaciones

La comunicación entre el helicóptero y tierra se hará de formas distintas en función de que datos se estén enviando y la dirección.

Los envíos se harán a través de funciones y la recepción a través de hilos, lo que hará que las funciones de recepción de datos puedan quedarse “colgadas” independientemente sin hacerlo todo el programa, debido a la estructura multitarea introducida por los hilos.

Tipo	Nombre	Datos	Dirección	Tipo	Puerto
función	env_d_red.c	Datos interrupción	H->T	UDP	4950
función	env_archivo.c	Archivo	H->T	TCP/IP	4954
función	env_errores.c	Nº error producido	H->T	UDP	4955
Hilo	Hilo21	Configuración	T->H	UDP	4951
Hilo	Hilo22	N_control	T->H	UDP	4952
Hilo	Hilo23	Trama	T->H	UDP	4953

En el caso del envío de datos en cada interrupción utilizamos el sistema UDP, para el envío del archivo completo al finalizar el experimento utilizamos el sistema TCP/IP asegurándonos así que el envío se realiza correctamente.

Para la recepción de datos utilizamos el sistema UDP (sockets de datagramas), en el que la comunicación se establece entre dos sockets sin que el escritor sepa si el lector esta realmente escuchando o no y si realmente le ha llegado el mensaje o no. Se utiliza este sistema debido a que nos asegura que no perdemos demasiado tiempo en estas funciones, y en caso de que un paquete importante (por ejemplo una orden) no haya llegado podemos enviarlo otra vez.

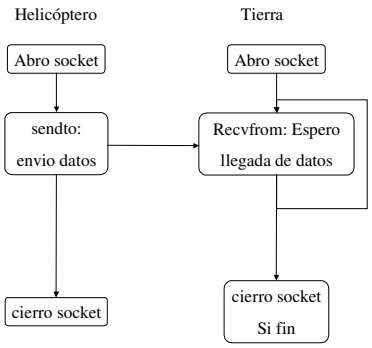
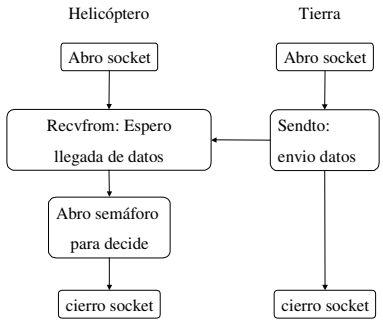
Helicóptero a tierra: hilo en tierra	Tierra a helicóptero: hilo en helicóptero
	

Ilustración 11: Envío y recepción de datos.

El helicóptero enviará a tierra los datos seleccionados cada interrupción y al finalizar el experimento le mandará el archivo escrito, además de los errores en caso de que se produzcan.

El envío de datos del helicóptero a tierra se hace a través de la función env_datos_red la cual es llamada en cada interrupción, mandando los datos que han sido seleccionados. Para conocer el numero de datos que debemos enviar he creado una variable llamada n_datos_red, que en ini() suma el total de datos a enviar a la red para que la función de envío lo conozca. Este numero también será conocido por la función que recibe los datos en tierra, ya que han sido seleccionados desde aquí.

Desde tierra se le enviarán los datos previamente seleccionados que configurarán el experimento, así como las ordenes que manejarán el experimento.

Desde tierra se seleccionaran los siguientes parámetros de inicio;

Elegimos los datos a muestrear, datos a disco y datos al ordenador de tierra, de los canales de la imu, la adquisición de datos analógicos y digitales.

El tamaño de la cola en la que se escribirán los datos.

La frecuencia de muestreo.

Sobre los cinco ejes seleccionaremos;

Si funcionan como controladores, como excitación o si no trabajan.

El control que llevan y sus parámetros de control o la secuencia de excitación.

También se le mandarán las tramas que servirán de excitación a los ejes.

Las ordenes que se le enviarán al helicóptero son las siguientes;

n_control	Switch (n_control) hace:
1	Recibe datos de configuración. Ejecuta ini
2	Inicio interrupciones.
3	Inicio interrupciones. Col = 1
4	Col = 1
5	Parar interrupciones. Col = 0
6	Col = 0
7	Grabar = 1
8	Grabar = 0
9	Envío archivo
10	Fin programa
11	Borrado de archivo

4.2.1.1.3 Hilos

En el software del helicóptero tendremos tres hilos dedicados a la comunicación con tierra y uno a la escritura de datos a disco.

Los hilos se ejecutarán de manera concurrente con lo que tendremos un sistema multitarea, y debido a que comparten el mismo espacio de direcciones los datos globales serán accesibles desde todos los hilos.

Tenemos los siguientes hilos;

3 hilos dedicados a recibir datos desde tierra, los cuales estarán bloqueados en la función recvfrom en espera de datos. Cuando estos lleguen activarán un semáforo

para que se ejecute el bucle switch (n_control), que actuara en función de que datos hayan llegado.
Estos hilos serán cancelados por main cuando llegue la orden de fin.
Hilo de escritura en disco (*d_disco*), este hilo escribirá en disco los datos de la cola (en bloques de *n_datos*, sumados por *ini*, para no perder la secuencia), una vez que haya llegado la orden de escritura en disco (*grabar=1*). Este hilo se ejecutara cada interrupción un numero de veces variable en función del tiempo que quede hasta la siguiente interrupción, con lo que aprovecharemos el tiempo hasta la siguiente interrupción.
Cuando el programa termine main esperara a que haya escrito todos los datos que había en la cola haciendo *pthread_join*.

4.2.1.1.4 Funciones

Tendremos las siguientes funciones:

4.2.1.1.4.1 Main:

Se encuentra en el archivo *main.c* junto con *ini, parar y arrancar*.
Encargada de inicializar los parámetros básico de placa, los de la umi, crear los hilos y gestionar switch (n_control) en función de los parámetros que llegan desde la red.
En switch (n_control) entrara en un bucle que ejecutará cada vez que se abra el semáforo que abren las funciones de recepción de datos desde red.
En función de lo que le llegue desde la red el *switch (n_control)* hará;
Si llegan los datos de configuración llamara a *ini*.
Si llega orden de arranca llama a *arranca*. Cancelará el hilo desde el que llega los datos de configuración para que no de error si llegan nuevos datos. *parado=0*.
Orden de iniciar el envió de datos a cola, *cola=1*.
Orden de parar el envió de datos a cola, *cola=0*.
Orden de parar las interrupciones. *parar()*, *parado=1*.
Orden de empezar la escritura en disco, g*rabar=1*.

Orden de terminar la escritura en disco, g*rabar=0*.

Orden de mandar archivo: enviara el archivo (si *parado =1*).

Si llega fin main saldrá del bucle *while(!ter)* en que se encontraba poniendo *ter=1*, cancelara las interrupciones, esperara a que termine el hilo de escritura en disco y finalizara el programa.

4.2.1.1.4.2 Arranca:

Se encuentra en el archivo *main.c.*

Función que inicia las interrupciones llamando cada tiempo de muestreo a la función *sec()*.

4.2.1.1.4.3 Sec:

Se encuentra en el archivo *sec.c*.

Función que llama a las funciones implicadas en la interrupción (*accion, umi, AD, envia_d_red*) además de contabilizar el tiempo que tardan cada una de ellas. La primera a la que se llama es a *accion* para que el control se produzca siempre en el mismo momento.

4.2.1.1.4.4 Umi:

Se encuentra en el archivo *umi.c*.

Función que lee los datos de la imu seleccionados y escribe en cola los que hayan sido seleccionados. Los datos se mandarán a cola una vez que la variable de control *col1=1*, esta variable se actualiza desde la variable *col,*

evitando que una orden de inicio de datos a cola (*col=1*) llegue entre la ejecución de el programa *imu()* y *ad()*, lo que originaria que la cola se llenase con un numero de datos distintos a *n_d_cola* y el consiguiente mal funcionamiento de esta (los datos no serian validos ya que estarían desordenados).

Para el control del llenado de los datos de cola hay un pequeño programa en *umi.c* llamado *ind_cola()* que se encarga de gestionar los índices de llenado y recuperación de la cola circular de tipo FIFO.

Para que este programa pueda funcionar se incluye los siguientes archivos en la misma carpeta que umi a la hora de ser compilados;

M3dmgUtils.c
M3dmgSerialLinux.c
M3dmgAdapter.c
M3dmgAdapter.h
M3dmgutils.h
M3dmgSerial.h
M3dmgErrors.h

4.2.1.1.4.5 Ind_cola():

Modifica los valores de ind_alm y ind_rec, los índices que me indican la posición de almacenamiento y recuperación de la cola cíclica FIFO en la que almacenamos los datos. Es llamada cada vez que se introduce un dato en la cola.

Al ir almacenando datos se aumenta el valor de *ind_alm*, y al ser cíclica si llegamos al final de cola, es decir *ind_alm = ind_rec,* el índice de recuperación avanzara para estar por delante de índice de almacenamiento. Este avance de ind_rec se hará en el numero de elementos que estamos almacenando en cada interrupción (*n_d_disco*), para no perder el orden en la cola.

4.2.1.1.4.6 AD:

Se encuentra en el archivo *ad.c*.

Función que lee los datos de la entrada AD para los datos seleccionados, realiza la conversión analógica - digital y escribe en cola los que hayan sido seleccionados (una vez que col1 = 1).

4.2.1.1.4.7 Acción:

Esta función se encuentra en el archivo *accion.c*.

Función que gestiona los actuadores (5 servomotores), en función de si actúan como control, como excitación o si no actúan.

Llamara a la función *control* que calculará el valor de los controladores y devolverá el valor que se tiene que aplicar a la salida (función PWM).

Si actúan como excitación leerá los datos de la trama que se vaya a usar y se los mandara a la salida (función PWM).

Los datos de la trama han sido enviados por red, el primer dato que se haya enviado será indicativo del actuador al que será asociado. El resto de los números serán los valores que tomará el pwm (de cero a cien).

4.2.1.1.4.8 Control:

En control.c.

Es llamada por accion para calcular el control de cada uno de los ejes seleccionados como control.

El valor de la salida es pasado por accion a control como referencia.

El valor de control será función de la entrada, del tipo de control y sus parámetros, de los datos de los sensores y del resto de los controles.

4.2.1.1.4.9 Env_d_red:

Situada en el archivo *env_d_red.c*.

Llamada por sec() en cada interrupción esta función se encarga de mandar por red los datos seleccionados, mediante el protocolo UDP. Tiene una variable *tam* para conocer el numero de bytes que envía de una vez a tierra.

Esta variable se calcula en el mismo programa contando los datos a enviar y multiplicándolos por 4 (ya que los datos a enviar son floats).

4.2.1.1.4.10 Env_archivo:

Situada en env_archivo.c.

Función que manda el archivo escrito por *d_disco* al ser pedido desde tierra una vez parado el experimento (necesita que *parado=1*).

4.2.1.1.4.11 Env_errores:

Situada en env_errores.c.

Función que manda el numero de error producido. El hilo que espera la llegada de datos interpreta el numero de error y nos muestra por la pantalla del ordenador de tierra el texto del error producido. dependiendo de la gravedad del error se enviará un numero o si este es grave conllevará la salida de programa.

4.2.1.1.4.12 Rec_d_red:

Formado por tres hilos, situados todos en *rec_d_red.c*.

Hilo21 se encarga de recibir los datos de configuración.

Una vez que llegan abre semáforo y pone n_control=1, indicando que han llegado datos de configuración.

Hilo22 recibe las ordenes de control (*n_control*). Abre semáforo cuando llega.

*Hilo2*3 se encarga de recibir las tramas.

Al recibir una trama de 9 elementos lee el primero y transforma la trama en una matriz. Así de

Int trama[9];

pasamos a

int tram[4][8];

en la que el primer índice nos indica a que actuador va asignado.

Cuando recibe trama pone *n_control=0* (indicando que ha llegado trama) y abre semáforo.

Cada uno de los hilos esta bloqueado en la función *recfrom*, en espera de datos, cuando le llega algún dato abre semáforo para que se ejecute el *switch (n_control)*, y actue en función de lo que le ha llegado.

4.2.1.1.4.13 D_disco

Hilo que gestiona la escritura en disco de los datos de la cola.

Realiza la apertura del archivo y espera a que llegue por red la orden de grabar (*grabar=1*). Hasta que no llegue este dato el programa ira chequeándolo y si no llega esperara 2s.

Al ser FIFO (primero en entrar, primero en salir) cuando recuperemos los datos iremos sacando los datos desde ind_rec, y siempre en un numero igual a n_d_disco, para no perder el orden de la cola.

Al ir sacando datos de la cola se preguntara si ha llegado al final de la cola y si es así si ha llegado la señal de fin de programa *terg=1*.

4.2.1.1.5 Diagrama flujo bucles principales. Cronograma

En los siguientes gráficos se muestran los diagramas de flujo de los bucles principales.

4.2.1.1.5.1 Diagrama de main:

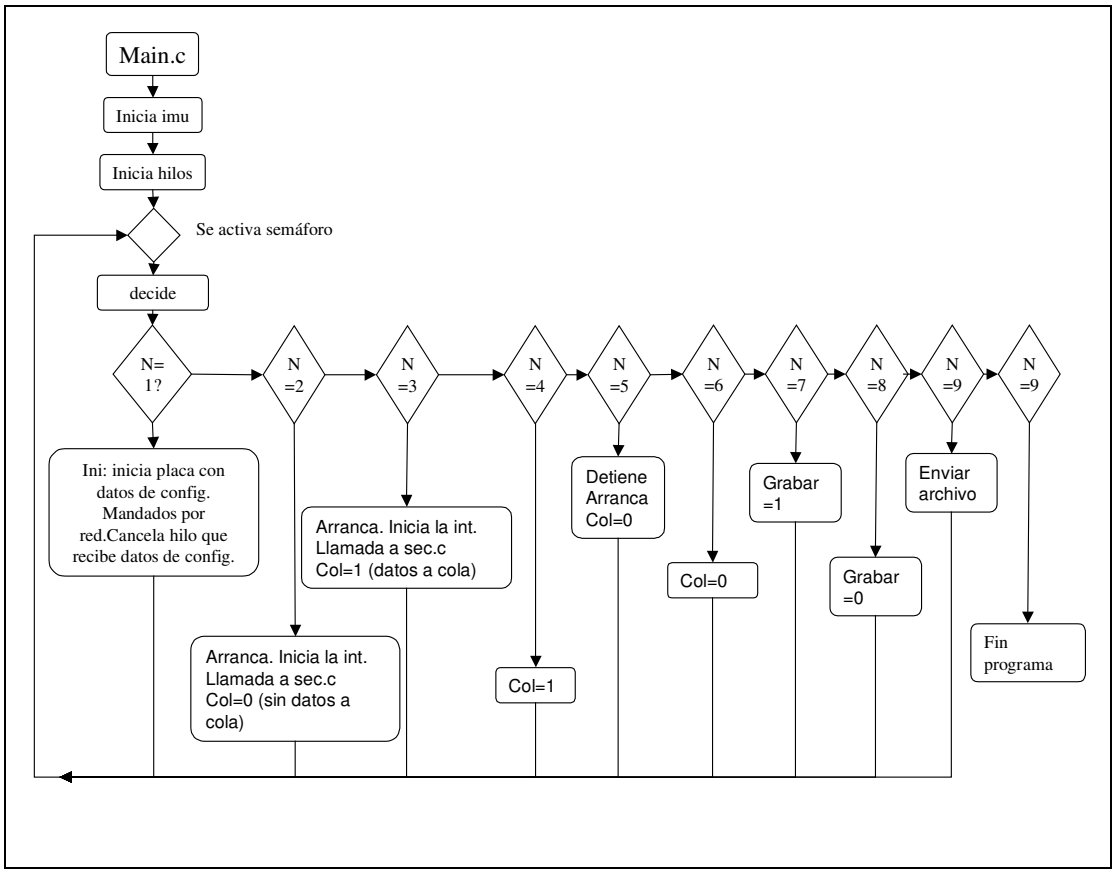


Ilustración 12: Diagrama de flujo de main.

4.2.1.1.5.2 Diagrama de interrupciones:

Cada tiempo de interrupción se llamará a la función *sec()*, y esta irá llamando a las siguientes funciones:

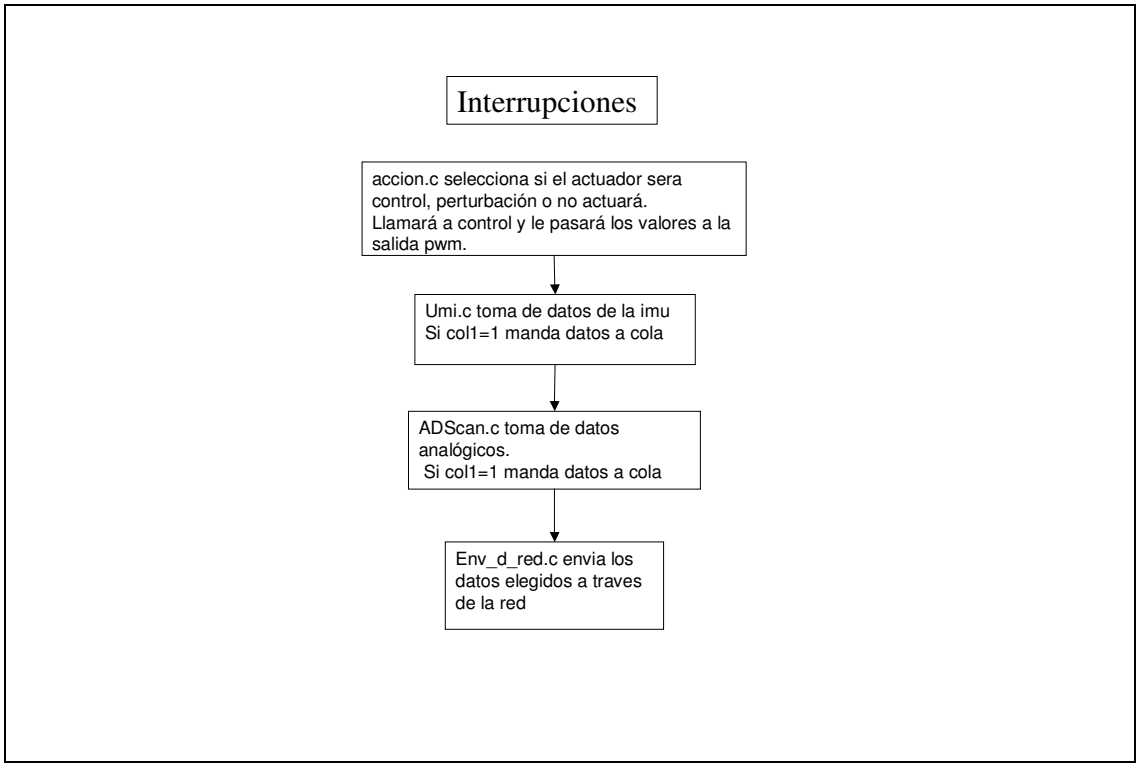


Ilustración 13: Diagrama de interrupciones.

4.2.1.1.5.3 Diagrama de d_disco:

Hilo que escribe los datos de la cola a disco, cuando la cola esta vacía y *terg=1* termina, siendo esperado por main.

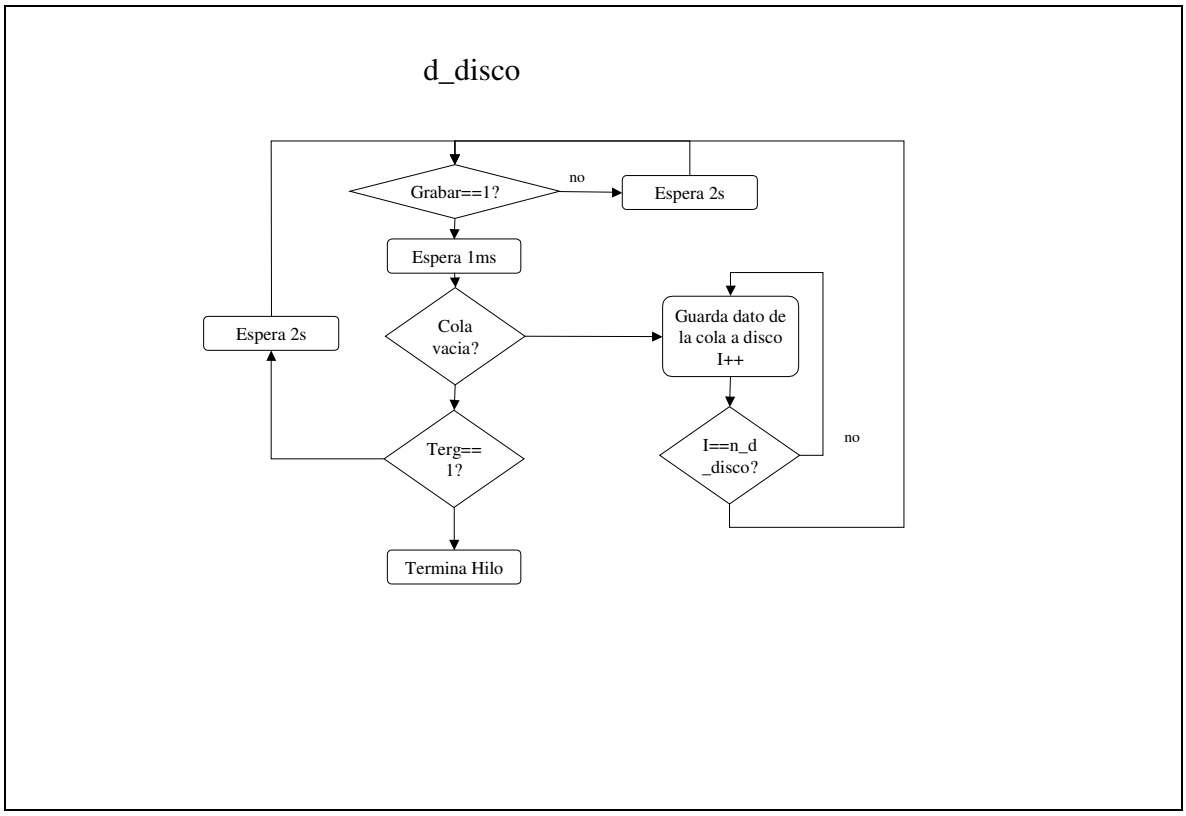


Ilustración 14: Diagrama de flujo de d_disco.

4.2.1.1.5.4 Diagrama de flujo de accion.c:

Programa en el que se realiza la acción de mandar la salida a los actuadores.

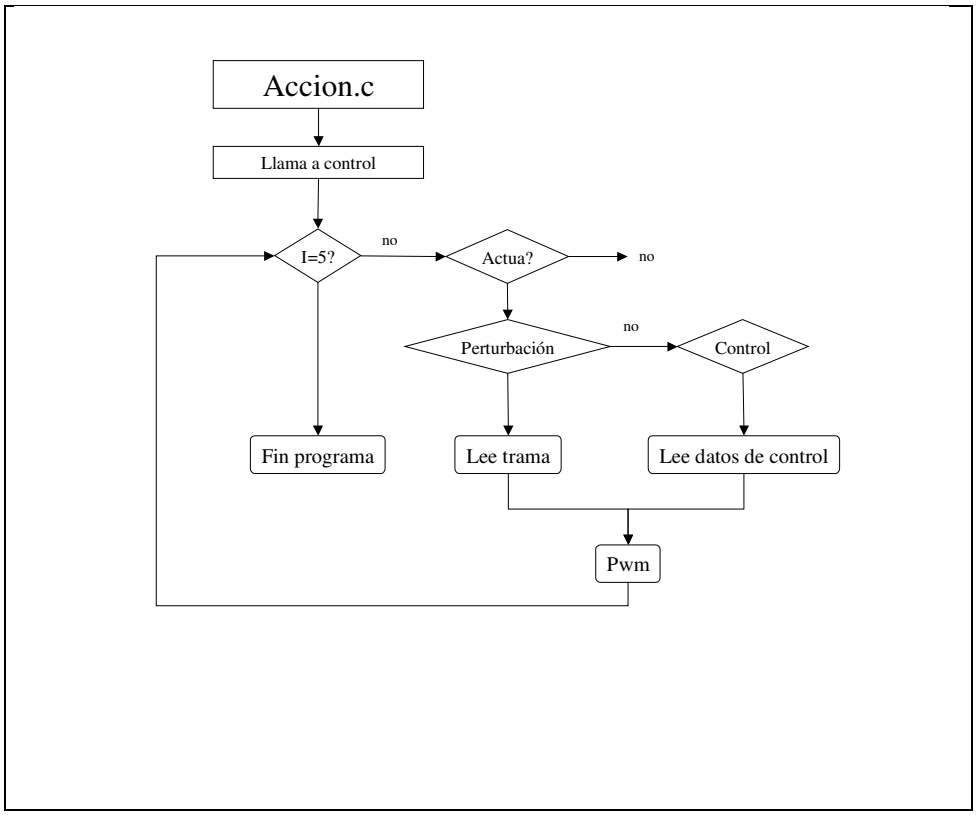


Ilustración 15: Diagrama de flujo de accion.

4.2.1.1.5.5 Diagrama de flujo de ind_cola.

Ind_cola es la función que gestiona los índices de la cola en la que son guardados los datos para luego mandarlos a disco.

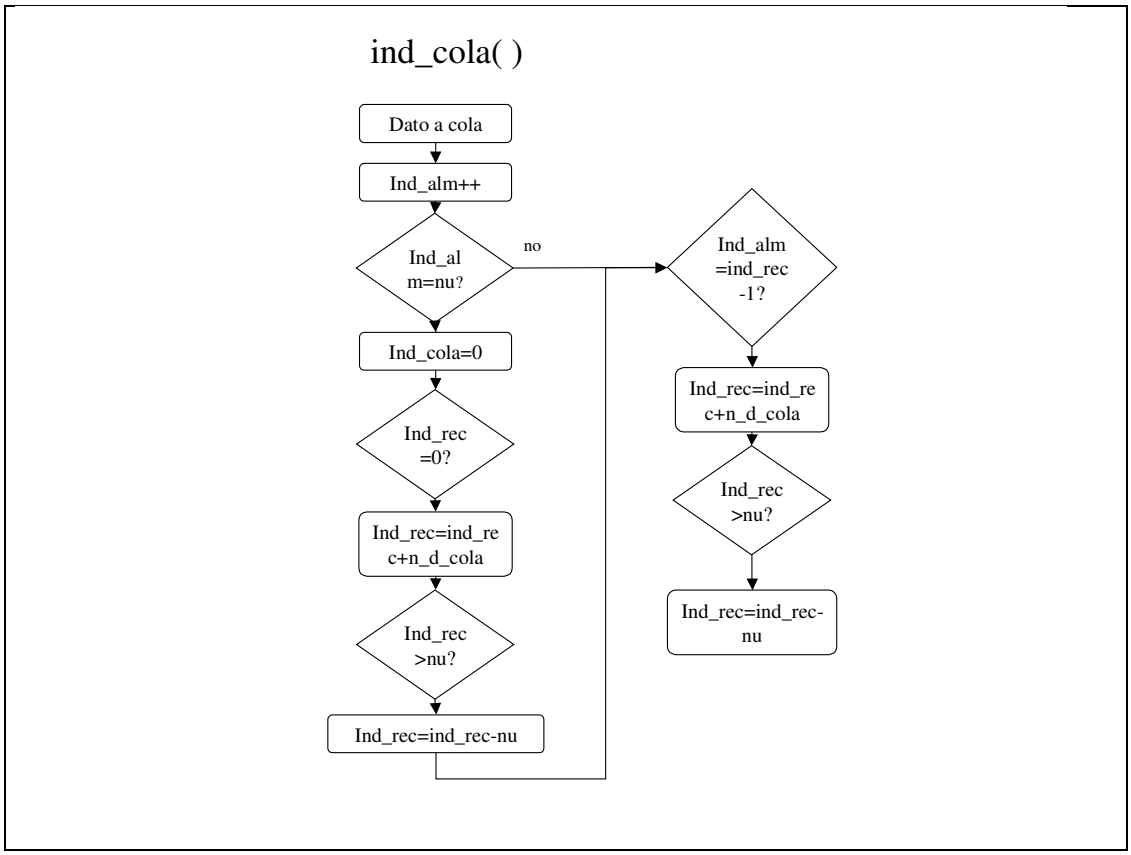


Ilustración 16: Diagrama de flujo de ind_cola.

4.2.1.1.5.6 Cronograma.

En el siguiente grafico se muestra un cronograma, donde podremos apreciar en que funciones gastamos el tiempo en el que invertimos una interrupción. El tiempo de cada interrupción varia en función de los datos que tomamos de la imu, variando desde 20ms (50Hz) leyendo los datos de pitch, roll y yaw a 33ms (30Hz) leyendo la matriz de quartenions que se compone de 4 elementos. Se representan los tiempos mínimos y máximos de estas funciones.

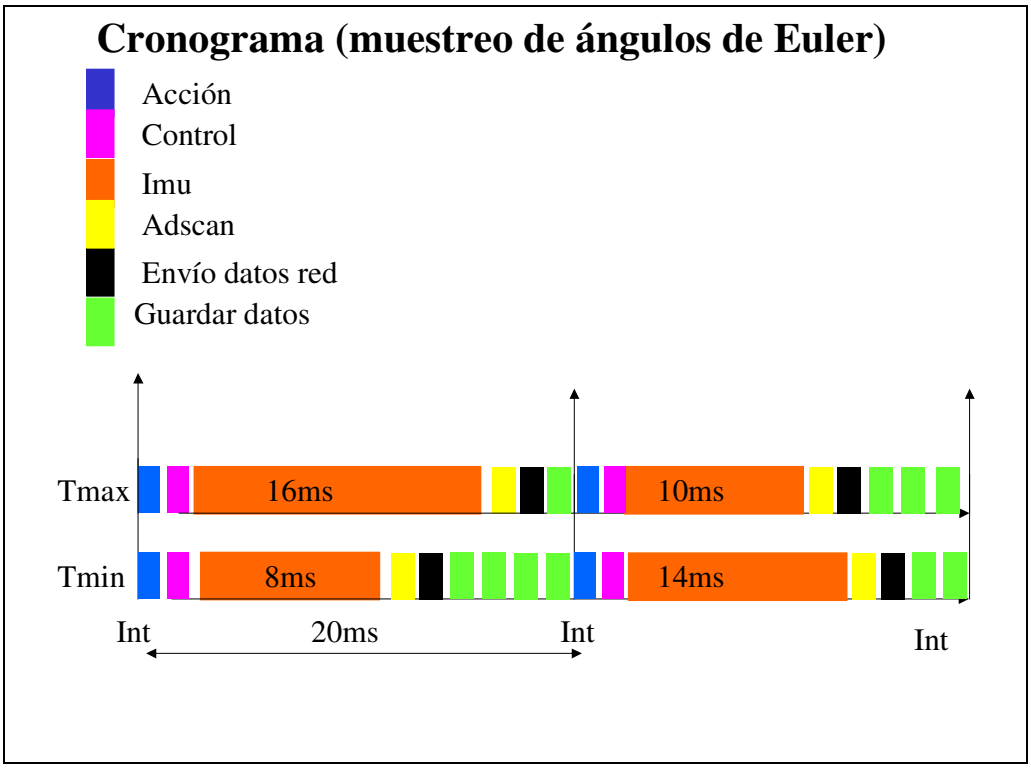


Ilustración 17: Cronograma.

4.2.1.1.6 Colas

Los datos que son muestreados y han sido seleccionados para ser enviados a disco pasan por una cola circular de tipo FIFO (primero en entrar primero en salir).
En esta cola son almacenados unos detrás de otros, así que en la lectura se tendrá en cuenta el orden en el que han sido introducidos para su identificación, así como el numero de datos introducidos en cada interrupción determinado en la variable n_datos_cola.
Los dos primeros datos escritos en cada interrupción serán el numero de interrupción y el tiempo tomado en esa interrupción desde el momento en que se arranco la maquina. Con estos datos podremos saber si nos hemos saltado alguna interrupción.
La profundidad de la cola se determinará al iniciar la ejecución del programa, haciendo una reserva dinámica de memoria.
Diapositiva de colas

4.2.1.2 PC tierra

Para realizar la teleoperacion se utilizará un PC, conectado al punto de acceso inalámbrico al que también se conectará la Hercules.
En un futuro el manejo de las ordenes a enviar se hará desde una ventana gráfica, todavía no implementada, pero parte de las mejoras futuras que se harán.

4.2.1.2.1 Matlab

Se utilizará Matlab para crear las tramas a mandar al helicóptero. Estas tramas serán secuencias de datos que utilizaremos como excitación en alguno de los actuadores, como parte de los primeros ensayos de control e identificación.

4.2.1.2.2 Control en línea de comandos

En estos momentos se controla al helicóptero desde tierra haciendo un control en línea de comandos.
Este control esta formado por los siguientes programas;

Control.c: gestiona la creación de hilos y hace las preguntas para seleccionar los datos a enviar. Llama a envía cuando ya tiene los datos.
Envía.c: se encarga de enviar por red al helicóptero los datos seleccionados en control. Tiene tres funciones diferentes;
Envia1: manda los datos de configuración.
Envia2: manda la variable de control n_control.
Envia3: manda las tramas para las excitaciones.
Hilo_t_red: hilo que recibe los datos que el helicóptero le manda por red, mandándolos a pantalla.
Recibe_archivo: función que recibe el archivo completo escrito en la Hercules, una vez terminado el experimento.
hilo_error: hilo que recibe los números de error enviados por la Hercules y los interpreta, mostrándonos por pantalla el texto con el error que ha llegado.

4.2.2 Hardware

4.2.2.1 Hercules

Nuestro sistema estará formado por la placa Hercules a la que le hemos conectado los sensores y actuadores. La placa se conectara con el ordenador de tierra a través de la red inalámbrica ethernet.
Los sensores utilizados son la IMU conectada a través del puerto RS-232, y sensores conectados a la entrada AD de la Hercules como son; velocidad del rotor, la batería, etc
Los actuadores utilizados son los 5 servomotores que actuarán sobre las variables del helicóptero que son; apertura del carburador, el rotor de cola, el colectivo y dos servos que regulan el cíclico longitudinal y lateral.
La comunicación helicóptero-tierra se realiza a través de la red ethernet. El helicóptero va dotado de un transmisor inalámbrico por el que recibirá las ordenes y mandara la información que el control de tierra le pida.

4.2.2.2 PC tierra

Esta maquina será un PC con Windows instalado y conexión ethernet para conectarse con el punto de acceso de la red. El PC de tierra nos servirá para teleoperar, monitorizar e identificar.
Utilizaremos windows para trabajar con matlab.

4.3 Programas

Se muestra el código de los programas principales ubicados en helicóptero.

4.3.1 Make

Primero se muestra el archivo make a través del que se han compilado los programas

```
#make para main.c

LIB=-L/usr/local/dscud5 -L/opt/dscud5 -ldscud5 -pthread -lm
INC=-I/usr/local/dscud5 -I/opt/dscud5
OBJ= env_errores.c borrado_archivo.c env_archivo.c accion.c control.c  env_d_red.c d_disco.c
ad.c umi.c  rec_d_red.c  sec.c m3dmgAdapter.c m3dmgUtils.c m3dmgSerialLinux.c

P=main\

all: $(P)

$(P):
        (gcc -o $@ $(OBJ) $@.c $(LIB) $(INC) )

clean:
        -rm -f $(P)
```

4.3.2 Main.c

Este archivo contiene a main, ini, arranca y parar.

```
//=====================================================================
// main1c.c
//=====================================================================
#include "cab_main.h"

//=====================================================================/
/ Name: main()
//        0.Inicializacion de la umi
//       II. Inicializacion Hilos reciben datos por red
//      III. Espero ordenes de tierra, segun le llegan los datos de red (activado pro semaforo)
//       IV. Finalizacion de programa
//      V. Ini().Inicializacion: Driver. Board. Parametros ad . Parametros interrupciones
//       VI. Arrancar()
//       VII. Parar()
//
//=====================================================================

int main( void )
{

        //variables de la umi
        static int parado=1;
        int portNum;
        int sn;
        short address, evalue;  // EEPROM address and data value
        char *fw;
        int tryPortNum = 1;
        //pwm
        int pwm=0;

        //=====================================================================
        //I.inicializacion de la umi ¡tiene que estar en el main!
```

```
        //=====================================================================
    portNum = m3dmg_openPort(tryPortNum, 115200, 8, 0, 1);   //open a port, map a device
    if (portNum<0) {
        printf("port open failed.\n");
        printf("Comm error %d, %s: ", portNum, explainError(portNum));
        env_errores(10);
                    return -1;//si no abre la imu que salga...
    }
    printf("Port is open as #%d - OK.\n", portNum);

    deviceNum = m3dmg_mapDevice(1, portNum);//map device - this is required!
    if (deviceNum <= 0) {
        printf("Could not map the Device to a Port\n");
        printf("%s\n", explainError(deviceNum));
                    env_errores(11);
        return -1;
    }
    printf("Device Number is #: %d\n\n", deviceNum);

    errorCode = m3dmg_getSerialNumber(deviceNum, &sn);//get serial number
    if (errorCode != M3D_OK)
            {
        printf("S/N Error - : %s\n", explainError(errorCode));
                    env_errores(12);
            }
    else
        printf("Serial #: %d\n", sn);
    printf("\n");

    fw = (char *) malloc( (size_t)12 );//get firmware number (as a string)
    errorCode = m3dmg_getFirmwareVersion(deviceNum, fw);
    printf("Firmware ");
    if (errorCode < 0)
            {
        printf("Error - : %s\n", explainError(errorCode));
                    env_errores(13);
```

```
                }
    else
        printf("#: %s\n", fw);
    printf("\n");

        //=====================================================================
        // I. DRIVER
        //=====================================================================

        if( dscInit( DSC_VERSION ) != DE_NONE )
        {
                dscGetLastError(&errorParams);
                fprintf( stderr, "dscInit error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
errorParams.errstring );
                env_errores(14);
                return 0;
        }

        //=====================================================================
        // II. BOARD
        //=====================================================================

        printf( "\nHERCULES BOARD INITIALIZATION:\n" );

        dsccb.boardtype = DSC_HERCEBX;
        dsccb.base_address =(WORD)0x240;
        dsccb.int_level = 5;

        if(dscInitBoard(dsccb.boardtype, &dsccb, &dscb) != DE_NONE)
        {
                dscGetLastError(&errorParams);
                fprintf( stderr, "dscInitBoard error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                env_errores(15);
                return 0;
        }
```

```
	//=====================================================================
	//a-d
	//=====================================================================
	memset(&dscadsettings, 0, sizeof(DSCADSETTINGS));
	dscadsettings.range = RANGE_10;
	dscadsettings.polarity = BIPOLAR;
	dscadsettings.gain = GAIN_1;
	dscadsettings.load_cal = (BYTE)TRUE;
	dscadsettings.current_channel = 0;
	dscadsettings.scan_interval = SCAN_INTERVAL_9;
	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;
	dscadscan.high_channel = N;
	dscadscan.gain = GAIN_1;

	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;
	}
```

```
//=====================================================================
//pwm
//=====================================================================

for (pwm=0;pwm<4;pwm++)
{
        memset(&pwmstruct[pwm], 0, sizeof(DSCPWM));
        pwmstruct[pwm].pwm_circuit=pwm;
        pwmstruct[pwm].polarity = 1;
        pwmstruct[pwm].output_enab = 1;
        pwmstruct[pwm].output_freq = 1;
}

//=====================================================================
// II. creacion hilos
//=====================================================================
pthread_create(&Hiloescribe,NULL,Hilo,NULL);//hilo.c que escribe datos a disco
pthread_create(&Hilo_t_h1,NULL,Hilo21,NULL);
pthread_create(&Hilo_t_h2,NULL,Hilo22,NULL);
pthread_create(&Hilo_t_h3,NULL,Hilo23,NULL);

//=====================================================================
// III. Espero ordenes de tierra,salgo si llega fin
//=====================================================================
printf("-------------------espero ordenes desde tierra--------------------\n");
while (!ter)
{
        sem_wait(&Semaforo1);//espero al semaforo que recibe desde tierra
        switch (n_control)
        {
                case 0: printf("llega 0 (trama)\n");break;
                case 1: printf("llega 1 (config)\n");ini();break;
                case 2: printf("llega 2 (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;
                        case 7: printf("llega 7 (datos a disco)\n");grabar=1;break;//graba datos a disco ;sem_post(&Semaforo)
                        case 8: printf("llega 8 (parar datos a disco)\n");grabar=0;break;//!!!!graba datos a disco ;sem_post(&Semaforo)
                        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;
                        default:printf("no llego opcion correcta\n");break;
                }
        }//fin while

        //=====================================================================
        //fin programa
        //=====================================================================

        // cancel interrupts manually if interrupts are still running
        if( dscCancelOp(dscb) != DE_NONE)
        {
                dscGetLastError(&errorParams);
                fprintf( stderr, "dscCancelOp error: %s %s\n", dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                env_errores(18);
                //return 0;
        }
```

```
        if( dscClearUserInterruptFunction(dscb) != DE_NONE)
        {
                dscGetLastError(&errorParams);
                fprintf( stderr, "dscClearUserInterruptFunction error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                env_errores(19);
                //return 0;
        }

        printf( "\nFinalizando\n" );
        dscFree();
        //========================================================================
        //fin umi
        //========================================================================
       free(fw);// free allocated memory
       m3dmg_closeDevice(deviceNum);// close device

        //paramos el pwm
        dscInp(dsccb.base_address + 12, &byteBuff);
        byteBuff ^= 0x10;  //Set DIOCTR0 to 0
        dscOutp(dsccb.base_address + 12, byteBuff);

        //fin de hilos
        //      sem_post(&Semaforo);//para que pueda entrar y terminar si no ha entrado
        pthread_join(Hiloescribe,&Retorno);//esperamos que acabe hiloescribe de grabar
        pthread_cancel(Hilo_t_h1);//cancelamos el hilo t_h(config)
        pthread_cancel(Hilo_t_h2);//cancelamos el hilo t_h
        pthread_cancel(Hilo_t_h3);//cancelamos el hilo t_h

        //fin de semaforos
        sem_destroy(&Semaforo1);   //destruyo el semaforo1

        printf( "nº de int=%d,int. con d. a disco=%d,d. a disco=%d,saltos cola=%d,enviados %d d.
red\n",counter,int_cola,s,salta,e);
```

```
	printf("d. calculados a disco=%d d. reales a disco=%d d. saltados %d suma %d\n",(int_cola*n_datos_cola),s,(salta*n_datos_cola),(s+(salta*n_datos_cola)));
	printf( "\nDSCUserInt completed.\n" );

	return 0;


} //fin main()

//=========================================================================/
/ Programa ini(void)
// Ini().Inicializacion: Driver. Board. Parametros ad . Parametros interrupciones
//=========================================================================

ini(void)//programa que inicializa el driver, la placa, los parametros AD, de interupcion
{
	int i;

	//=====================================================================
	// I. DRIVER
	//=====================================================================

	if( dscInit( DSC_VERSION ) != DE_NONE )
	{
		dscGetLastError(&errorParams);
		fprintf( stderr, "dscInit error: %s %s\n", dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
		env_errores(20);
		return 0;
	}

	//=====================================================================
	// II. BOARD
	//=====================================================================

	printf( "\nHERCULES BOARD INITIALIZATION:\n" );
```

```
	dsccb.boardtype = DSC_HERCEBX;
	dsccb.base_address =(WORD)0x240;
	dsccb.int_level = 5;

	if(dscInitBoard(dsccb.boardtype, &dsccb, &dscb) != DE_NONE)
	{
		dscGetLastError(&errorParams);
		fprintf( stderr, "dscInitBoard error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
		env_errores(21);
		return 0;
	}

	//=====================================================================
	// Interrupcion
	//=====================================================================
	counter = 0;// set the counter to 0
	dscuserint.intsource = 0;
	dscuserint.counter = 0;
	dscuserint.rate = 1.1f;
	dscuserint.clksource = (BYTE) 0;
	dscuserint.rate = (FLOAT)datos.frec;
	t_int=1000/(FLOAT)datos.frec;
	//sumamos los datos que se introducen en la cola
	n_datos_cola=0;//lo ponemos a 0 por si se ha ejecutado ini anteriormente!
	//datos de la imu
	for (i=0;i<M;i++)
	{
	if(datos.umi[i].umi_disco)
	if(i==3)
	n_datos_cola=n_datos_cola+4;
	else
	n_datos_cola++;
	}
	//datos de ad
```

```
	for (i=0;i<N;i++)
	{
	if(datos.ad[i].ad_disco)
	n_datos_cola++;
	}
	n_datos_cola=2+n_datos_cola;//se le suman el numero de la interrupcion y el tiempo
	printf("n_datos_cola= %d\n",n_datos_cola);
	//reserva de memoria

	NU=datos.t_cola;
	p=(float*)malloc(NU*sizeof(float));
	if (p==NULL)
	{
		printf("memoria insuficiente\n");
		env_errores(22);
		exit(0);
	}


}//fin ini

//=============================================================================
// Programa arranca(void). arranca las interrupciones
//=============================================================================

arranca (void)//arranca
{
	int i=0;
	printf("arranca\n");

	//inicializo los valores de pwm=0 (daba problemas, E1 sacaba valores sin haberselo pedido)
	for(i=0;i<4;i++)
	pwmstruct[i].duty_cycle=0;

	//esta es la llamada al programa sec
```

```
	if ( dscUserInt(dscb, &dscuserint, sec) != DE_NONE )
	{
		dscGetLastError(&errorParams);
		fprintf( stderr, "dscUserInt error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
		env_errores(23);
		return 0;
	}
}//fin arranca


parar (void)
{
	printf("paramos las interrupciones\n");
	// cancel interrupts manually if interrupts are still running
	if( dscCancelOp(dscb) != DE_NONE)
	{
		dscGetLastError(&errorParams);
		fprintf( stderr, "dscCancelOp error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
		env_errores(24);
		return 0;
	}


	if( dscClearUserInterruptFunction(dscb) != DE_NONE)
	{
		dscGetLastError(&errorParams);
		fprintf( stderr, "dscClearUserInterruptFunction error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
		env_errores(25);
		return 0;
	}

}
```

4.3.3 sec.c

Programa que es llamado cada interrupción. Este hace las llamadas a los distintos programas.

```
//sec.c programa que es llamado cada interrupcion. este hace las llamadas a los distintos programas...
#include <stdio.h>
#include <time.h>
#include "dscud.h"

//placa
extern BYTE result;   // returned error code
extern DSCB dscb;     // handle used to refer to the board
extern DSCCB dsccb;   // structure containing board settings
extern ERRPARAMS errorParams;       // structure for returning error messages
//variables externas
extern int counter;
extern DWORD t[10];
//variables internas
extern DWORD t_int;

void sec(void* params)
{

        static DWORD t_ant=0.0;
        t_ant=t[0];
        counter ++;
        printf("interrupcion %d\n",counter);
        dscGetTime(&t[0]);
        if((t[0]-t_ant)>(t_int+2))//el +2 es por los errores
        {
                printf("saltada interrupcion!!!!!!!!!!!! %d\n",counter);
                env_errores(100);
```

```
        }
        accion();
        dscGetTime(&t[1]);
        umi1();//llamada a umi.c
        dscGetTime(&t[2]);
        ad1();//llamada a ADScan1c.c
        dscGetTime(&t[3]);
        env_d_red();
        dscGetTime(&t[4]);
        printf("t.control %u, t.umi %u, t.a_d %u, t.env_d_red %u\n",t[1]-t[0],t[2]-t[1],t[3]-t[2],t[4]-t[3]);
}
```

4.3.4 umi.c

Archivo que contiene el programa umi e ind_cola

```
//inicializacion y cierre de la imu están en el main
#include <stdio.h>
#include "m3dmgErrors.h"
#include "m3dmgSerial.h"
#include "m3dmgAdapter.h"

#include <time.h>
#include "dscud.h"
#include "cab_v_red_ext.h"
#define MAX 1000 //esto es para la cola

//variables globales
        //umi
                extern int errorCode, deviceNum;
                extern float quat[3];      //  quaternions
                extern float roll, pitch, yaw;        //Euler
        //control
                extern DWORD t[10];
```

```
		extern int NU,counter,col,col1,int_cola;
		extern int ind_alm, ind_rec;//variables de la cola. se utilizan las mismas para todas las colas
		extern int n_datos_cola,salta;
		extern float *p;


void umi1c(void* params)
{
	//variables de la umi. minimizar!
		//float mag[3];	//  magetic
		//float accel[3];	//  acceleration
		//float angRate[3];	//  angular rate
		//char *axis[]={ "X", "Y", "Z"};
		//float quat[3];	//  quaternions
		//float xform[3][3];	//  transformation matrix
		//float roll, pitch, yaw;
		//extern short address, evalue;	// EEPROM address and data value
		//float raw[30];
	col1=col;//evito que col1 cambie entre la ejecucuion de umi y ad, lo que me daria un faloo en la cola
	if (col1)//escribo en la cola el nº de interrupción y el tiempo
	{
		//printf("escribiendo en cola\n");
		int_cola++;
		//counter contador de las interrupciones
		*(p+ind_alm)=counter;
		ind_cola();
		//tiempo
		*(p+ind_alm)=t[1];
		ind_cola();
	}
	 // Euler angles - instantaneous
	if((datos.umi[0].umi)|(datos.umi[1].umi)|(datos.umi[2].umi))
	{
```

```
		errorCode = m3dmg_getEulerAngles(deviceNum, &pitch, &roll, &yaw,
M3D_INSTANT);
	printf("Instantaneous Euler angles\n");
	if (errorCode < 0)
			printf("Error - : %s\n", explainError(errorCode));
		else
		{
			printf("Pitch : %4.2f\n", pitch);
			printf("Roll  : %4.2f\n", roll);
			printf("Yaw   : %4.2f\n", yaw);
		}
	printf("\n");

		//aqui va el código que escribe en la cola

		if (col1)
		{

			//pitch
			if (datos.umi[0].umi_disco)
			{
				*(p+ind_alm)=pitch;
				ind_cola();
			}
			//roll
			if (datos.umi[1].umi_disco)
			{
				*(p+ind_alm)=roll;
				ind_cola();
		}
			//yaw
			if (datos.umi[2].umi_disco)
			{
				*(p+ind_alm)=yaw;
				ind_cola();
			}
```

```
                }//fin if(col==1)
        }//fin if datos umi p,r,y

}//fin umi

ind_cola(void)

{
        ind_alm++;
        if (ind_alm==NU)//caso limite de ind alm
        {
                ind_alm=0;
                if (ind_rec==0)
                {
                        ind_rec=ind_rec+n_datos_cola;//salta segun su formato
                        salta++;
                        printf("cola llena\n");
                if (ind_rec>=NU) ind_rec=ind_rec-NU;   //caso limite de ind rec
                }
        }
        if (ind_alm==ind_rec-1)//la cola esta llena
        {
                ind_rec=ind_rec+n_datos_cola;//salta segun su formato
                salta++;
                printf("cola llena\n");
                if (ind_rec>=NU) ind_rec=ind_rec-NU;   //caso limite de ind rec
        }
        printf("ind_alm=%d ind_rec=%d\n",ind_alm,ind_rec);
}
```

4.3.5 AD.c

Programa que lee el canal analógico de la Hercules y hace la conversión digital.

```
#include <stdio.h>
#include <time.h>
#include "dscud.h"
#include "cab_v_red_ext.h"

//placa
	extern BYTE result;   // returned error code
	extern DSCB dscb;     // handle used to refer to the board
	extern DSCCB dsccb;   // structure containing board settings
	extern ERRPARAMS errorParams;       // structure for returning error messages
//AD
	extern DSCSAMPLE sample;            // sample reading
	extern DSCADSETTINGS dscadsettings; // structure containing A/D conversion settings
	extern DSCADSCAN dscadscan;   // structure containing A/D scan settings
	extern DSCSAMPLE* samples;    // sample readings
//variables externas
	extern int col1,ind_alm, ind_rec;//variables de la cola. se utilizan las mismas para todas las
colas;
	extern float *p;

AD1c (void)
{
	int i;
	DFLOAT voltage;
//=====================================================================
//ADscan
//=====================================================================

	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;
        }

        printf( "\nSample readouts:" );

        for( i = 0; i < (dscadscan.high_channel - dscadscan.low_channel)+ 1; i++)
                printf( " %hd", dscadscan.sample_values[i] );

        printf( "\nActual voltages:" );

        for( i = 0; i < (dscadscan.high_channel - dscadscan.low_channel)+ 1; i++)
        {
                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;
                }

                printf( " %5.3lfV", voltage);
        }
    printf("\n");

        //aqui va el código que escribe en la cola

        if (col1)
        {
                for(i=0;i<N;i++)
                {
                        if (datos.ad[i].ad_disco)
                        {
```

```
                                        *(p+ind_alm)=dscadscan.sample_values[i];
                                        ind_cola();
                                    }
                                }
                    }//fin if(col==1)
}//fin AD
```

4.3.6 Accion.c

Programa que gestiona las salidas de los actuadores.

```
#include <stdio.h>
#include "dscud.h"
#include "cab_v_red_ext.h"

//placa
extern BYTE result;   // returned error code
extern DSCB dscb;     // handle used to refer to the board
extern DSCCB dsccb;   // structure containing board settings
extern ERRPARAMS errorParams;       // structure for returning error messages
extern int tram[4][8];
//pwm
extern DSCPWM pwmstruct[4];  // estructura pwm para los 4 servos
extern void control(int*);

accion()
{
        int salida[5];
        static int ac=0,j=0;
        static int tra[4]={0,0,0,0};
        BYTE byteBuff;    // byte variable

        //habilita el puerto E como PWM
        dscInp(dsccb.base_address + 12, &byteBuff);
        byteBuff |= 0x10;  //Set DIOCTR0 to 1
```

```
dscOutp(dsccb.base_address + 12, byteBuff);


//llama a control que calcula los 5 actuadores
control(salida);
//actuadores
for (ac=0;ac<5;ac++)
{
        if (datos.a[ac].actuador!=0)//llamaremos al actuador si es distinto de cero
        {
                printf("\n entro con el actuador %d\n",ac);
                if (datos.a[ac].actuador==2)//en este caso es una excitacion
                {
                        pwmstruct[ac].duty_cycle = tram[ac][tra[ac]];//esta es la salida que
le enviamos
                        printf("actuador [%d][%d]=[%d]\n",ac,tra[ac],tram[ac][tra[ac]]);
                        tra[ac]++;
                        if (tra[ac]==8) tra[ac]=0;
                }

                else//es un control
                {
                        pwmstruct[ac].duty_cycle = salida[ac];//aqui esta el tema
                        printf("actuador %d salida=%d\n",ac,salida[ac]);
                }//fin else

                //una vez calculado el valor de pwm, llamamos a la funcion
                if( dscPWMFunction(dscb, &pwmstruct[ac]) != DE_NONE)
                {
                        dscGetLastError(&errorParams);
                        fprintf( stderr, "dscPWMFunction error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                        //return 0;
                        env_errores(0);
                }
```

```
                }//fin if
        }//fin for

} //fin accion1c
```

4.3.7 Control.c

Programa que calcula el control para los 5 actuadores.

```
//control.c devuelve el valor salida
#include <stdio.h>
#include <math.h>
#include "dscud.h"
#include "cab_v_red_ext.h"

//variables externas
//umi
extern float pitch, roll, yaw;
extern float quat[3];
//ad
extern DSCADSCAN dscadscan;   // estructura con datos A/D


void control(int k[])
{

        //aqui iria el codigo de los controladores
        int i;
        for(i=0;i<5;i++)
        k[i]=10+i;

}
```

4.3.8 d_disco.c

```
//d_disco.c hilo que escribe los datos de la cola al archivo
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include "dscud.h"


extern float *p;
extern int ind_alm, ind_rec;//variables de la cola
extern int terg,s,NU,n_datos_cola,grabar;
extern DWORD t[10];

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");
				//exit(1);
				env_errores(3);
	}

	while (1)
	{
		while (grabar)
		{
			dscGetTime(&t[5]);
			dscSleep(1);//para que sec continue con las interrupciones

	//=====================================================================
			// los datos que hayan sido seleccionados estaran en la cola en el orden
apropiado
```

```
                    // luego al sacarlos tendran ese orden, sacandolos en multiplos de los
introducidos.
                    // el dato del numero de elementos estara en la variable n_datos_cola.

        //=====================================================================
                    if (ind_rec==ind_alm)//la cola esta vacia
                    {
                            printf("cola vacia\n");
                            //dscSleep(500);
                            if (terg)//termina el programa
                            {
                                    printf("cola finalizada\n");
                                    break;
                            }
                            else
                            dscSleep(2000);
                    }
                    else //la cola no esta vacia
                    {

                            for (i=0;i<n_datos_cola;i++)
                            {
                            fwrite((p+ind_rec),4,1,fo);
                                    ind_rec++;
                                    if (ind_rec==NU)//actualizacion 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)
                    break;
```

```
                else
                        dscSleep(2000);
        }//fin while, acabo la cola y termino programa
        printf("d_disco: cierro el archivo\n");
        fclose(fo);//cierro el archivo
        //return s;
}//fin de d_disco
```

4.3.9 rec_d_red.c

```
//hilo que recibe datos de tierra, abre semaforo cuando llegan
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include <sys/timeb.h>
#include <time.h>
#include "dscud.h"
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
//red
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include "define.h"
```

```
#define MYPORT1 4951 // puerto t-h (config)
#define MYPORT2 4952 // puerto t-h (n_control)
#define MYPORT3 4953 // puerto t-h (trama)

#include "cab_v_red_ext.h"

sem_t Semaforo1;      //semaforo ordenes tierra
extern int n_control,ta,tr;
extern int tram[4][8];

void* Hilo21(void*P)//hilo que recibe datos de tierra (config), abre semaforo cuando llegan
{
        while (1)
        {

                int sockfd;
                struct sockaddr_in my_addr; // informacin sobre mi direccin
                struct sockaddr_in their_addr; // informacin sobre la direccin 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; // Ordenacin de bytes de mï¿½uina
                my_addr.sin_port = htons(MYPORT1); // short, Ordenacin de bytes de la red
                my_addr.sin_addr.s_addr = inet_addr(MYDIR); // rellenar con mi direccin
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);

		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

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; // informacin sobre mi direccin
		struct sockaddr_in their_addr; // informacin sobre la direccin 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; // Ordenacin de bytes de mï¿½uina
        my_addr.sin_port = htons(MYPORT2); // short, Ordenacin de bytes de la red
        my_addr.sin_addr.s_addr = inet_addr(MYDIR); // rellenar con mi direccin
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);
                //exit(1);
        }

        addr_len = sizeof(struct sockaddr);

        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

void* Hilo23(void*P)//hilo que recibe datos de tierra (trama), abre semaforo cuando llegan
{
```

```
while (1)
{
	int trama[9];
	int sockfd;
	struct sockaddr_in my_addr; // informacin sobre mi direccin
	struct sockaddr_in their_addr; // informacin sobre la direccin del cliente
	int addr_len, numbytes3;


	if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
	{
		perror("socket");
		env_errores(32);
		//exit(1);
	}

	my_addr.sin_family = AF_INET; // Ordenacin de bytes de mï¿½uina
	my_addr.sin_port = htons(MYPORT3); // short, Ordenacin de bytes de la red
	my_addr.sin_addr.s_addr = inet_addr(MYDIR); // rellenar con mi direccin 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(33);
		//exit(1);
	}

	addr_len = sizeof(struct sockaddr);

	if ((numbytes3=recvfrom(sockfd,&trama, sizeof(trama), 0,
		(struct sockaddr *)&their_addr, &addr_len)) == -1)
	{
		perror("recvfrom");
```

```
                env_errores(34);
                //exit(1);
        }

        ta=trama[0];
        for(tr=0;tr<8;tr++)
        {
                tram[ta][tr]=trama[tr+1];
                printf("tram [%d][%d]=[%d]\n",ta,tr,tram[ta][tr]);
        }

        n_control=0;//indica que ha llegado trama
        sem_post(&Semaforo1);//abro el semaforo
        close(sockfd);

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

4.3.10 Cabeceras.

Aqui se incluyen las cabeceras utilizadas. Estas han sido organizadas asi para hacer mas comodo y limpio el manejo de los archivos.

4.3.10.1 Cab_main.h

Esta cabecera solo la utliza main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include <sys/timeb.h>
#include <time.h>
#include "dscud.h"

#include "m3dmgErrors.h"
```

```
#include "m3dmgSerial.h"
#include "m3dmgAdapter.h"
#include "m3dmgUtils.h"

#include <pthread.h>
#include <semaphore.h>

#define SLEEP_TIME 10
#define MAXBUFLEN 10//para la red
#include "cab_v_red.h"
#include "define.h"

// declaracion de variables de main
        BYTE result;   // returned error code
        DSCB dscb;     // handle used to refer to the board
        DSCCB dsccb;   // structure containing board settings
        ERRPARAMS errorParams;       // structure for returning error messages
        DSCUSERINT dscuserint;    // userint setup structure
        BYTE byteBuff;    // byte variable
        int intBuff;      // integer variable
        long longBuff;     // long variable
        float floatBuff;  // float variable
//AD
        DSCSAMPLE sample;            // sample reading
        DSCADSETTINGS dscadsettings; // structure containing A/D conversion settings
        DSCADSCAN dscadscan;   // structure containing A/D scan settings
        DSCSAMPLE* samples;    // sample readings

//variables de umi (utilizadas en main y umi,env_d_red)
        int deviceNum,errorCode;
        float quat[3];      //  quaternions
        float roll, pitch, yaw;
//variables externas
        DWORD t[10],t_int;//utilizado en sec.c
        int counter;//utilizado en sec.c
        int terg,ter,s;//utilizado en d_disco.c
```

```
        DFLOAT voltage;//AD
//variables de reserva de memoria
        int NU;
        float *p;
        int int_cola;
        int ind_alm, ind_rec;
//variables de red
        sem_t Semaforo1;      //semaforo ordenes t_h
        char buf[MAXBUFLEN];
        int n_control,col,col1,e,n_datos_cola,salta,grabar,ta,tr;
        int tram[4][8];//tramas
//variables de pwm
DSCPWM pwmstruct[4];  // estructura pwm para los 4 servos

//declaracion de hilos
        pthread_t Hiloescribe;//hilo.c que escribe datos a disco
        void *Retorno;
        pthread_t Hilo_t_h1;//rec_d_red1c.c recibe datos de tierra
        void *Retorno21;
        pthread_t Hilo_t_h2;
        void *Retorno22;
        pthread_t Hilo_t_h3;
        void *Retorno23;

//declaracion de funciones
        void sec(void* params);  // user defined function
        extern void* Hilo(void*P); //declaracion externa hilo que se escribe los datos de la cola al
archivo
        extern void* Hilo21(void*P);//declaracion externa del hilo que recibe datos de tierra, abre
semaforo cuando llegan
        extern void* Hilo22(void*P);//declaracion externa del hilo que recibe datos de tierra, abre
semaforo cuando llegan
        extern void* Hilo23(void*P);//declaracion externa del hilo que recibe datos de tierra, abre
semaforo cuando llegan
```

4.3.10.2 Cab_v_red.h

Esta cabecera es usada por main y contiene las declaraciones de las variables que intercambiamos por red con tierra.

```
#include "define.h"

struct umi{
unsigned int umi:1;
unsigned int umi_disco:1;
unsigned int umi_red:1;
}umi[M];//datos a capturar

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

struct act{
int actuador;
//int perturbacion;
int control;
//char pos_p;
//float pos_previo;
float p1;
float p2;
float p3;
//int frec;
}a[4];//accion del actuador

struct tierra{
int d0;
int t_cola;
int frec;
struct  umi umi[M];
```

```
struct ad ad[N];
struct act a[4];
}datos;
```

4.3.10.3 cab_v_red_ext.h.

Esta cabecera es igual a la anterior, solo que sera usada por los otros programas que no son main, por lo tanto en su declaracion se añade el modificador *extern*.

```
#include "define.h"

extern struct umi{
unsigned int umi:1;
unsigned int umi_disco:1;
unsigned int umi_red:1;
}umi[M];//datos a tomar

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

extern struct act{
int actuador;
//int perturbacion;
int control;
//char pos_p;
//float pos_previo;
float p1;
float p2;
float p3;
//int frec;
}a[4];//accion del actuador
```

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

4.3.10.4 define.h

En este archivo se incluyen las variables que pueden ser modificadas antes de compilar, con objeto de ser encontradas y modificadas con rapidez.

```
//archivo en el que se incluyen las variables a modicar antes de compilar de helicoptero

#define M 3 //numero de datos de la imu que muestreamos
#define N 1 //numero de datos analogicos que muestreamos
#define NDATOS 10  //numero de datos que mandamos de una vez en envio_archivo.c
#define DIR "127.0.0.1" //direccion a la que mandamos los datos. usada por env_d_red.c
#define MYDIR "127.0.0.1"  //direccion de nuestra maquina. usada por rec_d_red.c
#define MAX 10   //numero maximo de variables a enviar por red en env_red.c
```