7 Apéndice

7.1 Apéndice A: Problemas ocurridos

Dentro de esta sección se exponen algunos problemas o incompatibilidades ocurridas durante la puesta a punto del software.
Algunas de ellas llevaron mucho tiempo en resolverlas, así que aquí va una lista para no volver a caer en ellas.
La resolución de estos pequeños problemas es lo que conforma el proyecto, lo que acota su dirección y desgraciadamente en lo que se han invertido mucho tiempo.
Estos problemas nacen de la puesta a punto de la placa Hercules, las funcionalidades de su driver, tales como interrupciones y toma de datos analógicos, del uso de los programas de toma de datos de la imu, y de la inclusión de programación concurrente a través de los hilos.

7.1.1 Incompatibilidad entre la flash disk, el disco duro y lector de CD

Aunque en el manual dice que es posible conectar un IDE a la placa de la flash disk, cada vez que se intento fallaba, por lo tanto utilizamos la placa de la flash en el IDE primario y el disco duro y lector de CD en el IDE secundario, no reportando esto ningún problema.

7.1.2 Inicialización de la umi

La inicialización de la umi tiene que estar en el main, sacándolo de ahí no funciona. Por eso las variables de la umi son globales.
Esto hace que de la forma actual no se pueda utilizar RTLinux, ya que este sistema operativo funciona a base de “módulos”, no de programas con un main como la umi exige en los programas que utilizamos.

7.1.3 Creación de hilos e inicialización de parámetros AD

La creación de hilos tiene que hacerse después de iniciar los parámetros ad de la placa. En caso contrario dará “segmentation fault”.
Es por ello por lo que el numero de parámetros AD a muestrear lo tenga impuesto desde el inicio, y luego se gestionaran a cola los deseados. Pero se muestrean un numero “fijo” por software. En caso de querer modificar esta variable abría que modificar el main del programa, aunque esto no reporta ningún problema debido a que muestrear un elevado numero por defecto no trae consigo un aumento del tiempo de muestreo.

7.1.4 Necesidad de ejecutar el programa como root

El programa de la umi necesita ser ejecutado por root, es decir por el administrador. Por lo tanto necesitaremos entrar en el sistema operativo directamente como root, ya que no tendremos periféricos para cambiar de usuario.

7.1.5 Inicialización de la placa

Los parámetros iniciales de la placa deben ser iniciados en el main.

7.1.6 Sobre el uso del semáforo

El semáforo se usa para que main pueda quedarse parado en un punto, y ejecute el switch (n_control) cuando este semáforo le de paso.
Cada vez que se abra el semáforo entrará una vez en el switch y no saldrá de él hasta que no le llegue la orden de fin de programa *ter=1*.

7.1.7 Problemas de robustez

Durante los ensayos han ocurrido los siguientes problemas de robustez, derivados de la llegada de datos por red y solucionados de la siguiente forma;

7.1.7.1 Modificación de la variable col

Esta variable posibilita el envió de datos a cola, con la llegada de col=1 por red. Si se pone col=1 después de haberse ejecutado umi pero antes de ad ocurrirá un error de secuencia en la cola, perdiendo los datos su posición establecida. Para solucionarlo de una manera simple he creado una variable col1, que se iguala con col antes de ejecutarse umi, por lo que un cambio afectaría a todos los datos en la misma interrupción.

7.1.7.2 Llegada de datos de configuración

La llegada de datos de configuración una vez que se han iniciado las interrupciones daría “segmentation fault” y el programa se pararía. Para evitarlo, una vez que se ejecuta arranca (que da paso a las interrupciones) se cancela el hilo que recibe los datos de configuración.
Esto hace que no se pueda cambiar los datos de configuración una vez empezado el ensayo, pero SÍ si aun no se ha dado la orden de comenzar las interrupciones.

7.1.7.3 Llega la orden de arrancar dos veces

Si llega la orden de arrancar y se ejecuta el programa *arranca()* dos veces sin antes haberlo parado dará un error y se detendrá el ensayo.
Para solucionar esto existe una variable *parado* que esta = 1 cuando esta parado y = 0 cuando están funcionando las interrupciones, así si llega arranca la segunda vez al estar parado = 0 hará que arranca no haga nada y no se estropee el experimento. Por lo tanto esta variable es = 1 al inicio, para permitir la ejecución de arranca.

7.2 Apéndice B: Instalación de RTLinux.

Para poder ejecutar una aplicación en tiempo real estricto necesitamos tener un Sistema Operativo que soporte el tiempo real. En Linux esto quiere decir que los programas se ejecutarán como módulos, en los que el modulo tomara el control

del kernel y de las interrupciones y así estas no tendrán retrasos debidos a actividades “secundarias” o no de tiempo real

Para convertir el Sistema Operativo RedHat Linux a un sistema en tiempo real necesitamos hacer los siguientes pasos:

7.2.1 Archivos necesarios

Bajar los siguientes archivos. Por ejemplo a /usr/src.

Kernel de Linux 2.4.22

desde:

http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.22.tar.bz2

rtlinux-3.2-pre3: parche para el kernel 2.4.22 desde: ftp://ftp.rtlinux.at/pub/rtlinux/contrib/hofrat/kernel_patch-2.4.22-rtl3.2-pre3

rtlinux desde:

ftp://ftp.rtlinux.at/pub/rtlinux/contrib/hofrat

7.2.2 Instalar:

Descomprimimos el kernel: tar xvfj linux-2.4.44.tar.bz2

mv linux-2.4.22 linux-2.4.22-rtl3.2-pre3

ln –s linux-2.4.22-rtl3.2-pre3 linux

descomprimimos el parche: tar xfvj rtlinux-3.2-pre3.tar.bz2

ln –s rtlinux-3.2-pre3 rtlinux

7.2.3 Parchear el kernel

Para parchear el kernel ejecutaremos las siguientes órdenes:

cd /usr/src/linux

patch –p1 --dry-run < ../kernel_path-2.4.22-rtl3.2-pre3 (testeamos)

patch –p1 < ../kernel_path-2.4.22-rtl3.2-pre3 (ahora lo parcheamos)

7.2.4 Configurar el Kernel

cp /usr/src/boot/config-2.4.22

make menuconfig (sin cambiar nada)

7.2.5 Compilar el Kernel

Lo hacemos siguiendo el siguiente proceso;
make dep
make bzImage
make modules
make modules_install
cp System.map /boot/System.map-2.4.22-rtl3.2-pre3
cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.22-rtl3.2-pre3

7.2.6 Configurar el gestor de arranque

Para que el sistema pueda botar desde rtlinux tendremos que modificar el gestor de arranque. En RedHat es Grub, tendremos que modificarlo para que quede;
Title RTLinux (2.4.22-3.2-pre3)
Root (hd0,0)
Kernel /vmlinuz-2.4.22-rtl3.2-pre3 apm=off acpi=off single

7.2.7 Reiniciar

Elegir el kernel de RTLinux

7.2.8 Instalar RTLinux

cd /usr/src/rtlinux
make menuconfig (no hacer cambios, guardar y salir)
make dep
make
make devices

7.2.9 Probar ejemplos

Una vez instalado y configurado solo queda probar los ejemplos en los que los programas ya no tendran la forma habitual (con main) sino que serán módulos.

7.3 Apéndice C: Glosario

7.3.1 kernel

Los sistemas operativos modernos están construidos en capas, cada una añadiendo nuevas capacidades, como técnicas de acceso a disco o interface gráfico.

La capa esencial sobre la que reposa el resto del sistema operativo es normalmente llamada kernel. En general el kernel proporciona servicios de bajo nivel, como manejo de memoria, interacción con el hardware básico y seguridad. Sin el kernel el sistema se pararía.

7.3.2 SO

Un sistema operativo (SO) es un conjunto de programas o software destinado a permitir la comunicación del usuario con un ordenador y gestionar sus recursos de manera eficiente. Comienza a trabajar cuando se enciende el ordenador, y gestiona el hardware de la máquina desde los niveles más básicos, abstrayéndolo.

Otra definición posible y bastante aceptada define un sistema operativo como una capa compleja entre el hardware y el usuario, concebible también como una máquina virtual, que facilita al usuario o al programador las herramientas e interfaces adecuadas para realizar sus tareas informáticas, abstrayéndole de los complicados procesos necesarios para llevarlas a cabo. Por ejemplo, un usuario normal simplemente abre los ficheros grabados en un disco, sin preocuparse por la disposición de los bits en el medio físico, los tiempos de espera del motor del disco, la posición de un cabezal, el acceso de otros usuarios, etc.

Aunque es un tema propenso a la discusión, algunos expertos están de acuerdo en que un sistema operativo debe constar de, por lo menos, un conjunto de programas *similar* al siguiente:

Un compilador de algún lenguaje de programación, en Unix es de C.

Un enlazador.

Un ensamblador.

Un intérprete de comandos.
Una amplia biblioteca del lenguaje de la plataforma.
Un kernel o núcleo.

7.3.3 Compilador

Un compilador acepta programas escritos en un lenguaje de alto nivel y los traduce a otro lenguaje, generando un programa equivalente independiente, que puede ejecutarse tantas veces como se quiera. Este proceso de traducción se conoce como compilación.
GCC es un compilador rápido, muy flexible, y riguroso con el estándar de C ANSI. Como ejemplo de sus múltiples virtudes, diremos que gcc puede funcionar como *compilador cruzado* para un gran número de arquitecturas distintas. gcc no proporciona un entorno IDEs, es solo una herramienta más a utilizar en el proceso. gcc se encarga de realizar (o encargar el trabajo a otras utilidades, como veremos) el preprocesado del código, la compilación, y el enlazado. Dicho de otra manera, nosotros proporcionamos a gcc nuestro código fuente en C, y él nos devuelve un archivo binario compilado para nuestra arquitectura.
Como curiosidad, mencionar que en realidad gcc no genera código binario alguno, sino código ensamblado. La fase de ensamblado a código binario la realiza el ensamblador de GNU (*gas*), y el enlazado de los objetos resultantes, el enlazador de GNU. Este proceso es transparente para el usuario, ya que a no ser que se lo especifiquemos, gcc realiza el paso desde código en C a un binario ejecutable automáticamente.
GCC es un compilador integrado del proyecto GNU para C, C++, Objective C y Fortran; es capaz de recibir un programa fuente en cualquiera de estos lenguajes y generar un programa ejecutable binario en el lenguaje de la máquina donde ha de correr.
La sigla GCC significa "GNU Compiler Collection". Originalmente significaba "GNU C Compiler"; todavía se usa GCC para designar una compilación en C. G++ refiere a una compilación en C++.

7.3.3.1 Etapas de compilación

El proceso de compilación involucra cuatro etapas sucesivas: preprocesamiento, compilación, ensamblado y enlazado. Para pasar de un programa fuente escrito por un humano a un archivo ejecutable es necesario realizar estas cuatro etapas en forma sucesiva. Los comandos gcc y g++ son capaces de realizar todo el proceso de una sola vez.

7.3.3.1.1 Preprocesado

En esta etapa se interpretan las directivas al preprocesador. Entre otras cosas, las variables inicializadas con `#define` son sustituidas en el código por su valor en todos los lugares donde aparece su nombre.

7.3.3.1.2 Compilación

La compilación transforma el código C en el lenguaje ensamblador propio del procesador de nuestra máquina

7.3.3.1.3 Ensamblado

El ensamblado transforma el programa escrito en lenguaje ensamblador a código objeto, un archivo binario en lenguaje de máquina ejecutable por el procesador.

7.3.3.1.4 Enlazado

Las funciones de C/C++ incluidas en nuestro código, tal como printf() en el ejemplo, se encuentran ya compiladas y ensambladas en bibliotecas existentes en el sistema. Es preciso incorporar de algún modo el código binario de estas funciones a nuestro ejecutable. En esto consiste la etapa de enlace, donde se reúnen uno o más módulos en código objeto con el código existente en las bibliotecas.

7.3.3.1.5 Lenguaje de programación

Un lenguaje de programación es una técnica estándar de comunicación que permite expresar las instrucciones que han de ser ejecutadas en una

computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen un programa informático.

Aunque muchas veces se usa lenguaje de programación y lenguaje informático como si fuesen sinónimos, no tiene por qué ser así, ya que los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como, por ejemplo, el HTML.

Un lenguaje de programación permite a un programador especificar de *manera precisa*: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados y transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar *relativamente* próximo al lenguaje humano o natural, tal como sucede con el lenguaje Léxico.

Un programa escrito en un lenguaje de programación necesita pasar por un proceso de compilación, es decir, ser traducido al lenguaje de máquina, o ser interpretado para que pueda ser ejecutado por el ordenador.

7.3.4 Wi-Fi

Wi-Fi (o Wi-fi, WiFi, Wifi, wifi), acrónimo de *Wireless Fidelity*, es un conjunto de estándares para redes inalámbricas basado en las especificaciones IEEE 802.11.

Wi-Fi se creó para ser utilizada en redes locales inalámbricas, pero es frecuente que en la actualidad también se utilice para acceder a Internet.

7.3.5 IMU

Unidad de medida inercial. Conectado al puerto RS-232 de la Hercules nos proporciona los datos de medidas angulares, velocidades angulares etc.

7.4 Apéndice D: Comandos de Linux

En esta sección se muestran algunos de los comandos y trucos utilizados durante el proyecto, agrupados por temas

Fdisk /dev/hda opciones de formateo para la flash disk

7.4.1 Montar/desmontar dispositivos

mount vemos lo que tenemos montado

more /etc/fstab. Vemos lo que podemos montar

vi /etc/fstab editamos el archivo fstab con el editor vi.cuando estamos en el editor i nos mete en modo escritura, ecs nos saca y :wq salimos guardando

mount /dev/cdrom /cdrom nos monta el cdrom

umount cdrom lo desmonta

mount /dev/sda /floppy monta la disquetera

7.4.2 Manejo de archivos

Cp –a /bin /mnt/hda1=> copia todo lo que hay en bin al nuevo disco duro

Df-h=> espacio ocupado y libre en nuestros discos

Du=> lo que ocupa cada carpeta

Ls –F=> lista los archivos del directorio

Ln –s nombre directorio enlace=>crea un enlace simbolico

Kwrite xxx=> escribo en el archivo (entorno grafico)

jed lilo.conf=> modificamos el archivo de lilo con el editor jed

chroot /mnt/hda1 convierte ese punto del sistema de archivos a la raiz “/”

find <directorio> -name <nombre> =>busca el archivo por debajo del directorio

7.4.3 Comprimir/descomprimir archivos

Zip nom * -> comprime lo que hay en la carpeta y le da nombre nom. –r para subdirectorios

tar –xvzf nombre archivo.tar.gz-> descomprime

7.4.4 Módulos

apt –get install xxx =>instala xxx

apt –get remove xxx =>quita xxx

dpkg =>instala módulos en Debian

which modulo=>donde esta ese modulo

lsmod=>módulos cargados

insmod modulo=>carga modulo

rmmod modulo =>elimina modulo

7.4.5 Cargar TR

cd/usr/src/rtlinux

./scripts/insrtl carga RT

./scripts/rmrtl elimina RT

7.4.6 Pantalla

Dmesg ver los mensajes enviados por el núcleo

Dmesg –c borra los mensajes

Crtl+F5 cambio a modo grafico

7.4.7 Apagado-salida de procesos

ctrl+z =>deja un proceso en espera

ctrl+c =>interrrumpo el proceso

ps veo los procesos que tengo

kill –9 # =>mata al proceso #

Fg kill %1=> mata al proceso 1, no al de PID 1 (seria el apagado)

Shutdown –h =>apaga =halt

-r =>rebota ==reboot

-f =>rebota rapido (sin cfs)

exit=> cambiamos de usuario =logout

su=> pasamos a superusuario (root)

7.4.8 Uso de ayuda

Xxx –help=> veo el archivo de ayuda del comando

man xxx=> leo el archivo o el comando

7.4.9 Trucos

alias 'ls=ls –color –F'=> convierte un comando en otro

unalias=>

bashrc=>archivo de configuración de los alias

seleccionar texto y pulsar boton central=>copia

7.4.10 Kernel

úname –a=> versión del kernel

7.4.11 Permisos

chmod {a,u,g,o}{+,-}{r,w,x} [filenames]

chmod 777 #=>da permisos de lectura,escritura y ejecucion a todos los usuarios

ls –o lista los archivos con información sobre permisos

7.4.12 Makefile

make compila todos

make main1 fuerza a que compile solo main1 (por si hay varios)

make clean para borrar los ejecutables

7.4.13 Compilar

Gcc –o # #.c genera un ejecutable con nombre #

Ejecutar con ./#

Gcc –c #.c genera un punto .o por cada modulo de código fuente. No enlaza

7.5 Apéndice E: Recursos

7.5.1 Libros

Kurt Wall et al. Al descubierto. Programación en linux. Prentice may

Castaño Castaño Luis F. (1999)Lenguaje C: Herramienta de Ingeniería. S.P.E.S.I.S. Sevilla

Kernighan B. W & Ritchie D. M. (1991) El lenguaje de programación C. Prentice Hall.

Ogata K. (1998) Ingenieria de Control Moderna. Prentice Hall.

Bryson,JR . Arthur E. (1994) Control of Spacecraft and Aircraft. Princenton.

7.5.2 Artículos y paginas de Internet

Building Tiny Linux system with busy box
Linux journal

Pagina de ejemplos de C y Linux:
http://www.chuidiang.com

Guia Beej de programación de redes.
http://www.ecst.csuchico.edu/~beej/guide/net/

Three-Time Scale Singular Perturbation Control for a Radio-Control Helicopter on a Platform.
Sergio Esteban, Javier Aracil and Francisco Gordillo

Universidad de Sevilla, Sevilla, 41092, Spain

Nonlinear modelling and control of helicopters
Available online at www.sciencedirect.com

Getting started with RTLinux. FMS Labs, inc.

A Sample Control Application Employing RTLinux. Petter Wurmsdobler

RTLinux Installation Guide. Prof. Chang-Gun Lee

POSIX TIMERS implementation in RTLinux. Authors: J. Vidal, F. Gonzálvez, I. Ripoll.

Foro de linux:
www.espaciolinux.com

7.5.3 Manuales

Manual de la placa: Hercules EBX Model HRC 400-SA 128

Manual del driver de la placa: DSCUD V 5.7

Manual de la imu: 3DM-GX1 user manual

7.5.4 Software utilizado

Breve descripción del software utilizado y donde encontrarlo:

Linux Debian 3.0 (woody) sistema operativo utilizado
http://debian.org

DSCUD 5.7 driver de la placa Hercules
www.diamondsystems.com/

Partition magic. Programa que gestiona la partición de discos duros.