ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Carrera

ESTUDIO DISEÑO Y DESARROLLO DE UNA APLICACIÓN DE TIEMPO REAL Y DE UN SIMULADOR PARA SU COMPROBACIÓN: PÉNDULO INVERTIDO

Para acceder al Titulo de

INGENIERO TÉCNICO DE TELECOMUNICACIÓN

Autor: Rubén Miguélez García

Junio - 2002

ÍNDICE

1. INTRODUCCIÓN

1.1 SISTEMAS DE TIEMPO REAL

En sistemas de control, medición, comunicaciones, etc., es tan importante llevar a cabo la tarea para la cual fueron diseñados, como cumplirla en un tiempo preciso. En un sistema de tiempo real, el tiempo de respuesta está acotado a un valor perfectamente predecible.

Un sistema de tiempo real es una combinación de uno o varios computadores, dispositivos hardware de entrada / salida, y software de propósito especial, en el que hay una fuerte interacción con el entorno, que a su vez cambia con el tiempo. Es decir, el sistema controla o reacciona de forma simultánea a diferentes aspectos del entorno [14].

Como resultado de estas características se imponen requerimientos temporales sobre el software, que debe ser de naturaleza concurrente.

En las aplicaciones de tiempo real, el funcionamiento correcto no sólo depende de los resultados del cálculo, sino también del instante en el que se generan estos resultados.

Algunos requerimientos de tiempo real frecuentes son: realizar actividades periódicas que se deben completar antes de un plazo; responder a eventos aperiódicos solicitados a intervalos irregulares en los que en algunos casos, se debe completar el trabajo antes de un plazo o alcanzar un requerimiento de tiempo de respuesta promedio. Este es nuestro caso, ya que como veremos más adelante tenemos una aplicación que recoge y envía señales del exterior de forma periódica mientras realiza cálculos con las mismas, a la vez que se encarga de otras tareas periódicas, como son mostrar los datos por pantalla, y aperiódicas, como puede ser atender a las operaciones que sobre el control quiera hacer el usuario.

Si bien un sistema de tiempo real puede ser diseñado a la medida, dado que en general estará formado por varias tareas cooperativas, siempre es más conveniente utilizar como soporte un sistema operativo que ofrezca primitivas para manejar las restricciones temporales de cada tarea (contadores de tiempo, primitivas de sincronización, etc.), así como las facilidades típicas de cualquier sistema operativo (manejo de periféricos, de memoria, comunicaciones, etc.). Ellos son los sistemas operativos de tiempo real, como por ejemplo LynxOS, RT Linux, el estándar POSIX.13, MaRTE O.S., etc. A veces, no se emplea un sistema operativo completo sino sólo un núcleo básico de ejecución, como en el caso del lenguaje Ada cuando opera sobre máquina desnuda.

El estándar POSIX persigue la portabilidad de aplicaciones a nivel de código fuente. En el estándar POSIX general (llamado POSIX.1), se definen los servicios para aplicaciones de tiempo real, mientras que en el estándar POSIX.13 lo que se hace es definir subconjuntos de esos servicios, cada uno para un perfil de aplicaciones diferente. En particular, el más pequeño de los subconjuntos, el llamado perfil mínimo, es el que sigue el MaRTE O.S. y está pensado para sistemas empotrados.

1.2 MARTE O.S.

MaRTE O.S. es un sistema operativo de tiempo real para aplicaciones empotradas que sigue el subconjunto de sistema de tiempo real mínimo del estándar POSIX.13. La mayoría de su código ha sido escrito en Ada, con algunas partes de C y ensamblador.

Permite el desarrollo cruzado de aplicaciones en Ada y C usando los compiladores GNU Gnat y Gcc.

El entorno de desarrollo cruzado del MaRTE O.S. está formado por un PC que ejecuta Linux como "ordenador principal" o "host" y un PC como "objetivo" o "target". Ambos sistemas se conectan a través de una LAN Ethernet (para el arranque de la aplicación) y una línea serie (para depuración

remota). También es posible el arranque de la plataforma de ejecución mediante un disquete que contenga la aplicación a cargar junto con el sistema operativo.

El MaRTE O.S. se está desarrollando en el grupo de "Computadores y Tiempo Real" del departamento de "Electrónica y Computadores" de la Universidad de Cantabria. No es un producto acabado, pero por lo pronto es apto para propósitos educativos y/o experimentales [25].

El MaRTE O.S. está disponible bajo la licencia GNU "General Public License", lo cual implica que el software se puede ejecutar, copiar, distribuir, estudiar, cambiar y mejorar libremente.

1.3 OBJETIVOS GENERALES

En este proyecto, el objetivo principal es la demostración del uso del sistema operativo de tiempo real MaRTE O.S. para aplicaciones de control. Para conseguir este objetivo se planteó el realizar una aplicación de tiempo real que conjuntase el uso de la programación concurrente en Ada, con el desarrollo de drivers en C y con la implementación de algoritmos complejos de control, todo ello montado sobre un sistema operativo de tiempo real, MaRTE O.S., mencionado en el apartado anterior y sobre el que había pocas experiencias.

Después de pensar en varias opciones se optó por el control de un péndulo invertido, ya que había en la literatura y en Internet suficientes ejemplos y diseños de los que nos podríamos ayudar adaptándolos a nuestras necesidades.

Además, el péndulo invertido representa un problema clásico de control complejo, ya que es un sistema inherentemente inestable en el que los métodos tradicionales de control (p.e. PID) no funcionan correctamente [24].

Paralelamente y ante la ausencia de un sistema real se desarrollo también un simulador del sistema físico del péndulo invertido, el cual nos permitiría comprobar la validez de la aplicación de control.

Se pretende dejar preparado además el camino para una futura implementación real del péndulo invertido mediante:
- Un programa en Matlab que automatiza el cálculo de las ecuaciones ante posibles modificaciones de los parámetros del sistema físico y del sistema de control.
- Un entorno de test compuesto por las aplicaciones y los drivers escritos para Linux, ya que representa un medio más cómodo para hacer ensayos.
- Driver de la tarjeta escrito en Linux, ya que en un futuro los drivers de MaRTE O.S. tendrán el mismo formato.

1.4 PÉNDULO INVERTIDO

Un péndulo invertido es un dispositivo físico que consiste en una barra cilíndrica que oscila libremente alrededor un pivote fijo. El pivote está montado sobre una pieza móvil que se desplaza en dirección horizontal. La barra naturalmente tiende a caerse desde la posición vertical, ya que es una posición de equilibrio inestable.

El objetivo del sistema de control es estabilizar el péndulo (barra) en la posición vertical y situar además el móvil en la posición deseada. Esto es posible ejerciendo una fuerza sobre la pieza móvil (carro) que tienda a estabilizar el péndulo a la vez que posiciona el carro en el lugar adecuado. La fuerza correcta tiene que ser calculada midiendo los valores en cada instante de la posición horizontal del carro y del ángulo de péndulo. Esta fuerza constituye el parámetro de control.

En nuestro caso el carro es conducido por un motor, que ejerce sobre él una fuerza variable. Y las medidas de la posición del carro y ángulo del péndulo se realizan a través de dos potenciómetros.

El sistema completo puede ser modelado como un sistema lineal en torno a la posición de equilibrio si todos los parámetros son conocidos (masas, longitudes, etc.), para encontrar un controlador que permita estabilizar el sistema.

Es de resaltar la dificultad que entraña el control, ya que hay que hay que controlar dos variables, la posición del carro y el ángulo del péndulo y que no puede hacerse de forma independiente. Por ejemplo: si queremos desplazar la base hacia un lado, es necesario primero moverla en el sentido contrario (lo que provocará un aumento del error) para posteriormente desplazarla en el sentido deseado.

1.5 ¿PARA QUÉ PUEDE SER USADO EL PÉNDULO INVERTIDO?

El péndulo invertido es un tradicional ejemplo de un sistema controlado. Así, puede ser usado en simulaciones y experimentos para mostrar la respuesta de diferentes controladores (p.e., controladores PID, controladores de variables de estado, controladores de lógica difusa "fuzzy controllers"...).

El péndulo Invertido puede ser usado también como banco de pruebas de sistemas de tiempo real, para comprobar la validez tanto del algoritmo de control como del sistema operativo que lo implemente.

El algoritmo de control está formado por un conjunto de tareas, que son activadas periódicamente por el sistema operativo, y que realizan operaciones diferentes.

En la realidad, se pueden encontrar problemas que se modelan como un péndulo invertido, como pueden ser el control sísmico de estructuras de edificación [27], diversos métodos de regulación fiscal [28] o el control de posicionamiento de antenas de radar situadas sobre plataformas móviles.

1.6 DRIVERS

Para la adquisición de señales del exterior se eligió una tarjeta A/D D/A modelo AX5411 de AXIOM. Esta tarjeta disponía de 16 canales analógicos de entrada y 2 de salida con unas especificaciones temporales suficientes para nuestra aplicación, ya que usando dos canales de muestreo, la frecuencia máxima de cada uno de ellos está en torno a 30 Khz, mientras que en nuestra aplicación, son suficientes frecuencias por debajo de 1Khz. Aunque la tarjeta incorporaba un driver escrito en Pascal y C montado sobre Basic, se decidió hacer uno propio ya que este driver no nos era útil, no se conocía el código fuente del mismo y además, lo que se pretendía era desarrollar uno propio con el formato de los drivers en Linux, ya que MaRTE O.S. en un futuro tendría también ese formato.

Un driver para un dispositivo es una interfaz entre el sistema operativo y el hardware del dispositivo. Los drivers forman parte del núcleo (parte mas interna del software del sistema operativo) y tienen acceso restringido a las estructuras del sistema operativo.

El objetivo de un driver debe ser flexibilizar el uso de los dispositivos, proporcionando un mecanismo de uso.

Actualmente, MaRTE O.S. no tiene un formato definido para los drivers de dispositivos, pero hay un proyecto en marcha que está trabajando para que se puedan implementar con un formato similar al de los drivers de Linux [25]. Así, el acceso a los dispositivos se realizaría a través de "ficheros de dispositivo" con sus operaciones estándar (open, close, read, write).

Por esta razón, nuestro driver se realizó primero en Linux y luego se usó en MaRTE O.S. como una librería que implementaba las funciones de acceso y configuración de la tarjeta conversora.

1.7 PLANIFICACIÓN DE SISTEMAS DE TIEMPO REAL

En un sistema monoprocesador el objetivo de ejecutar en estricta concurrencia todas las tareas no se puede cumplir. La única posibilidad está entonces en seleccionar el orden adecuado de ejecución para que, jugando con los tiempos libres disponibles, se puedan cumplir los plazos de todas ellas. Al algoritmo encargado de hacer la selección se le llama planificador.

La planificación de sistemas de tiempo real consiste pues, en la definición de las reglas de uso de cada uno de los recursos disponibles. Un sistema de tiempo real se considera planificable si, en función de la política de planificación elegida, es capaz de satisfacer todos los requisitos temporales impuestos [20].

Existen diversos tipos de planificación dependiendo de cuando y como se hagan:
- Planificación estática u off-line: Esta planificación, para tareas periódicas, se realiza en tiempo de compilación, esto es, una vez conocido el sistema y antes de su ejecución. A cada tarea del sistema se le asignan rodajas temporales durante las cuales puede ejecutar, según una tabla (plan estático)
- Planificación dinámica u on-line: En esta política de planificación se sigue el concepto de tarea como "thread" de ejecución (proceso que comparte con otros, recursos y localizaciones en memoria). A cada tarea se le asigna una prioridad y, en función de ella, se resuelven los conflictos de utilización del procesador. Cuando hay varias tareas que quieren ejecutar, el planificador elige de entre todas ellas aquella con prioridad más alta y le asigna el uso del procesador.

Si la prioridad de una tarea no cambia una vez asignada, hablaremos de prioridades fijas. Por el contrario, si la prioridad puede variar en tiempo de ejecución, en función del estado de operación del sistema, hablaremos de prioridades dinámicas. La asignación dinámica es más eficiente que la asignación estática, desde el punto de vista de que consigue mayor utilización de los recursos. Sin embargo, la implementación del planificador es mucho más sencilla cuando se utiliza la asignación estática y es la que incorporan la mayoría de los planificadores comerciales existentes.

Dentro de las políticas de planificación por prioridades se puede elegir entre un planificador expulsor o no expulsor. En un planificador no expulsor, cuando una tarea toma el control del procesador no detiene su ejecución hasta que ésta ha finalizado. En cambio, en un planificador expulsor una tarea cede el uso del procesador (es expulsada) si se activa una tarea de prioridad superior.

Entre los planificadores estáticos expulsores de prioridad fija, podemos destacar:
- Rate Monotonic (RM): Se le asigna la mayor prioridad a la tarea más frecuente. Se utiliza cuando en cada tarea su plazo coincide con su periodo.
- Deadline Monotonic (DM): Se le asigna la mayor prioridad a la tarea con menor plazo. Se utiliza cuando en cada tarea su plazo es menor que su periodo.

Y entre los dinámicos expulsores:
- Earliest Deadline First (EDF): En cada instante de tiempo se le asigna la mayor prioridad a la tarea cuyo plazo se vaya a vencer antes.
- Least Laxity First (LLF): En cada instante de tiempo se le asigna la mayor prioridad a la tarea que tenga menos laxitud. La laxitud se define como el plazo menos el tiempo de cómputo que falta.

Como en todo sistema de tiempo real, se realizará el análisis de planificabilidad que posibilitará saber de antemano si el sistema será o no planificable, es decir, si se cumple el test de planificabilidad, se garantiza que el conjunto de tareas no violará ninguno de sus requerimientos temporales.

En este cálculo se tendrá en cuenta también el efecto debido a la inversión de prioridad que se produce cuando una tarea de prioridad alta tiene que esperar a que se libere un recurso o una sección crítica que está siendo utilizado por otra tarea de prioridad menor. Aunque no se puede evitar la inversión de prioridades cuando se comparten recursos, sí se puede lograr acotar su duración usando una de las siguientes técnicas:
- Deshabilitando la expulsión de tareas selectivamente, concretamente al entrar en zonas críticas.

- Usando el protocolo de herencia de prioridades PIP: la tarea que está usando el recurso adquiere la prioridad de la más alta de las que quieren usarlo en ese momento.
- Usando el protocolo de techo de prioridades PCP: cada sección crítica se ejecuta con la prioridad más alta de todas las tareas que la van a llamar.

1.8 SIMULADOR DEL PÉNDULO INVERTIDO

Un simulador es un programa que imita las acciones del sistema que se quiere controlar. Usando un simulador, se pueden reproducir comportamientos normales o anormales del sistema simulado. Incluso cuando se ha terminado el sistema final, ciertos estados de error solamente se pueden experimentar con seguridad con ayuda de un simulador.

Los simuladores pueden reproducir exactamente la secuencia de eventos esperada en el sistema verdadero. Además, permiten realizar experimentos de una manera que generalmente es imposible en el modo de operación normal del sistema real [15].

La construcción de un simulador es facilitada por el proceso de modelado del sistema durante la construcción del sistema de control.

Aunque los simuladores no tienen tantos requerimientos como los sistemas de control, la construcción de un simulador no es una tarea trivial y habitualmente suelen convertirse en sistemas costosos.

Para probar la aplicación de control desarrollada se realizo otra aplicación, llamada *Sistema,* con idénticas características (SO de tiempo real, tarjeta conversora A/D D/A, driver y estructura de programas en Ada) que simulase al sistema modelado, de forma que respondiese ante estímulos de igual forma que lo haría el péndulo invertido. Así, de esta manera podríamos por un lado ver la respuesta del sistema ante distintos tipos de control y por otro, ver cómo cambia la respuesta de un tipo de control al modificar características del sistema físico (masa del carro, longitud del péndulo, etc.)

El motivo que nos llevó a realizar un simulador fue la demostración funcional del sistema de control elaborado, ya que la construcción de un sistema real era demasiado costosa y difícil. Este simulador nos permitiría así realizar una verificación del sistema de control más allá de una simple simulación, ya que su funcionamiento es totalmente independiente y de tiempo real.

Otra opción más sencilla, pero menos adecuada, hubiera sido el simular el sistema físico en el mismo computador de control, sin usar hardware de entrada / salida. Esto presenta dos inconvenientes: por un lado, con este método no se comprueba el correcto funcionamiento del hardware de entrada / salida mediante su control a través del driver diseñado, y por otro lado, no se verifica totalmente el adecuado funcionamiento del software (y en particular del comportamiento de tiempo real), ya que al incluir el simulador, el software no sería el mismo que habría en un computador que sólo realiza la tarea de control.

1.9 OBJETIVOS CONCRETOS

A continuación, se mostrarán a modo de resumen de lo visto en la introducción los objetivos concretos de este trabajo:

- Estudiar el funcionamiento de la tarjeta conversora y elaborar un driver en Linux que sirviese para satisfacer todas las necesidades de la posterior aplicación de control.
- Modelar el sistema del péndulo invertido y del motor y encontrar un algoritmo de control en Matlab.
- Trasladar ese algoritmo a código en Ada.
- Elaborar la aplicación de control en Ada sobre Linux que recogiese el algoritmo de control.
- Elaborar la aplicación que simula al péndulo invertido ‘Sistema’, también con Ada sobre Linux.
- Trasladar las dos aplicaciones y el driver a MaRTE O.S..

- Medida de los tiempos de ejecución de las tareas (tanto de Control como Sistema) para realizar el test de planificabilidad.
- Realizar una serie de pruebas con las aplicaciones Control y Sistema a diferentes frecuencias de funcionamiento para observar tanto su validez como su rendimiento.

1.10 ESTRUCTURA DE ESTA MEMORIA

Esta memoria está estructurada de la siguiente manera: en el capítulo 2 se describirá la estructura global de las aplicaciones y del driver desarrollado junto con sus interfaces y modos de uso. En el capítulo 3, se modelará matemáticamente el sistema físico para posteriormente realizar el diseño del algoritmo de control del mismo, mediante el método de variables de estado con la ayuda de Matlab. En el capítulo 4, se describirá en detalle cada uno de los módulos que componen las aplicaciones Control y Sistema. Después, en el capítulo 5, se hará lo mismo con el driver de la tarjeta conversora, tanto en Linux como en MaRTE O.S. En el capítulo 6, se realizará el análisis de planificabilidad de las dos aplicaciones de tiempo real, Control y Sistema. Finalmente, en el capítulo 7, se expondrán los resultados de una serie de pruebas realizadas uniendo el controlador y el simulador, para terminar en el capítulo 8 con unas conclusiones finales sobre el trabajo realizado y el posible trabajo futuro.

2. DISEÑO ARQUITECTÓNICO

2.1 INTRODUCCIÓN

El software consta de dos 2 módulos por equipo. Por una parte está el driver de la tarjeta A/D D/A escrito en C que tiene las funciones típicas de un fichero de dispositivo, ya que proviene de uno escrito para Linux que posteriormente se trasladó a MaRTE O.S. Por otra parte está el programa que lo utiliza, que está escrito en Ada y realiza las funciones de 'control' o 'sistema' dependiendo de en qué PC se encuentre. El módulo de control lee la posición y el ángulo del sistema en forma de voltajes a través de su driver correspondiente y responde emitiendo una señal de control en forma de voltaje también. Esta señal es captada por el sistema y atendiendo a sus ecuaciones se ocasiona un cambio en sus variables (posición y ángulo) que transmite por los canales de salida.

Antes de ponerse a desarrollar el software, había que determinar cuáles eran las tareas a realizar y en qué lenguajes se iban a implementar. Por una parte estaba el driver de la tarjeta y por otra la aplicación de control propiamente. Para la aplicación de simulación no fue necesario un estudio de su implementación, ya que su estructura era similar a la de control.

2.2 DRIVER DE LA TARJETA CONVERSORA

El driver se desarrolló en C bajo Linux, ya que en este sistema operativo es el único lenguaje soportado para escribir módulos que se puedan integrar en su núcleo. Además, era uno de los dos lenguajes con los que se podían hacer desarrollos en MaRTE O.S. [25].

Como el funcionamiento del driver es similar al de un "fichero de dispositivo" en Linux, posee las funciones básicas que abstraen el funcionamiento del dispositivo de forma que su uso sea idéntico al de un fichero [10]. Dichas funciones son las siguientes:

- init_module : Instala el módulo en el sistema operativo e inicializa los registros de la tarjeta para la posterior ejecución del resto de las funciones.
- cleanup_module : Desinstala el módulo, extrayéndolo del sistema operativo y eliminando todo rastro del mismo.
- open_tarjeta y release_tarjeta : Arranca / paraliza el ciclo de muestreo abriendo o cerrando el fichero de dispositivo de la tarjeta. Cada canal de muestreo tiene asociado un fichero distinto.
- read_tarjeta y write_tarjeta : Lee / escribe en los canales de la tarjeta mediante la lectura / escritura en el fichero de dispositivo. El dato que se lee, es siempre el último muestreado y el que se escribe es trasladado inmediatamente al exterior en forma de voltaje.

2.3 INTERFAZ DEL DRIVER

La especificación de las funciones del driver en Linux es la siguiente:

```
//--------- READ_TARJETA ----------
static ssize_t read_tarjeta(struct file * file,char * buf, size_t count,
loff_t *offp)

//--------- WRITE_TARJETA ---------
static ssize_t write_tarjeta(struct file * file,const char * buf, size_t
count, loff_t *offp)

//-------- OPEN_TARJETA -------------
static int open_tarjeta(struct inode *inode, struct file * file)
```

```
//-------- RELEASE_TARJETA ----------
static int release_tarjeta(struct inode *inode, struct file * file)

//--------FUNCION_MANEJADORA--------
void funcion_manejadora (int irq,void *dev_id,struct pt_regs *regs)

//-------- INIT_MODULE -------------
int init_module(void)

//--------- CLEANUP_MODULE ----
void cleanup_module(void)
```

Posteriormente, a la hora de trasladar toda la aplicación a MaRTE O.S., y como en este sistema operativo todavía no estaba implementada la funcionalidad de los ficheros de dispositivo, se tuvo que hacer un parche al driver para que funcionase en MaRTE O.S. Este parche consistía en transformar a simples funciones C, las funciones que implementaban las acciones de instalación y manipulación del fichero de dispositivo en Linux. Posteriormente, se construyó un paquete Ada que se encargaría de importar a Ada las funciones C. Su especificación es la siguiente:

```
package      Driver_Marte is
      ---------------------------------
      type Short is mod 4096;
      for  Short'Size use 16;
      ---------------------------------
      function Init_Module (D_Cont1,D_Cont2 : in Integer)  return Integer ;
      pragma Import (C, Init_Module,"init_module");
      ---------------------------------
      procedure Cleanup_Module;
      pragma Import (C, Cleanup_Module,"cleanup_module");
      ---------------------------------
      procedure Open_Tarjeta;
      pragma Import (C,Open_Tarjeta,"open_tarjeta");
      ---------------------------------
      procedure Release_Tarjeta;
      pragma Import (C,Release_Tarjeta,"release_tarjeta");
      ---------------------------------
      function Read_Short (Minor :in Integer) return Short;
      pragma Import (C,Read_Short,"read_short" );
      ---------------------------------
      procedure Write_Short(Minor :in Integer;Dato : in Short);
      pragma Import (C,Write_Short,"write_short");
      ---------------------------------
end Driver_Marte;
```

Se observa una característica nueva en la función Init_module, y es que ésta posee un parámetro adicional, la frecuencia de muestreo de la tarjeta. Se ha hecho esto por comodidad, ya que como veremos más tarde, se harán numerosos experimentos variando la frecuencia de muestreo y el meter la frecuencia en los parámetros de la función Init_module nos evita tener que hacer una compilación del driver por cada frecuencia de muestreo.

2.4 APLICACIÓN

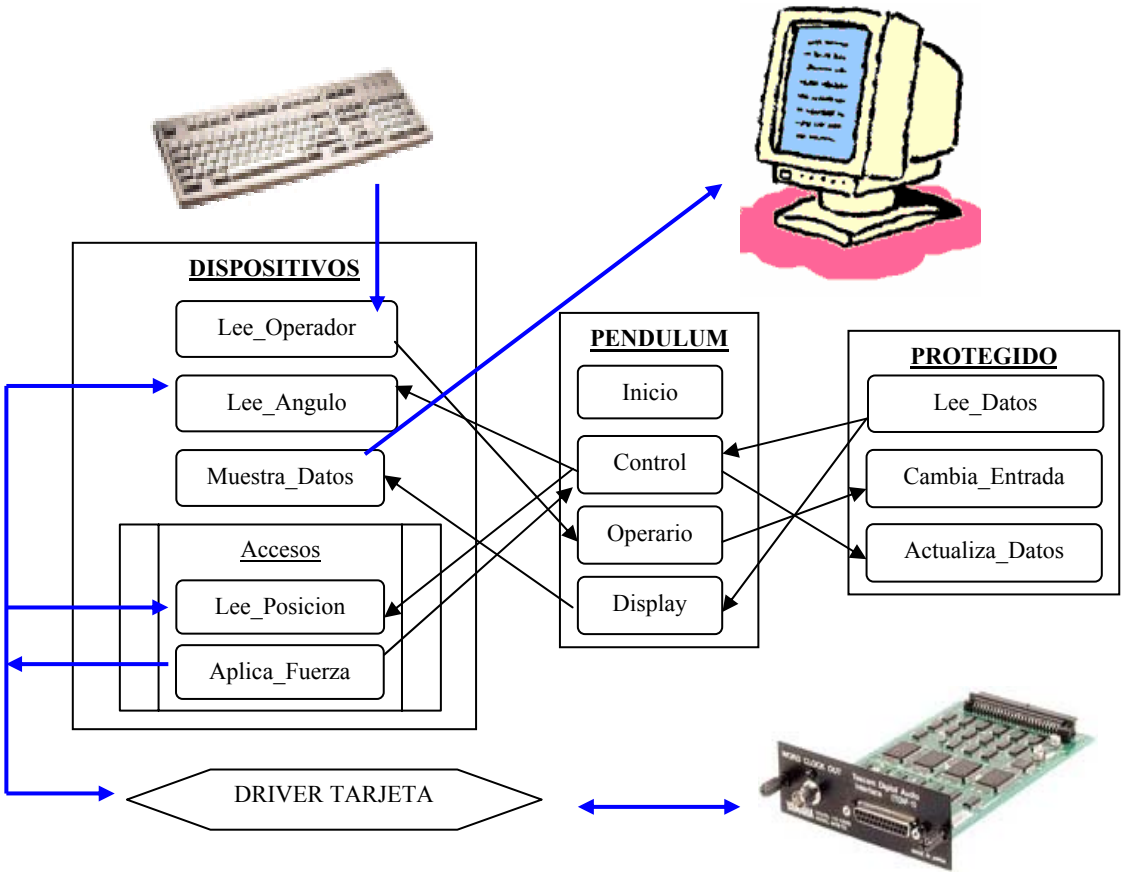
La aplicación de control fue escrita en Ada, ya que este lenguaje posee las funcionalidades básicas y necesarias para realizar programas de tiempo real. Soporta perfectamente la programación concurrente, se pueden poner prioridades a las tareas y facilita métodos para el uso de objetos protegidos que proporcionen accesos en exclusividad a estructuras de datos. Otra opción era escribirlo en C, pero dado que el lenguaje Ada ofrece programas más legibles, estrictos, con una fiabilidad

mayor y que la concurrencia era implementable desde el lenguaje de forma sencilla, se optó por este lenguaje [1] [7].

Tras un estudio de las tareas y funciones que debía realizar la aplicación, se optó por encapsular sus funciones y tareas en paquetes de forma que su uso fuera mucho más genérico e independiente. Así se crearon tres paquetes y un programa principal llamado *Main*. El programa principal se encargaría únicamente de llamar al paquete que contenía las tareas que iniciarían su ejecución en concurrencia. Las tareas utilizan un objeto protegido (contenido en el paquete protegido) en el que se almacenan los datos y cuyo acceso y manipulación se realiza a través de unos procedimientos establecidos que garanticen la consistencia de los datos en él almacenados. Además, para facilitar la interacción con el exterior se utilizan unas funciones y procedimientos que abstraen el funcionamiento y manejo tanto de la pantalla y el teclado como del driver (encerrados en el paquete Dispositivos).

Los paquetes y su contenido son:
- Dispositivos: Contiene las funciones para el manejo del driver, del teclado y de la pantalla.
- Protegido: Este es el objeto protegido que se encarga de contener los principales datos del programa y de proporcionar funciones y procedimientos para su lectura y manipulación.
- Pendulum: Contiene una tarea de inicialización llamada Inicio que tras ejecutarse llama a otras tres más, que se ejecutarán de modo indefinido:
  - Operario: Espera la entrada por teclado y actúa en consecuencia. Es aperiódica, ya que realiza su actividad cada vez que se manipula el teclado.
  - Control: Realiza las lecturas de los datos del exterior, calcula los nuevos valores según sus ecuaciones y traslada el resultado al exterior. Se activa periódicamente a una frecuencia de 100 Hz con cada muestreo de la tarjeta conversora.
  - Display: Se encarga de mostrar los datos por pantalla periódicamente cada 0.5 seg.

Un gráfico descriptivo de los paquetes y sus interrelaciones para la aplicación Control es el siguiente:



2.5 INTERFACES DE LOS PAQUETES

Inmediatamente, describiremos la funcionalidad de cada paquete y su especificación para proporcionar la información necesaria y suficiente para su uso. Como la aplicación *Sistema* es muy similar a la de *Control* en su estructura, sólo se expondrán las diferencias más notables.

2.5.1 PAQUETE DISPOSITIVOS

2.5.1.1 Aplicación Control

Posee funciones para la lectura del teclado, escritura en pantalla y lectura / escritura en la tarjeta a través del driver. A continuación, se detallarán sus funciones agrupadas por dispositivo.

Para la pantalla:
- Muestra_Datos: Muestra en pantalla los tres datos que se le pasan con un formato que se verá más adelante y un pequeño gráfico en modo texto, ilustrativo de los valores del sistema.

Para el teclado:
- Lee_Operador: Se queda esperando en el teclado la recepción de algún dato, el cual devuelve como resultado.

Para la tarjeta conversora:
- Lee_Posicion: Devuelve la posición del carro medida en metros, mediante la lectura del canal de entrada 0 de la tarjeta. Los valores de posición posibles están entre [-0.5, +0.5] m.
- Lee_Angulo: Devuelve el ángulo del péndulo en radianes, mediante la lectura del canal de entrada 1 de la tarjeta. Los valores de ángulo posibles oscilan entre [-40, +40] grados (cero grados corresponde a la vertical), es decir [-0.69, +0.69] rad.
- Aplica_Fuerza: Traslada al exterior en forma de señal eléctrica a través del canal de salida 0 de la tarjeta, la señal de control aplicada. Hay que decir que no es exactamente fuerza, sino que esa señal se aplicara al motor, el cual sí producirá una fuerza que se aplicará sobre el sistema. Los límites de la fuerza en Newtons se introducen por teclado en el proceso de inicialización.

Las funciones `Lee_Posicion` y `Aplica_Fuerza` están dentro de un objeto protegido llamado `Accesos` ya que son dos operaciones (una de lectura y otra de escritura) que operan sobre el mismo fichero de dispositivo y, por lo tanto, requieren que se ejecuten en exclusión mutua para que sean operaciones seguras.

Su especificación es la siguiente:

```
package Dispositivos is
   ---------------------------------------
   subtype Posicion_T is Float;
   subtype Angulo_T   is Float;
   subtype Fuerza_T   is Float;
   --------------------------
   procedure Muestra_Datos (Pos,Ang,Fuerza : in Float);
   function  Lee_Angulo   return Angulo_T;
   function  Lee_Operador return Posicion_T;
   -------------------------
   protected Accesos is
      function  Lee_Posicion   return Posicion_T;
      procedure Aplica_Fuerza (Fuerza: in Fuerza_T);
   end Accesos;
   ------------------------------------
end Dispositivos;
```

2.5.1.2 Aplicación Sistema

Respecto a la aplicación de control, tan sólo cambian las funciones de lectura / escritura en el driver. *Lee_Angulo* se cambia por *Escribe_Angulo*, *Lee_Posicion* por *Escribe_Posicion* y *Aplica_Fuerza* por *Lee_Fuerza*.

Su utilización es la siguiente:
- Escribe_Posicion: Traslada al exterior a través de una señal eléctrica mediante la escritura en el canal de salida 0 de la tarjeta la posición del carro en metros que se le pasa. Los valores de posición posibles están entre [-0.5, +0.5] m.
- Escribe_Angulo: Traslada al exterior el ángulo del péndulo en radianes, mediante la escritura en el canal de salida 1 de la tarjeta. Los valores de ángulo posibles oscilan entre [-40, +40] grados (cero grados corresponde a la vertical), es decir [-0.69, +0.69] rad.
- Lee_Fuerza: Lee por el canal de entrada 0 de la tarjeta, la señal de control (fuerza en Newtons) que se está aplicando sobre el sistema. El rango de fuerzas de entrada se especifica en el proceso de inicialización y debe estar acorde con el establecido en la aplicación Control.

Su especificación es la siguiente:

```
package Dispositivos is
   ---------------------------------------
   subtype Posicion_T is Float;
   subtype Angulo_T   is Float;
   subtype Fuerza_T   is Float;
   --------------------------
   procedure Muestra_Datos   (Pos,Ang,Fuerza : in Float);
   procedure Escribe_Angulo  (Angulo: in Angulo_T);
   function  Lee_Operador    return Float;
   --------------------------
   protected Accesos is
      procedure Escribe_Posicion (Posicion: in Posicion_T);
      function  Lee_Fuerza  return Fuerza_T ;
   end Accesos;
   ------------------------------------
end Dispositivos;
```

2.5.2 PAQUETE PROTEGIDO

2.5.2.1 Aplicación Control

Como hemos dicho, este paquete contiene una estructura de datos llamada Datos_T que alberga los principales datos del sistema. Además proporciona funciones y procedimientos para un uso de los mismos en exclusión mutua.

Los datos que se almacenan en él son:

Xs: Vector que contiene las variables de estado estimadas. Estas son: posición del carro (m), velocidad del carro (m/s), posición angular del péndulo (rad), y velocidad angular del péndulo (rad/s).

Y: Vector que contiene la salida real del sistema. Posición del carro (m) y ángulo del péndulo (rad).

Ys: Vector que contiene la salida estimada del sistema. Similar a Y.

U : Señal de salida del controlador. Expresa la fuerza en Newtons que ha de ser aplicada por el motor al carro.

R : Señal de referencia del controlador. Indica la posición deseada del carro. Su valor inicial es 0.

Las funciones y procedimientos realizan los siguientes cometidos:

- Lee_Datos: Devuelve la estructura de datos Datos_T mediante la cual se puede acceder a todos los elementos.
- Cambia_Entrada: Cambia el valor de la señal de referencia del sistema por la introducida en el argumento.
- Actualiza_Datos: Sustituye los datos X, Y, Ys y U de la estructura por los que se suministran en los argumentos.

Su especificación es la siguiente:

```
package Protegido  is
   ----------------------
   type Datos_T is record
      --valores de inicializacion--
      Xs:Vector(1..4):=(others=>0.0);
      Y,Ys:Vector(1..2):=(others=>0.0);
      U : Float :=0.0;
      R : Float :=0.0;
   end record ;
   -----------------------
   protected Sistema is
      pragma Priority (10);

      function  Lee_Datos       return Datos_T ;
      procedure Cambia_Entrada  (Entrada    : in Float);
      procedure Actualiza_Datos (IXs,IY,IYs : in Vector; Iu :in Float);
   private
      Datos:Datos_T;
   end Sistema;
   -----------------------
end Protegido;
```

2.5.2.2 Aplicación Sistema

En la aplicación Sistema los datos almacenados son los correspondientes únicamente al sistema real:

X: Vector que contiene las variables de estado del sistema. Estas son: posición del carro (m), velocidad del carro (m/s), posición angular del péndulo (rad), y velocidad angular del péndulo (rad/s).

Y: Vector que contiene la salida real del sistema. Posición del carro (m) y ángulo del péndulo (rad).

U : Señal de control recibida. Indica la fuerza en Newtons que se está aplicando al carro.

Las funciones y procedimientos realizan las mismas operaciones que en la aplicación control, con la excepción de `Cambia_Entrada` que se cambia por `Mueve_Pendulo`, la cual sirve para provocar perturbaciones sobre el péndulo, es decir, cambios en la inclinación del péndulo, provocados por agentes externos al propio sistema físico y al controlador. `Mueve_Pendulo` modifica el ángulo del péndulo sumándole el valor (positivo o negativo) en radianes pasado al procedimiento.

```
package Protegido  is
   ----------------------
   type Datos_T is record
      -- Valores de inicializacion --
      X:Vector(1..4):=(others=>0.0);
      Y:Vector(1..2):=(others=>0.0);
      U : Float :=0.0;
   end record ;
   -----------------------
```

```
   protected Sistema is
      pragma Priority (12);

      function  Lee_Datos       return Datos_T ;
      procedure Mueve_Pendulo   (Entrada : in Float);
      procedure Actualiza_Datos (IX,IY : in Vector;Iu :in Float);

   private
      Datos:Datos_T;
   end Sistema;
   -----------------------
end Protegido;
```

2.5.3 PAQUETE PENDULUM

2.5.3.1 Aplicación Control

Este paquete contiene el "corazón" de la aplicación, ya que los otros dos paquetes sólo son herramientas que serán utilizadas por las tareas que contiene éste.

Al ser llamado el paquete por el programa *Main,* se activará una tarea llamada Inicio que se encargará de inicializar el sistema y posteriormente, activará a las otras tres y terminará.

Cada tarea posee una prioridad y un periodo de funcionamiento (excepto Control que funciona a la frecuencia de muestreo de la tarjeta conversora).

Las tareas Control, Operario y Display se han declarado como tipos para que se pudiesen lanzar después del proceso de inicialización realizado por la tarea Inicio.

Su especificación es la siguiente:

```
package Pendulum  is
   P_I:Integer:=10;

   P_C:Integer:=8;

   T_Operario : Duration:=0.400;
   P_O:Integer:=6;

   T_Display  : Duration:=0.500 ;
   P_D:Integer:=4;

   -----------------------
   task   Inicio is
      pragma Priority(P_I);
   end  Inicio;

   task type  Control is
      pragma Priority(P_C);
   end Control;

   task type  Operario is
      pragma Priority(P_O);
   end Operario;

   task type  Display is
      pragma Priority(P_D);
   end Display;
   -----------------------
```

```
end Pendulum;
```

2.5.3.2 Aplicación Sistema

Lo explicado anteriormente para Control, vale también para Sistema. Las tres tareas operan concurrentemente, cada una con una prioridad. La más urgente es la tarea Control que se encarga de leer la señal de control recibida y modificar la salida del sistema después de aplicar dicha señal a las ecuaciones que rigen el péndulo invertido.

```
----------------------------------------
package Pendulum  is
   ----------------------
   P_I:Integer:=10;

   P_C:Integer:=8;

   T_Operario : Duration:=0.400;
   P_O:Integer:=6;

   T_Display  : Duration:=0.500;
   P_D:Integer:=4;
   ----------------------
   task Inicio is
      pragma Priority(P_I);
   end  Inicio;

   task type Control is
      pragma Priority(P_C);
   end Control;

   task type  Operario is
      pragma Priority(P_O);
   end Operario;

   task type  Display is
      pragma Priority(P_D);
   end Display;
   ----------------------
end Pendulum;
```

2.6 INTERFAZ DE USUARIO

En nuestra aplicación las interacciones con el usuario se llevan a cabo a través del teclado y la pantalla.

Por pantalla el usuario operador puede observar la evolución del sistema en todo momento, viendo tanto el valor numérico de los principales datos que se manejan (posición en metros, ángulo de inclinación en radianes y fuerza de control que se aplica en Newtons) como un sencillo gráfico en modo texto que refleja de forma más cómoda y visual la evolución del sistema. En la aplicación de control hay un dato más a mostrar por pantalla: la media de los últimos 10 valores de posición del carro. Esto permitiría apreciar más fácilmente la posición del sistema, ya que éste nunca llega a la estabilidad absoluta, al hallarse en equilibrio inestable.

A continuación, se describe el procedimiento de ejecución de las aplicaciones 'control' y 'sistema':

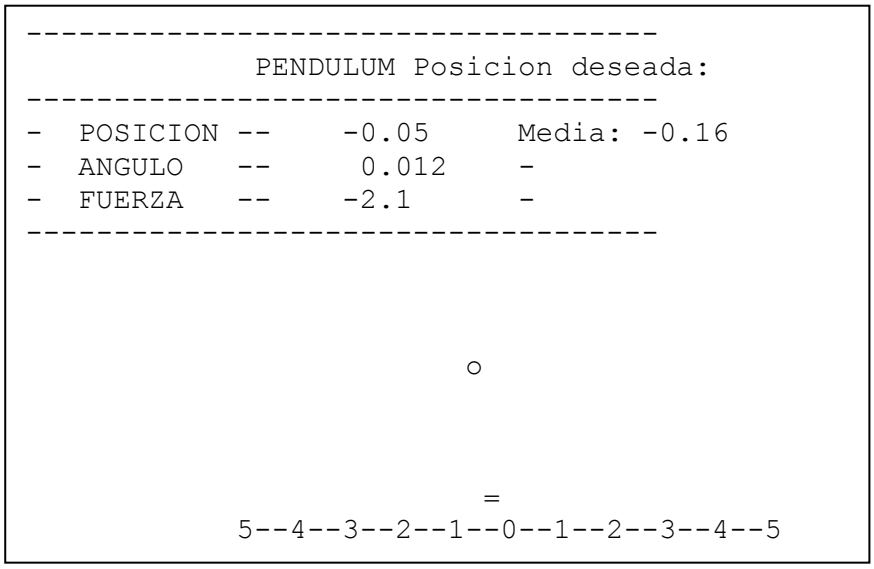
Por teclado el usuario operador debe realizar una primera tarea de inicialización de las aplicaciones para después poder proceder a su uso. Para ello, después de que se haya cargado cada aplicación en su correspondiente PC, ya sea por red o por disquete, es necesario introducir un valor numérico distinto

de cero para situar en posición de equilibrio al sistema y de no-acción al control, para posteriormente introducir un cero en las dos aplicaciones a la vez (no estrictamente, pero sí, al menos, antes al control que al sistema) para que empiecen a ejecutar de forma conjunta.

Seguidamente, será posible intervenir tanto en el control como en el sistema siguiendo las siguientes explicaciones:

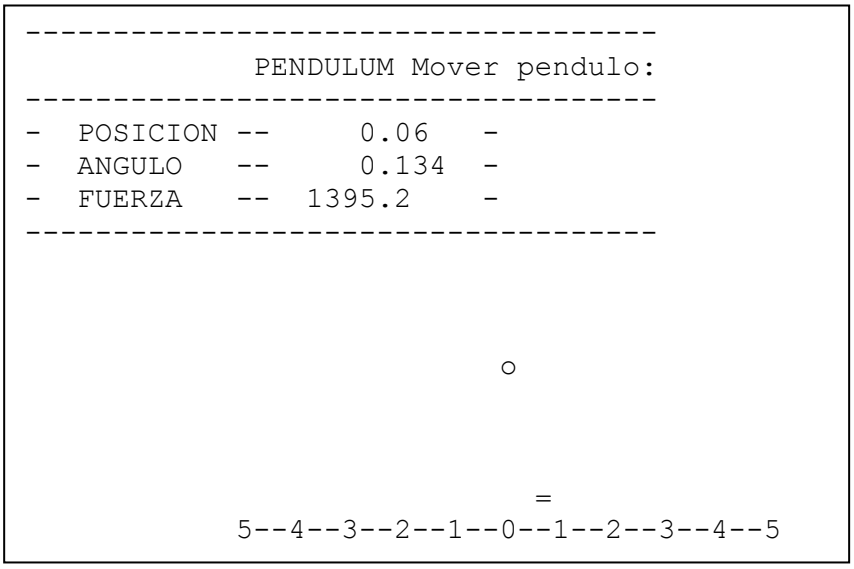
- Control: Mediante el teclado podemos cambiar la posición en que deseamos que se ponga la base del péndulo, con valores comprendidos entre [-5, +5] con sólo introducir el número correspondiente y pulsar enter. El número introducido se dividirá entre 10 (ya que la base sólo se puede desplazar a lo largo de 1 metro) y será aplicado al sistema de control. Si se desea finalizar la aplicación, basta con pulsar 99.

El aspecto que presentaría la pantalla de la aplicación de control sería el siguiente:



- Sistema: La única interacción que podemos tener con la aplicación del sistema es desplazar el péndulo deliberadamente hacia un lado u otro, es decir, introducir perturbaciones al sistema. Eso es posible introduciendo el número de grados que queremos que se añadan a su inclinación actual y pulsando enter.

El aspecto que presentaría la pantalla de la aplicación Sistema sería el siguiente:


PENDULUM Mover pendulo:
POSICION -- 0.06
ANGULO -- 0.134
FUERZA -- 1395.2
5--4--3--2--1--0--1--2--3--4--5

3. DISEÑO DEL ALGORITMO DE CONTROL

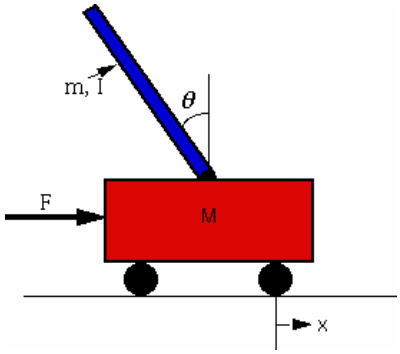
3.1 MODELADO DE UN PÉNDULO INVERTIDO

A continuación, vamos a modelar el sistema formado por el carro y el péndulo (péndulo invertido) para obtener sus ecuaciones y calcular su función de transferencia. Elegiremos un algoritmo de control ya realizado [22].

Generalmente, el sistema de péndulo invertido es modelado como un sistema lineal, y de aquí que el modelado sólo sea válido para oscilaciones pequeñas del péndulo.

3.1.1 DEFINICIÓN DEL PROBLEMA Y REQUERIMIENTOS

Tenemos un carro sobre el que se apoya una varilla que gira libremente alrededor de un pivote fijo. El carro es "empujado" con una fuerza de impulso, F. Hay que determinar las ecuaciones dinámicas que modelan el sistema, y linealizarlas en torno al punto en el cual el ángulo del péndulo $\theta = \pi$ (en otras palabras, asumir que péndulo no se mueve mas que unos pocos grados a ambos lados de la vertical, a la cual se le ha asignado el ángulo π). Posteriormente, hay que encontrar un controlador que satisfaga todos los requisitos dados abajo.



Para este problema, supondremos que los datos del sistema son:

M	Masa del carro	0.1 Kg
m	Masa del péndulo	0.04 Kg
b	Fricción del carro	0.1 N/m/sec
l	Longitud al centro de gravedad del péndulo	0.3 m
I	Inercia del péndulo	0.0048 Kg*m^2
F	Fuerza aplicada al carro	N
x	Posición de carro	m
Theta (θ)	Ángulo del péndulo desde la vertical inferior	rad

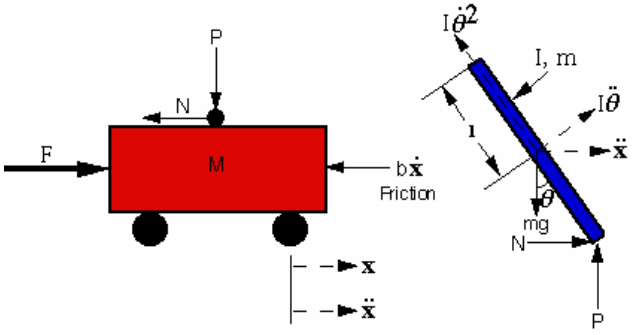
Los valores de masa, longitud y fricción los hemos calculado así, estimando que serán valores fáciles de conseguir en una futura construcción del sistema real.

Estableceremos los requerimientos de funcionamiento para una entrada escalón de 0.2 m :
Tiempo de establecimiento para 'x' y 'θ' menor de 5 segundos.
Tiempo de pico para 'x' menor de 2 segundos.
Máximo sobreimpulso para 'θ' menor de 20 grados (0.35 radianes).

Estos requerimientos vienen condicionados por las limitaciones del motor que veremos más tarde y sirven para fijar un objetivo mínimo de funcionamiento.

3.1.2 ANÁLISIS DE FUERZAS Y ECUACIONES DEL SISTEMA

Abajo tenemos los dibujos de los dos cuerpos aislados.



Sumando las fuerzas en dirección horizontal en el dibujo del carro, obtenemos las ecuaciones siguientes de movimiento:

$$M\ddot{x}+b\dot{x}+N=F$$

Haciendo lo mismo para el dibujo del péndulo tenemos:

$$N=m\ddot{x}+ml\ddot{\theta}\cos\theta-ml\dot{\theta}^2\sin\theta$$

Si sustituimos esta ecuación dentro la ecuación primera, obtenemos la primera ecuación de movimiento del sistema completo:

(1) $$(M+m)\ddot{x}+b\dot{x}+ml\ddot{\theta}\cos\theta-ml\dot{\theta}^2\sin\theta=F$$

Para obtener la segunda ecuación de movimiento, sumamos las fuerzas perpendiculares al péndulo. Resolviendo el sistema a lo largo del eje obtenemos la ecuación siguiente:

$$P\sin\theta+N\cos\theta-mg\sin\theta=ml\ddot{\theta}+m\ddot{x}\cos\theta$$

Para aislar los términos P y N de la ecuación superior, sumamos los momentos alrededor del centro de gravedad del péndulo para obtener la ecuación siguiente:

$$-Pl\sin\theta-Nl\cos\theta=I\ddot{\theta}$$

Combinando estas dos últimas ecuaciones, obtenemos la segunda ecuación dinámica:

(2) $$(I+ml^2)\ddot{\theta}+mgl\sin\theta=-ml\ddot{x}\cos\theta$$

Para trabajar con funciones lineales, debemos linealizar este conjunto de ecuaciones en torno a $\theta = \pi$. Asumiendo que $\theta = \pi + \phi$ (ϕ representa un ángulo pequeño desde la vertical). Por lo tanto, $\cos(\theta) = -1$, $\sin(\theta) = -\phi$, y $(d\theta/dt)^2 = 0$. Después de linealizar las dos ecuaciones de movimiento tenemos (donde u representa la señal de control o fuerza aplicada):

$$(M+m)\ddot{x}+b\dot{x}-ml\ddot{\phi}=u$$

$$(I+ml^2)\ddot{\phi}-mgl\phi=ml\ddot{x}$$

3.1.3 FUNCIÓN DE TRANSFERENCIA

Para obtener la función de transferencia del sistema linealizado analíticamente, primero debemos aplicar la transformada de Laplace a las ecuaciones de sistema (suponiendo condiciones iniciales nulas), lo que nos da los valores de las variables en el dominio s (X(s), U(s) y Φ(s)):

$$(\mathbf{I}+\mathbf{ml}^2)\Phi(s)s^2-\mathbf{mgl}\Phi(s)=\mathbf{ml}X(s)s^2$$
$$(\mathbf{M}+\mathbf{m})X(s)s^2+\mathbf{b}X(s)s-\mathbf{ml}\Phi(s)s^2=\mathbf{U}(s)$$

Ahora tenemos el ángulo como salida del sistema. Resolviendo la ecuación primera para X(s),

$$X(s)=\left[\frac{(\mathbf{I}+\mathbf{ml}^2)}{\mathbf{ml}}-\frac{\mathbf{g}}{\mathbf{s}^2}\right]\Phi(s)$$

y sustituyendo dentro de la segunda ecuación:

$$(\mathbf{M}+\mathbf{m})\left[\frac{(\mathbf{I}+\mathbf{ml}^2)}{\mathbf{ml}}+\frac{\mathbf{g}}{\mathbf{s}}\right]\Phi(s)s^2+\mathbf{b}\left[\frac{(\mathbf{I}+\mathbf{ml}^2)}{\mathbf{ml}}+\frac{\mathbf{g}}{\mathbf{s}}\right]\Phi(s)s-\mathbf{ml}\Phi(s)s^2=\mathbf{U}(s)$$

Agrupando términos, la función de transferencia es:

$$\frac{\Phi(s)}{\mathbf{U}(s)}=\frac{\dfrac{\mathbf{ml}}{\mathbf{q}}s^2}{s^4+\dfrac{\mathbf{b}(\mathbf{I}+\mathbf{ml}^2)}{\mathbf{q}}s^3-\dfrac{(\mathbf{M}+\mathbf{m})\mathbf{mgl}}{\mathbf{q}}s^2-\dfrac{\mathbf{bmgl}}{\mathbf{q}}s}$$

donde,

$$\mathbf{q}=\left[(\mathbf{M}+\mathbf{m})(\mathbf{I}+\mathbf{ml}^2)-(\mathbf{ml})^2\right]$$

De la función de transferencia superior podemos ver que hay un polo y un cero en el origen. Estos se pueden cancelar y la función de transferencia que queda es:

$$\frac{\Phi(s)}{\mathbf{U}(s)}=\frac{\dfrac{\mathbf{ml}}{\mathbf{q}}s}{s^3+\dfrac{\mathbf{b}(\mathbf{I}+\mathbf{ml}^2)}{\mathbf{q}}s^2-\dfrac{(\mathbf{M}+\mathbf{m})\mathbf{mgl}}{\mathbf{q}}s-\dfrac{\mathbf{bmgl}}{\mathbf{q}}}$$

3.1.4 VARIABLES DE ESTADO

Expresando las ecuaciones de sistema en forma de variables de estado [21]:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\Phi} \\ \ddot{\Phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \Phi \\ \dot{\Phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mml^2} \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \Phi \\ \dot{\Phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u$$

$$\dot{X} = A \cdot X + B \cdot U$$

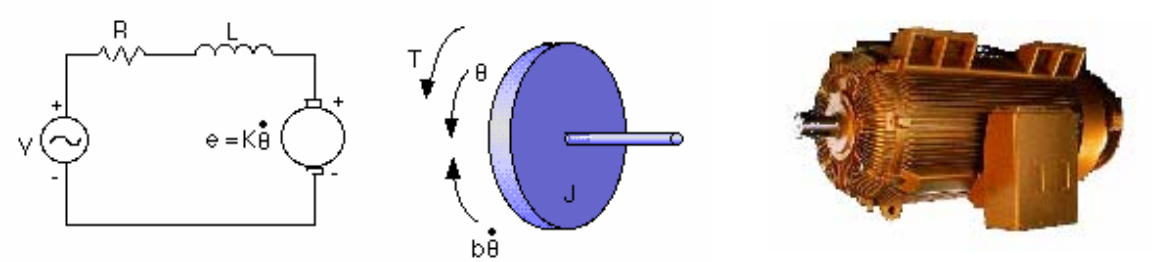
$$Y = C \cdot X + D \cdot U$$

La matriz C es de 2 por 4, porque tanto la posición del carro como el ángulo del péndulo forman parte de la salida. Es decir, las salidas del sistema son el desplazamiento de carro (x en metros) y el ángulo de desviación de péndulo (phi en radianes).

3.2 MODELADO DE UN MOTOR DC

3.2.1 DISPOSICION FÍSICA Y ECUACIONES DE SISTEMA

Un actuador muy común en sistemas de control es el motor DC. El motor proporciona un movimiento rotativo que emparejado con engranajes, ruedas y correas es capaz de proporcionar movimiento de traslación [23]. El circuito eléctrico de la armadura y el diagrama del rotor son mostrados en las siguientes figuras:



Los parámetros físicos de interés son los siguientes:
* momento de inercia del rotor (J) = (kg.m^2/s^2)
* damping ratio del sistema mecánico (b) = (Nms)
* constante de fuerza electromotriz (K=Ke=Kt) = (Nm/Amp)
* resistencia eléctrica (R) = (ohmios)
* inductancia eléctrica (L) = (H)
* entrada (V): Fuente de Voltaje
* salida (theta): ángulo de la varilla de giro

Asumimos que el rotor y el móvil están unidos de forma rígida.
El par del motor, **T**, se relaciona con la corriente de armadura "i", por un factor constante **Kt**. La fuerza electromotriz, **e**, se relaciona con la velocidad de rotación por las siguientes ecuaciones:

$$T = K_t i$$

$$e = K_e \dot{\theta}$$

En unidades del SI (que serán las que usaremos), **Kt** (constante de la armadura) es igual a **Ke** (constante del motor).
Atendiendo al dibujo anterior, podemos escribir las siguientes ecuaciones basadas en la ley Newton combinada con las leyes de Kirchhoff:

$$J\ddot{\theta} + b\dot{\theta} = Ki$$

$$L\frac{di}{dt} + Ri = V - K\dot{\theta}$$

Aunque estas últimas ecuaciones no nos serán necesarias, ya que nosotros utilizaremos un amplificador de corriente a continuación de la salida de control (voltaje), por lo que lo que introduciremos al motor será corriente y como su relación con el par (T) es lineal, no necesitamos introducir ningún control adicional al sistema motor-péndulo.

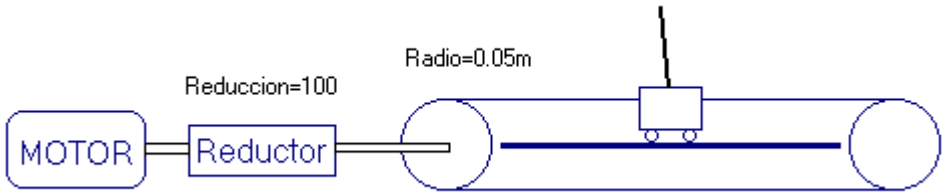
3.2.2 MOTOR COMERCIAL

Para posibilitar una futura implementación del sistema completo, usamos las características de un motor comercial del cual destacamos las siguientes.

Características:
Modelo ABB RS 130-E.

Velocidad Nominal:	Nn =	3000 min^{-1}
Potencia nominal:	Pn =	40 W
Corriente máxima en rotación lenta:	Imax =	9.7 A
Par por amperio (25ºC):	Kt =	0.0516 Nm/A

La unión entre el motor y el carro se realiza a través de una correa y unas ruedas dentadas. Utilizaremos también el motor con un reductor para aumentar su potencia.



Factor reducción: Reducción = 100
Radio de giro: Radio = 0.05 m

Calculamos con estos datos cuál es la fuerza máxima que podemos conseguir con este motor:

Par max = (Imax* Kt) * Reducción = 5 Nm
Par max = Fuerza max * Radio → Fuerza max = 100 N

3.3 CONTROL DIGITAL DEL PÉNDULO INVERTIDO USANDO EL MÉTODO DE VARIABLES DE ESTADO

En la sección de modelado del péndulo obtuvimos las ecuaciones del sistema y posteriormente se linealizaron, puesto que el sistema no era lineal. El proceso de linealizar sistemas no lineales es importante, ya que linealizar ecuaciones no lineales permite aplicar numerosos métodos de análisis lineal que proporcionen información acerca del comportamiento de los sistemas no lineales. Para el control del péndulo invertido, usaremos uno de esos métodos de control de sistemas lineales, el método de variables de estado.

Un sistema complejo posee muchas entradas y salidas que se relacionan entre sí en una forma complicada. Para analizar un sistema de este tipo, es esencial reducir la complejidad de las expresiones matemáticas, además de recurrir a una computadora que realice gran parte de los tediosos cálculos necesarios en el análisis. El enfoque en el espacio de estados para los análisis de sistemas es el más conveniente desde este punto de vista.

El uso de la notación matricial simplifica enormemente la representación matemática de los sistemas de ecuaciones. El incremento en la cantidad de variables de estado, de entradas o de salida no aumenta la complejidad de las ecuaciones. De hecho, el análisis de los sistemas complicados con entradas y salidas múltiples se realiza mediante procedimientos sólo ligeramente más complicados que los requeridos para el análisis de sistemas de ecuaciones diferenciales escalares de primer orden [21].

Seguidamente, veremos como se lleva a cabo el proceso de obtención del algoritmo de control con la ayuda de Matlab [24].

3.3.1 DISCRETIZACIÓN DEL SISTEMA

Dado que el control lo vamos a realizar mediante un computador, y que éste sólo puede operar con datos discretos, es necesario discretizar el sistema obtenido, ya que a partir de ahora es con el que vamos a trabajar. Por lo tanto, la primera cosa que hay que hacer es convertir el sistema de ecuaciones de continuo a discreto. Para ello, utilizaremos la función de Matlab *c2dm*. La función *c2dm* convierte las matrices que definen un sistema continuo en variables de estado, en uno discreto, asumiendo una frecuencia y un método de muestreo determinado. Para usar c2dm, debemos especificar seis argumentos: cuatro matrices de variables de estado (A, B, C, y D), el periodo de muestreo (Ts en sec/muestra), y el método de muestreo. El periodo de muestreo debe ser más pequeño que 1/(30*BW) sec, donde BW es el ancho de banda del sistema en lazo cerrado. El método que usaremos será el típico zero-order hold ('zoh') [21]. Suponiendo que en lazo cerrado el ancho de banda está alrededor de 1rad/seg para el péndulo, pondremos un periodo de muestreo de 1/100 seg/muestra. El resultado de *c2dm* serán cuatro matrices [F,G,H,J] que definen el sistema introducido pero en tiempo discreto.

El código de los cálculos realizados con Matlab es el siguiente:

```
%=================================================
% PARÁMETROS DEL SISTEMA
%=================================================

M = 0.1;    % masa del carro
m = 0.04;   % masa del pendulo
b = 0.1;    % friccion del carro
i = 0.0048; % inercia del pendulo
g = 9.8;    % fuerza de gravedad
l = 0.3;    % distacia al centro de masas del pendulo


%=================================================
% ECUACIONES EN VARIABLES DE ESTADO DEL SISTEMA
%=================================================

p = i*(M+m)+M*m*l^2; % denominador
A = [0      1              0           0;
   0 -(i+m*l^2)*b/p (m^2*g*l^2)/p  0;
   0      0              0           1;
   0 -(m*l*b)/p     m*g*l*(M+m)/p 0];

B = [0; (i+m*l^2)/p; 0; m*l/p];

C = [1 0 0 0;
```

```
   0 0 1 0];

D = [0;
   0];

%================================================
% DISCRETIZACIÓN DE LAS ECUACIONES DEL SISTEMA
%================================================

Ts=1/100;   % frecuencia de muestreo
[F,G,H,J]=c2dm (A,B,C,D,Ts,'zoh');
```

Que da como resultado:

```
F =
   1.0000    0.0100    0.0001    0.0000
        0    0.9909    0.0266    0.0001
        0   -0.0001    1.0016    0.0100
        0   -0.0226    0.3117    1.0016

G =
   0.0005
   0.0905
   0.0011
   0.2264

H =
    1    0    0    0
    0    0    1    0

J =
    0
    0
```

Ahora tenemos las ecuaciones de variables de estado en el siguiente formato, pero con diferentes valores, ya que hemos modificado ciertos parámetros físicos respecto del diseño original.

$$\begin{bmatrix} \dot{x}(k) \\ \ddot{x}(k) \\ \dot{\phi}(k) \\ \ddot{\phi}(k) \end{bmatrix} = \begin{bmatrix} 1 & 0.01 & 0.0001 & 0 \\ 0 & 0.9982 & 0.0267 & 0.0001 \\ 0 & 0 & 1.0016 & 0.01 \\ 0 & -0.0045 & 0.3119 & 1.0016 \end{bmatrix} \begin{bmatrix} x(k-1) \\ \dot{x}(k-1) \\ \phi(k-1) \\ \dot{\phi}(k-1) \end{bmatrix} + \begin{bmatrix} 0.0001 \\ 0.0182 \\ 0.0002 \\ 0.0454 \end{bmatrix} [u(k-1)]$$

$$y(k-1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x(k-1) \\ \dot{x}(k-1) \\ \phi(k-1) \\ \dot{\phi}(k-1) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} [u(k-1)]$$

Expresado en forma matricial

$$x[k] = F \cdot x[k-1] + G \cdot u[k-1]$$
$$y[k-1] = H \cdot x[k-1] + J \cdot u[k-1]$$

3.3.2 CONTROLABILIDAD Y OBSERVABILIDAD

Se dice que un sistema es controlable en el tiempo t_o si se puede llevar de cualquier estado inicial $x(t_o)$ a cualquier otro estado, mediante un vector de control sin restricciones, en un intervalo de tiempo finito [21].

Se dice que un sistema es observable en el tiempo t_o, si con el sistema en el estado $x(t_o)$, es posible determinar este estado a partir de la observación de la salida durante un intervalo de tiempo finito [21].

Se dice que el sistema es completamente observable si el estado $x(t_o)$ se determina a partir de la observación de y(t) durante un intervalo de tiempo finito, $t_o <= t <= t_1$. Por lo tanto, el sistema es completamente observable si todas las transiciones de estado afectan eventualmente a todos los elementos del vector de salida. El concepto de observabilidad es útil al resolver el problema de reconstruir variables de estado no mensurables a partir de variables que sí lo son en el tiempo mínimo posible [21].

El próximo paso es comprobar la controlabilidad y la observabilidad del sistema. Para que el sistema sea completamente controlable, la matriz de controlabilidad (C_T) debe tener rango **n**. El rango de la matriz es el número de filas independientes (o columnas). Asimismo, para que el sistema sea completamente observable, la matriz de observabilidad (O_T) debe tener también tener rango n.

$$C_T = \left[G | FG | F^2G | \cdots | F^{n-1}G \right]$$

$$O_T = \begin{bmatrix} H \\ HF \\ \vdots \\ HF^{n-1} \end{bmatrix}$$

Entonces, como nuestras matrices de controlabilidad y de observabilidad son de '4x4', su rango debe de ser 4.

```
%================================================
% ANÁLISIS DE LA OBSERVABILIDAD Y CONTROLABILIDAD
%================================================

% 'Controlabilidad.'
co = ctrb (F,G);
Controllability = rank (co)
% 'Observabilidad. '
ob = obsv (F,H);
Observability = rank (ob)
```

Lo que produce la siguiente respuesta:

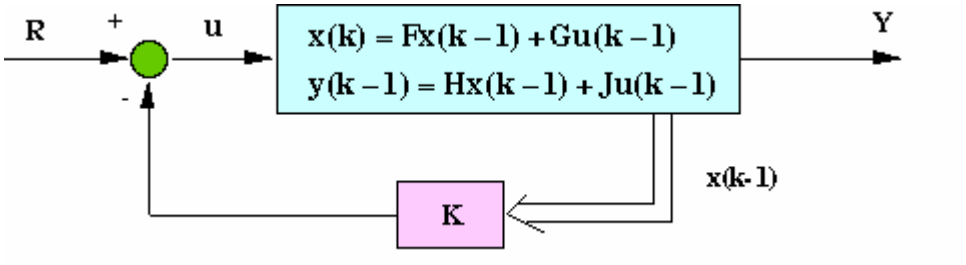
```
Controllability =

             4

Observability =

             4
```

Esto demuestra que nuestro sistema discreto es completamente controlable y observable.

3.3.3 DISEÑO DE CONTROL POR AJUSTE DE POLOS.

El esquema de un sistema de realimentación de variables de estado completo es mostrado a continuación. En él, 'R' representa la señal de referencia (posición deseada del carro), 'u' es la señal de

control que se introduce al sistema definido por las matrices [F,G,H,J], 'K' es la matriz de control, 'x' el vector de variables de estado e 'Y' el vector que contiene la salida del sistema.



El próximo paso es encontrar la matriz de control (K) suponiendo que los cuatro estados son mensurables. Para ello, usaremos un **Regulador Lineal Cuadratico (LQR)**. Este método le permite encontrar la matriz de control óptima que resulta de balancear entre errores de sistema y esfuerzo de control [21]. Para usar este método, necesitamos encontrar tres parámetros: Matriz índice de actuación (R), matriz de costo de estados (Q), y factores de peso. Para mayor sencillez, elegiremos la matriz índice de actuación igual a 1 (R=1), y la matriz de costo de estados (Q) igual a H' x H. Los factores de peso serán elegidos por ensayo y error. La matriz de costo de estados (Q) tiene la estructura siguiente

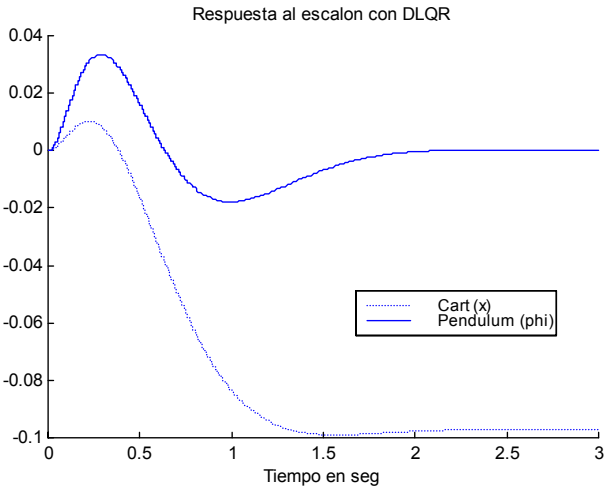
```
Q =1 0 0 0
   0 0 0 0
   0 0 1 0
   0 0 0 0
```

El elemento en la posición [1,1] será usado para medir la posición de carro y el elemento en la posición [3,3] será usado para medir el ángulo del péndulo. El peso de cada factor para la posición de carro y el ángulo de péndulo será elegido individualmente.
Ahora estamos listos para encontrar la matriz de control (K) y ver la respuesta del sistema.

```
%==================================================
% DISEÑO LQR
%==================================================

%disp 'Ahora buscaremos el controlador K'
T=0:0.01:5;
U=0.2*ones(size(T));

x=5;                % factor de ponderación para la posición del carro
y=0.1;              % factor de ponderación para el ángulo del pendulo

Q=[x 0 0 0;
   0 0 0 0;
   0 0 y 0;
   0 0 0 0];

R = 1;

K = dlqr(F,G,Q,R)
[Y,X]=dlsim(F-G*K,G,H,J,U);

stairs(T,Y)
legend('Cart (x)','Pendulum (phi)')
```

El peso de los factores (x e y) ha sido modificado hasta encontrar unos valores que resolviesen los requerimientos temporales y como veremos más tarde, que el esfuerzo de control se pueda llevar a cabo con un motor comercial de medianas prestaciones. Los valores que resuelven el problema son x=5, y=0.1.

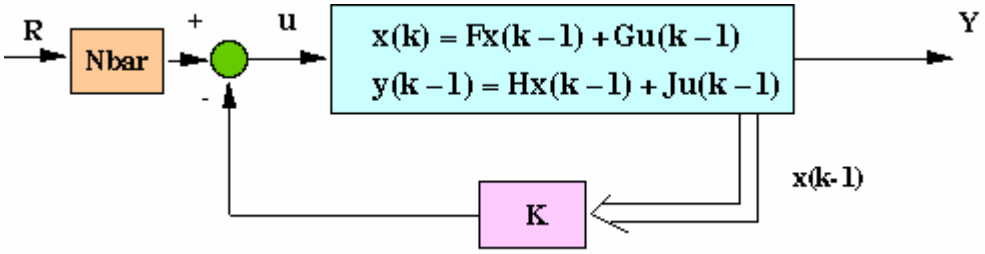
La curva superior representa el ángulo de péndulo, en radianes, y la curva inferior representa la posición de carro en metros ante una entrada escalón de valor 0.2 m.



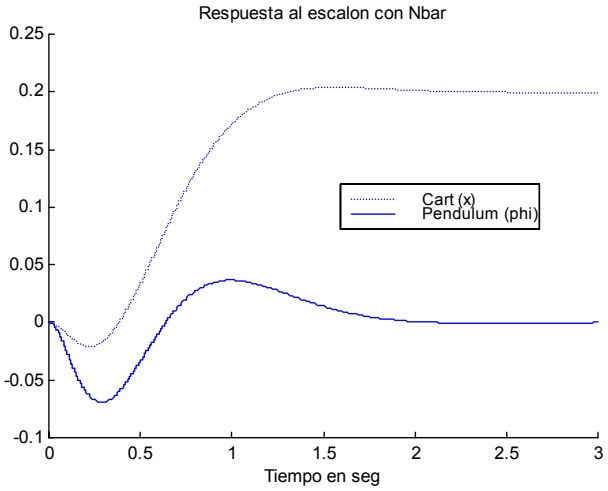
En este dibujo, podemos ver que todos los requerimientos han sido satisfechos excepto el error en estado estacionario de la posición del carro (x). Para corregir este error, introduciremos un factor de escalado en la trayectoria directa (Nbar), tal como se explica en el siguiente apartado.

3.3.4 ENTRADA DE REFERENCIA

A diferencia de otros métodos de diseño, el método "full-state feedback system" no compara la salida con la referencia; en lugar de eso, compara todas las variables de estado multiplicadas por la matriz de control (K*x) con la referencia (ver el esquema superior). Así, que no es de esperar que la salida sea igual que la entrada. Para obtener la salida deseada, hay que escalar la entrada de referencia para que la salida sea igual a la señal de referencia. Esto se puede hacer fácilmente introduciendo un factor de escalado llamado **Nbar**. El esquema con Nbar es mostrado posteriormente.



Para encontrar su valor, procedemos a su cálculo mediante ensayo y error. Después de varias pruebas, Nbar = -2.06 proporciona la respuesta satisfactoria. La respuesta al escalón es la siguiente.




```
%================================================
% ENTRADA DE REFERENCIA
%================================================
Nbar=-2.06;
[Y,X]=dlsim(F-G*K,G*Nbar,H,J,U);
stairs(T,Y)
legend('Cart (x)','Pendulum (phi)')
```

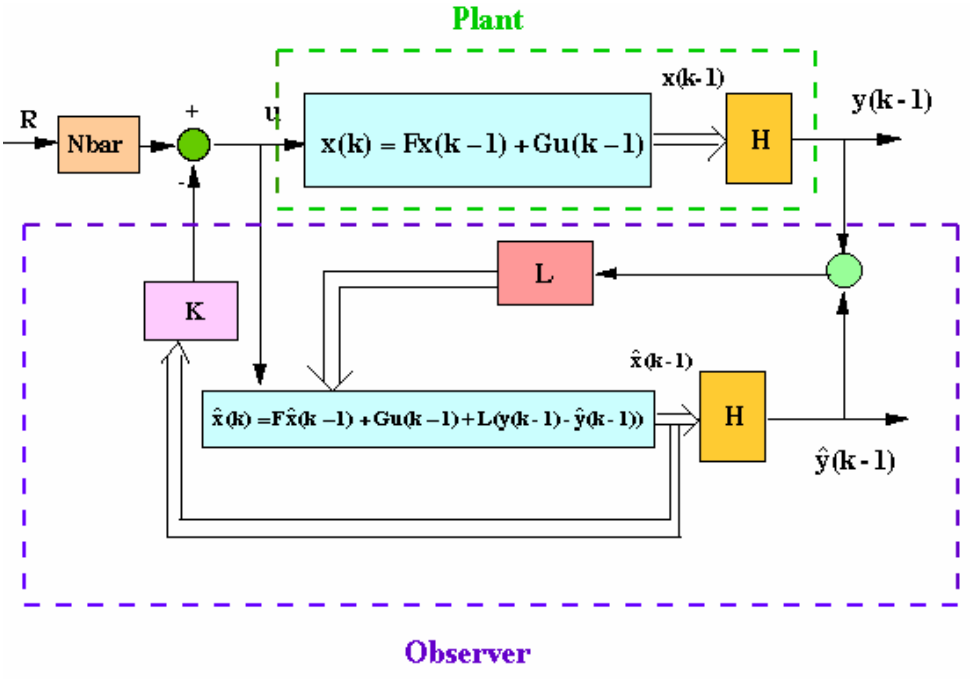
Puede observarse que el error en estado estacionario de la posición del carro ha sido eliminado. Por lo tanto ya tenemos un diseño que satisface todos los requisitos.

3.3.5 DISEÑO DE UN OBSERVADOR

El sistema diseñado satisface todos requisitos dados; sin embargo, éste fue diseñado asumiendo que todos los estados eran mensurables. Este supuesto no es válido en nuestro caso, por lo que desarrollaremos un método para estimar algunos estados de la planta a través de la información que sea disponible de la misma. El sistema que estima los estados de otro sistema con base en las mediciones de las variables de salida y de control se llama **estimador** u **observador**. Los observadores de estado pueden diseñarse si y solo si se satisface la condición de observabilidad [21].

Así, en esta sección obtendremos un observador de orden completo para tener información de esos estados que no son mensurables.

Un esquema del sistema de control con el observador viene a continuación.



Para obtener el observador, primero, hay que hallar la matriz L. Para encontrarla hay que localizar los polos del sistema sin el observador (los polos de F G*K).

```
%================================================
% DISEÑO DE UN OBSERVADOR
%================================================
```

```
poles=eig(F-G*K)

poles =
  0.9691 + 0.0257i
  0.9691 - 0.0257i
  0.9472 + 0.0066i
  0.9472 - 0.0066i
```

Tenemos que situar los polos del observador de forma que el observador trabaje más rápido que el sistema sin el observador. Posicionaremos pues, los polos del observador bastante más a la izquierda, en [0.3 0.31 -0.32 -0.33]. Estos polos pueden ser cambiados más tarde, si es necesario. Ahora, usaremos la función de Matlab 'place' para encontrar la matriz L.

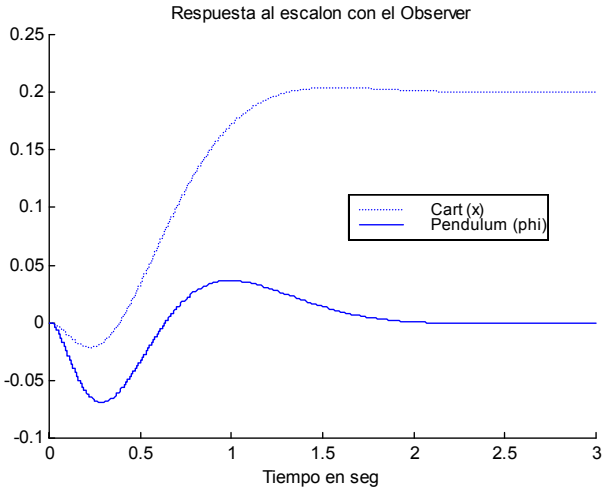
```
P = [-0.3 -0.31 -0.32 -0.33];

L = place (F',H',P)'

L =
   2.6208   -0.0105
 171.3022   -1.3581
  -0.0318    2.6333
  -5.2021  173.5720
```

Ahora obtendremos la respuesta en conjunto del sistema incluyendo el observador.

```
Fce = [F-G*K            G*K;
       zeros(size(F))    (F-L*H)];
Gce = [G*Nbar;
      zeros(size(G))];
Hce = [H zeros(size(H))];
```

```
Jce = [0;0];

[Y,X] = dlsim (Fce,Gce,Hce,Jce,U);


stairs (T,Y)
legend ('cart (x)','pendulum (phi)')
```

Así tendremos la siguiente respuesta al escalón



Como podemos ver, es igual que la anterior, y todos los requerimientos han sido satisfechos.

3.3.6 IMPLEMENTACIÓN DE LOS ALGORITMOS EN LOS PROGRAMAS.

Los resultados del proceso de diseño no son más que unas cuantas matrices. Pero para que esas matrices realicen el propósito para el que fueron concebidas es necesario algo más. Para ello, se desarrolló en Matlab un programa que recogiese el algoritmo y cuyos resultados fuesen idénticos a los de la simulación de Matlab. El resultado fue (en forma resumida) el siguiente:

```
% Se inicializan las variables
yys=[0 0]    ;    xxs=[0 0 0 0];
xx =[0 0 0 0];    yy =[0 0];
uu =0;
i=1;

for i=1:length(R)
   %-- tomar datos -------
   x = xx(i,:)';
   y = yy(i,:)';
   ys=yys(i,:)';
   xs=xxs(i,:)';

   %-- calculos ---------
      u =R(i)*Nbar-(xs'*K');  %-- control ---------
   xs=F*xs+G*u+L*(y-ys);      %-- control - observer -------
   x =(F*x + G*u);            %-- planta ---------
   y =x'*H';                  %-- planta ---------
   ys=xs(:)'*H';              %-- control - observer -------

   %-- anotar resultados
   xx(i+1,:) =x(:)';
   yy(i+1,:) =y(:)';
```

```
   uu(i+1)    =u;
   xxs(i+1,:)=xs(:)';
   yys(i+1,:)=ys(:)';
end
```

Donde:
X: vector con las variables de estado del sistema real
Y: vector con la salida del sistema real (posición, ángulo)
Xs: Igual que X pero de las variables estimadas por el observer
Ys: Igual que Y pero de las variables estimadas por el observer
U: almacena el valor de la entrada a la planta.
R: contiene la entrada de referencia
Xx, YY, uu, xxs, yys: son variables que almacenan los resultados para posteriormente mostrarlos en una gráfica y compararlos con los de la simulación de Matlab.

Posteriormente, este código se trasladó a Ada, con algunas variaciones y sin el almacenamiento de los datos.

El código que ejecutará la aplicación de control será el etiquetado en los cálculos como `%-- control ---` y el que se desarrolla en la aplicación que simula el sistema físico el etiquetado con `%-- planta ---`

Se elaboró además un fichero donde estuviesen todos los cálculos del procedimiento de diseño en función de unos parámetros iniciales, para que en futuras modificaciones, el trabajo de recalcular las ecuaciones fuese semiautomático. Su contenido es el siguiente:

```
%-------------------------------------------------------------------
%------ Fichero para el cálculo de las ecuaciones del sistema   -----
%-------------------------------------------------------------------

%================================================
% PARÁMETROS DEL SISTEMA
%================================================

M = 0.1;    % masa del carro
m = 0.04;   % masa del pendulo
b = 0.1;    % friction del carro
i = 0.0048; % inercia del pendulo
g = 9.8;    % fuerza de gravedad
l = 0.3;    % distancia al centro de gravedad del pendulo

%================================================
% ECUACIONES EN VARIABLES DE ESTADO DEL SISTEMA
%================================================

p = i*(M+m)+M*m*l^2; % denominador
A = [0      1              0           0;
   0 -(i+m*l^2)*b/p (m^2*g*l^2)/p  0;
   0      0              0           1;
   0 -(m*l*b)/p     m*g*l*(M+m)/p 0];

B = [0; (i+m*l^2)/p; 0; m*l/p];

C = [1 0 0 0;
   0 0 1 0];

D = [0;
   0];
```

```
%================================================
% DISCRETIZACIÓN DE LAS ECUACIONES DEL SISTEMA
%================================================

Ts=1/100;   % frecuencia de muestreo
[F,G,H,J]=c2dm (A,B,C,D,Ts,'zoh');

%================================================
% ANÁLISIS DE LA OBSERVABILIDAD Y CONTROLABILIDAD
%================================================

% 'Controlabilidad.'
co = ctrb (F,G);
Controllability = rank (co)
% 'Observabilidad. '
ob = obsv (F,H);
Observability = rank (ob)
%pause

%================================================
% DISEÑO LQR
%================================================

%disp 'Ahora buscaremos el controlador K'
T=0:0.01:8;
U=0.2*ones(size(T));

x=5;              % factor de ponderación para la posición del carro
y=0.1;            % factor de ponderación para el ángulo del pendulo

Q=[x 0 0 0;
   0 0 0 0;
   0 0 y 0;
   0 0 0 0];

R = 1;

K = dlqr(F,G,Q,R)
[Y,X]=dlsim(F-G*K,G,H,J,U);
stairs(T,Y)
legend('Cart (x)','Pendulum (phi)')
%pause

%================================================
% ENTRADA DE REFERENCIA
%================================================

Nbar=-2.06;
[Y,X]=dlsim(F-G*K,G*Nbar,H,J,U);
stairs(T,Y)
legend('Cart (x)','Pendulum (phi)')
%pause;

%================================================
% DISEÑO DE UN OBSERVADOR
%================================================

disp 'Ahora buscaremos los polos del controlador'
poles = eig (F-G*K)
disp 'Y ahora la localizacion adecuada de los mismos para el observer'
P = [-0.3 -0.31 -0.32 -0.33];
L = place (F',H',P)'
```

```
Fce = [F-G*K            G*K;
      zeros(size(F))    (F-L*H)];
Gce = [G*Nbar;
     zeros(size(G))];
Hce = [H zeros(size(H))];
Jce = [0;0];

[Y,X] = dlsim (Fce,Gce,Hce,Jce,U);
stairs (T,Y)
legend ('cart (x)','pendulum (phi)')
grid;
zoom;
%================================================
```

4. IMPLEMENTACIÓN DE LOS PROGRAMAS

Las aplicaciones (control y sistema) como hemos dicho están estructuradas de la misma forma. En primer lugar, hay un programa llamado *Main* y tres paquetes llamados Dispositivos, Protegido y Pendulum. El programa *Main* tan sólo realiza una acción que es referenciar al paquete Pendulum, lo que provoca que se activen las tareas definidas en ese paquete.

La construcción de las dos aplicaciones se desarrolló en un primer momento sobre Linux, usando el compilador de Ada Gnat 3.13p. Para ello, se utilizaron algunos paquetes externos, los cuales serán expuestos seguidamente. Llegado el punto final, se trasladaron las dos aplicaciones a MaRTE O.S.

Próximamente, se describirá con profundidad cada uno de los paquetes, primero de la aplicación CONTROL y posteriormente de SISTEMA, explicando qué hacen y cómo. Se podrá ver también el código fuente de los mismos y sus relaciones con otros paquetes externos.

4.1 MÓDULOS AUXILIARES

En la realización de las aplicaciones de Control y Sistema se hizo uso de diversos módulos externos ya implementados que nos facilitaban la realización de ciertas tareas. A continuación, se explicará para qué sirve cada uno de ellos y cuál es la especificación de las partes que se usan de los mismos.

4.1.1 MATRICES

Este paquete proporciona funciones básicas para realizar operaciones con vectores y matrices. Tan sólo la usa la tarea Control cuando realiza operaciones con las matrices que definen el sistema o el controlador. Su especificación es la siguiente:

```
package matrices is
    type Matrix is array (Integer range <>, Integer range <>) of Float;
    type Vector is array (Integer range <>) of Float;
    function "*" (A,B : Matrix) return Matrix;
    function "+" (A,B : Matrix) return Matrix;
    function "*" (M : Matrix; V : Vector) return Vector;
    function "+" (A,B : Vector) return Vector;
    function "-" (A,B : Vector) return Vector;
    function "*" (A : Vector; B : float) return Vector;
    function "*" (A,B : Vector) return float;
    unmatched_dimensions : exception;
end Matrices;
```

4.1.2 TIME_MEASUREMENTS

Este paquete, incluido en la librería de funciones de MaRTE O.S. se usó solamente una vez para realizar las medidas de los tiempos de ejecución de las tareas y de uso de los objetos compartidos, necesarias para el análisis de planificabilidad. Para medir el tiempo de ejecución de la rutina de atención a la interrupción, se usó su homólogo en C, `medidas_tiempos`. Su especificación viene a continuación:

```
package Time_Measurements is

   type Measure_Id is range 0 .. 49;
   for Measure_Id'Size use 8;

   procedure Reset_Measures;
```

```
   pragma Inline (Reset_Measures);

   procedure Take_Measure (Id : in Measure_Id);
   pragma Inline (Take_Measure);

   procedure Show_Interval (From : in Measure_Id; To : in Measure_Id);
   pragma Inline (Show_Interval);

   procedure Show_Each_Interval (From : in Measure_Id; To : in Measure_Id);
   pragma Inline (Show_Each_Interval);

end Time_Measurements;
```

4.1.3 FULL_CONSOLE_MANAGEMENT

Este paquete, incluido también en la librería de funciones de MaRTE O.S. es usado para las operaciones que tienen que ver con la pantalla (situación, borrado, etc.). Posee numerosas funciones para muchas otras tareas, pero únicamente mostraremos la especificación de las funciones que usamos:

```
package Full_Console_Management is

   subtype Rows is Natural range 0..24;
   subtype Columns is Natural range 0..79;

   type Position is
     record
       Row    : Rows;
       Column : Columns;
     end record;
   pragma Convention (C, Position);

   -- Upper left corner of the screen
   UpLeft_Position   : constant Position := (Row => 0, Column => 0);
   -- Lower right corner of the screen
   DownRight_Position: constant Position := (Row    => Rows'Last,
                                             Column => Columns'Last);

   ------------------------------------------------------------------------
   -- Position the cursor -------------------------------------------------
   ------------------------------------------------------------------------

   procedure Set_Cursor (To : in Position);
   pragma Export (C, Set_Cursor, "set_cursor");
   pragma  Export_Procedure  (Internal  =>Set_Cursor,External=>"set_cursor",
                              Mechanism => Reference);
   procedure Set_Cursor (Row : in Rows; Column : in Columns);
   --  Establish the position from where the characteres will be printed.

   ------------------------------------------------------------------------
   -- Clear the screen ----------------------------------------------------
   ------------------------------------------------------------------------

   procedure Clear_Screen;
   pragma Import (C, Clear_Screen, "clrscr");
   --  The whole screen takes the active background color.

   ------------------------------------------------------------------------
   -- Console input configuration -----------------------------------------
   ------------------------------------------------------------------------

   procedure Disable_Echo;
```

```
   pragma Export (C, Disable_Echo, "disable_echo");
   -- Input characters are not echoed

end Full_Console_Management;
```

4.2 IMPLEMENTACIÓN DE LA APLICACIÓN DE CONTROL

4.2.1 PAQUETE DISPOSITIVOS

En el paquete Dispositivos, como se ha dicho en el capítulo 2, se agrupan las funciones de manejo tanto del driver como del monitor y del teclado, encapsulándolas y abstrayendo su funcionamiento para las tareas que utilicen alguno de estos elementos.

Este paquete implementa 5 funciones: una para imprimir datos por pantalla, otra para obtener entradas desde teclado y tres para operar sobre el driver de la tarjeta A/D D/A.

En su especificación podemos encontrar las siguientes llamadas a paquetes externos:

```
with Ada.Text_Io        ; use Ada.Text_Io;
with Ada.Integer_Text_Io; use Ada.Integer_Text_Io;
with Ada.Float_Text_Io  ; use Ada.Float_Text_Io;

with Full_Console_Management ; use Full_Console_Management;
pragma Elaborate_All(Full_Console_Management);

with Driver_Marte       ; use Driver_Marte;
pragma Elaborate_All(Driver_Marte);
```

Y las siguientes definiciones de datos y operaciones:

```
package Dispositivos is
   ---------------------------------------
   subtype Posicion_T is Float;
   subtype Angulo_T   is Float;
   subtype Fuerza_T   is Float;
   --------------------------
   procedure Muestra_Datos (Pos,Ang,Fuerza : in Float);
   function  Lee_Angulo   return Angulo_T;
   function  Lee_Operador return Posicion_T;
   --------------------------
   protected Accesos is
      function  Lee_Posicion   return Posicion_T;
      procedure Aplica_Fuerza (Fuerza: in Fuerza_T);
   end Accesos;
   ------------------------------------
end Dispositivos;
```

En este paquete se hace uso de paquetes estándar de Ada para la entrada / salida de datos, externos (`Full_Console_Management`) para el manejo de la pantalla (posicionamiento y borrado) y el que especifica las funciones de acceso al driver escrito en C (`Driver_Marte`).

Aquí definimos tres tipos de datos que serán utilizados tanto en este paquete como en el que contiene las tareas, `Posicion_T, Angulo_T y Fuerza_T`. La posición se lee en el canal de entrada 0 de la tarjeta, el ángulo en el 1 y la fuerza se escribe en el 0 de salida. Por esta razón, es por la que los accesos a las funciones que usan el canal 0 de la tarjeta se hayan introducido en un objeto protegido llamado Accesos, ya que no se pueden realizar lecturas y escrituras simultáneamente.

Al comienzo del cuerpo del paquete, está también el dato de la frecuencia de muestreo del driver, que será también la frecuencia a la que trabajará la tarea control, ya que ésta se activa con cada muestreo del mismo.

El paquete dispositivo tiene un proceso de inicialización en el que básicamente se realizan las siguientes actividades:
- Se crean algunas variables para el manejo general.
- Se inicializa el driver de la tarjeta.
- Se pregunta por la fuerza máxima aplicable por el control (esto optimiza la precisión en las señales de salida, ya que distribuye el rango de fuerzas aplicables entre el rango de voltajes de salida).
- Se pinta el marco para los datos que van a mostrarse por pantalla.

```
package body  Dispositivos is
   ---------------------------------------
   Fd_Pos : Integer:=0;
   Fd_Ang : Integer:=1;

   Freq   : Integer:= 100;

   F_Max : Float :=20.0;

   ---------------------------------------
   Titulo:String:="             PENDULUM               ";
   Linea:String :="-----------------------------------";
   M11:String   :="- POSICION  -";
   M21:String   :="- ANGULO    -";
   M31:String   :="- FUERZA    -";
   Limpia:String:=             "-                     -";

   Base:String  :="         5--4--3--2--1--0--1--2--3--4--5";
   Limp:String  :="                                                    ";
   Movil:String :="=";
   Pendulo:String:="o";
   Pi:Float:=3.141592653589;
```

Aquí es donde se sitúan las funciones del paquete, pero por claridad se mostrarán y explicarán individualmente más adelante.

```
begin
   -- Abrir fichero --
   if (Init_Module( Freq )=-1) then
      Put("El init no va");
   end if;
   Open_Tarjeta;

   -- Pintar marco --
   Clear_Screen;
   Put("F_max?:");
   Get(F_Max);
   Clear_Screen;

   Disable_Echo;
   Set_Cursor(3,1);
   Put_Line(Linea);
   Put_Line(Titulo);
   Put_Line(Linea);
   Put_Line(M11);
   Put_Line(M21);
   Put_Line(M31);
   Put_Line(Linea);
```

```
   Set_Cursor(18,4);
   Put(Base);
   ---------------------------------------
end Dispositivos;
```

A continuación, se describen de forma mas detallada las funciones que implementa el paquete dispositivos:

Muestra_datos : Se encarga de poner en pantalla tanto la posición (metros) y el ángulo (radianes) del sistema como la fuerza que esta siendo aplicada (Newtons). Además, dibuja un pequeño gráfico en modo texto de cuál es la situación del sistema. Los datos que utiliza le son pasados por la tarea que le llama (Display). Para el manejo de la pantalla, utiliza un paquete externo () que le permite situar el cursor y borrar la pantalla entre otras cosas. La utilización de este paquete era ligeramente diferente en Linux y en MaRTE O.S., lo que implica algunas diferencias entre ambas implementaciones.

```
---------------------------------------
   procedure Muestra_Datos (Pos,Ang,Fuerza : in Float) is
      X:Float:=20.0;
      A:Float:=0.0;
   begin
      Set_Cursor(6,14);      Put(Limpia);
      Set_Cursor(6,19);      Put(Pos,3,2,0);

      Set_Cursor(7,14);      Put(Limpia);
      Set_Cursor(7,18);      Put(Ang,4,3,0);

      Set_Cursor(8,14);      Put(Limpia);
      Set_Cursor(8,18);      Put(Fuerza,4,1,0);
      ----------------
      X:=(Pos*30.0)+20.0;

      A:=X-(Ang*10.0);
      Set_Cursor(17,1);      Put(Limp);
      Set_Cursor(17,Integer(X)+8);      Put(Movil);
      Set_Cursor(13,1);      Put(Limp);
      Set_Cursor(13,Integer(A)+8);      Put(Pendulo);
      Set_Cursor(1,1);
   end Muestra_Datos;
---------------------------------------
```

Lee_ángulo : Realiza una lectura del canal 1 de entrada del driver. Posteriormente, opera con el dato recibido para transformarlo en un valor en radianes y lo devuelve. El rango de grados mensurables se estima entre +/- 40 grados desde la vertical, ya que ángulos mayores indican que el péndulo se ha caído y no tiene ningún sentido seguir midiéndolos y así de esta forma, cuanto más estrecho es el rango medible, mayor será la precisión de las medidas.

```
---------------------------------------
   function Lee_Angulo  return  Angulo_T  is
      Ang_Max: constant Float :=Pi*40.0/180.0;
      Factor_Out_Ang: constant Float:=4096.0/(2.0*Ang_Max);
      Factor_In_Ang:  constant Float:=(4.0*Ang_Max)/4096.0;
      Resto_In_Ang:   constant Float:=3.0*Ang_Max;
      Cod_In : Short;
      Angulo : Angulo_T;
   begin
      Cod_In:=Read_Short(Fd_Ang);
      Angulo:=( Float(Cod_In) * Factor_In_Ang) - Resto_In_Ang;

      return Angulo;
   end Lee_Angulo;
```

```
   ---------------------------------------
```

Lee_Operador : Se queda bloqueado esperando una entrada del teclado. El dato recibido es obtenido y devuelto. El dato introducido por teclado significa la posición sobre la que queremos que se sitúe la base del péndulo. También hace uso del paquete `Full_Console_Management`.

```
   ---------------------------------------
function  Lee_Operador return Posicion_T is
      Dato : Float;
   begin
      Set_Cursor(4,25);
      Put("Posición deseada:");
      Get(Dato);
      return Posicion_T(Dato);
 end Lee_Operador;
   ---------------------------------------
```

Accesos : Es un objeto protegido que contiene las funciones que operan sobre un mismo canal de la tarjeta, por lo que hay que realizarlas en exclusión mutua. Son las siguientes:

Lee_posicion : Realiza una lectura del canal de entrada 0 de la tarjeta. Después, opera con el dato recibido para transformarlo en un valor en metros y lo devuelve. El rango del dato posición es de medio metro a cada lado de la posición central de 0.

Aplica_fuerza : Escribe en el canal 0 de salida de la tarjeta el valor de voltaje que será aplicado al motor, que a su vez lo transformará en fuerza a aplicar sobre el sistema. En el lado del simulador del sistema esta señal será leída por el canal 0 de entrada. El rango de 'fuerza' aplicable se ajustará mediante lo introducido en la inicialización del paquete dispositivos.

```
protected body  Accesos is
      ---------------------------------------
      function  Lee_Posicion   return Posicion_T is
         Pos_Max: constant Float :=0.5;
         Factor_Out_Pos: constant Float:=4096.0/(2.0*Pos_Max);
         Factor_In_Pos:  constant Float:=(4.0*Pos_Max)/4096.0;
         Resto_In_Pos:   constant Float:=3.0*Pos_Max;
         Cod_In   : Short;
         Posición : Posicion_T;
      begin
         Cod_In:=Read_Short(Fd_Pos);
         Posicion:=((Float(Cod_In)*Factor_In_Pos)-Resto_In_Pos);

         return Posición;
      end Lee_Posicion;
      ---------------------------------------
      procedure Aplica_Fuerza (Fuerza : in Fuerza_T) is

         Factor_Out_F: constant Float:=4096.0/(2.0*F_Max);
         Factor_In_F:  constant Float:=(4.0*F_Max)/4096.0;
         Resto_In_F:   constant Float:=3.0*F_Max;
         Cod_Out : Short;
      begin
         Cod_Out:=Short(  (Float(Fuerza) + F_Max ) * Factor_Out_F );

         Write_Short(Fd_Pos,Cod_Out);
      end Aplica_Fuerza ;
      ---------------------------------------
   end Accesos;
```

4.2.2 PAQUETE PROTEGIDO.

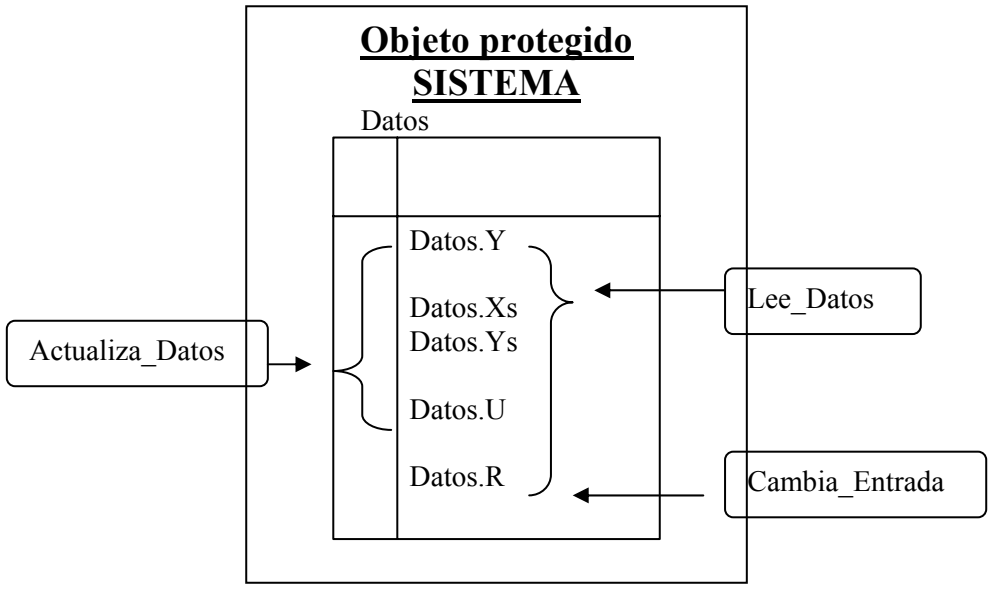
Este paquete contiene la estructura de datos que albergará los principales datos de la aplicación y proporciona mecanismos para el acceso y manipulación de los mismos de una forma exclusiva.

La razón por la que los datos se han introducido en un objeto protegido radica en que los accesos a los mismos tienen que hacerse en exclusión mutua [7]. Algunas características de la exclusión mutua en objetos protegidos son:

- Un objeto protegido garantiza que los procedimientos y procedimientos condicionados (entries) relativos al mismo se llevan a cabo bajo exclusión mutua de lectura / escritura de los datos encapsulados.
- Las funciones definidas en un objeto protegido pueden ejecutarse concurrentemente con otras funciones, pero son ejecutadas en exclusión mutua respecto de los procedimientos y entries.
- Cuando varias tareas ejecutan concurrentemente subprogramas sobre un objeto protegido, todas menos una de ellas son suspendidas hasta que aquella finalice. El orden en que las tareas en espera se ejecutan, no está definido por el lenguaje. Cuando el anexo de tiempo real está soportado, el acceso al objeto puede priorizarse.

Los datos que se almacenan en el objeto protegido son:

Xs: Vector que contiene las variables de estado estimadas (posición, velocidad lineal, ángulo, velocidad angular).
Y: Vector que contiene la salida real del sistema (posición, ángulo).
Ys: Vector que contiene la salida estimada del sistema.
U : Señal de salida del controlador.
R : Señal de referencia del controlador. Indica la posición deseada del carro. Su valor inicial es 0 y sus valores están comprendidos entre [-0.5, +0.5] m.



Esquema del Objeto Protegido Sistema

La función `Lee_Datos` es utilizada por las tareas *Control* y *Display*, el procedimiento `Cambia_Entrada` por la tarea *Operario* y el procedimiento `Actualiza_Datos` por la tarea *Control*.

Cuando varias tareas deben compartir recursos, se puede producir el fenómeno de la inversión de prioridad; aunque no puede evitarse, existen técnicas para acotar su duración [18]. En nuestro caso,

hemos usado el protocolo de techo de prioridades PCP (priority ceiling protocol) o HLP (highest locker's priority): cada sección crítica se ejecuta con la prioridad más alta de todas las tareas que la van a llamar. Para que esto funcione, dentro de una sección crítica (objeto protegido), una tarea no debe suspenderse a sí misma. La razón de esta elección estriba en que es sencillo de implementar, además de que se ajustaba perfectamente a nuestras necesidades (todas las tareas hacen uso del objeto protegido) e introduce menos sobrecarga por cambios de contexto que otros protocolos de sincronización.

Su especificación es la siguiente:

```
with Dispositivos ; use Dispositivos;
with Matrices;     use Matrices;

package Protegido  is
   ----------------------
   type Datos_T is record
      -- Valores de inicialización --
      Xs:Vector(1..4):=(others=>0.0);
      Y,Ys:Vector(1..2):=(others=>0.0);
      U : Float :=0.0;
      R : Float :=0.0;
   end record ;
   -----------------------
   protected Sistema is
      pragma Priority (10);

      function  Lee_Datos       return Datos_T ;
      procedure Cambia_Entrada  (Entrada    : in Float);
      procedure Cambia_Nbar     (Entrada    : in Float);
      procedure Actualiza_Datos (IXs,IY,IYs : in Vector; Iu :in Float);
   private
      Datos:Datos_T;
   end Sistema;
   -----------------------
end Protegido;
```

Y las funciones y procedimientos que contiene son:

Lee_Datos : Devuelve una estructura Datos_T con los datos que hay en el objeto protegido. Esta función es utilizada por las tareas Control y Display.

Cambia_Entrada : Recoge un valor que será almacenado en la variable R de la estructura y que representa la señal de referencia del sistema. Es utilizado por la tarea Operario.

Actualiza_Datos : Actualiza en la estructura los datos recibidos. Es vital guardar esta información, ya que las decisiones que toma el control dependen de cuáles fueron los valores de las variables de estado en el momento anterior.

Todos los datos utilizados son de tipo `Float` o `Vector`. Para usar el tipo `Vector` se hace uso del paquete `Matrices`.

En seguida, veremos su cuerpo:

```
package body Protegido  is
   -----------------------
   protected body Sistema is
      -------------------------------
      function Lee_Datos return Datos_T is
```

```
      begin
         return Datos;
      end Lee_Datos ;
      ---------------------------
      procedure Cambia_Entrada  (Entrada : in Float)  is
      begin
         Datos.R  := Entrada;
      end Cambia_Entrada;
      ------------------------------
      procedure  Actualiza_Datos(IXs,IY,IYs : in Vector;Iu : in Float)  is
      begin

         Datos.Xs :=Ixs ;
         Datos.Y  :=Iy  ;
         Datos.Ys :=Iys ;
         Datos.U  :=Iu  ;

      end  Actualiza_Datos;
      ------------------------------
   end Sistema;
   ----------------------
end Protegido;
```

4.2.3 PAQUETE PENDULUM.

Este paquete es el que contiene las tareas que utilizarán tanto al paquete Protegido como al Dispositivos. Los nombres de las tareas son *Control*, *Operario* y *Display*. Operario es una tarea esporádica, ya que depende de la introducción de datos por teclado, pero tanto ella como Display tienen una frecuencia de activación fija, mientras que la más importante, Control, se activa con los muestreos de la tarjeta conversora. Seguidamente paso a detallar en pseudocódigo su funcionalidad:

Operario

Lee entrada de teclado
Si entrada = 99 → terminar aplicación
Si no → cambiar referencia
Se bloquea hasta próxima activación.

Control

Lee_Datos del objeto protegido
Lee_Posicion sistema (aquí se queda bloqueado hasta que llegue un nuevo muestreo)
Lee_Angulo sistema
Calcula salida a aplicar
Aplica la salida.
Calcula las próximas variables de estado.
Actualiza datos en el objeto protegido.

Display

Lee_Datos del objeto protegido
Llama a Muestra_Datos del paquete dispositivo.
Se bloquea hasta próxima activación.

Los paquetes externos utilizados aquí son los siguientes:
[a] Ada.Calendar : Para tomar las medidas de tiempos necesarias para que las tareas se activasen con una periodicidad fija.
[b] Ada.Exceptions : Para control de errores y excepciones.

[c] Matrices : Contiene funciones básicas para realizar operaciones con matrices.
[d] Time_Measurements : Necesario al tomar las medidas de los tiempos de ejecución de las tareas y sus bloqueos para el análisis de planificabilidad.
[e] Full_Console_Management : Para poner algún mensaje por pantalla, junto con Ada.Text_Io, Integer_Text_Io y Float_Text_Io.

Se puede ver una descripción más completa de algunos de estos paquetes en la sección 4.1.

En la especificación se definen tanto las prioridades de las tareas como sus periodos de activación. La tarea Inicio se pone a la prioridad más alta, ya que será la que realice el proceso de inicialización del sistema completo para seguidamente, tras una orden del teclado, activar el resto de las tareas.

```
with Ada.Calendar           ; use Ada.Calendar;
with Ada.Exceptions         ; use Ada.Exceptions;
with Ada.Text_Io            ; use Ada.Text_Io   ;
with Ada.Integer_Text_Io    ; use Ada.Integer_Text_Io;
with Ada.Float_Text_Io      ; use Ada.Float_Text_Io;

with Driver_Marte   ; use Driver_Marte;

with Dispositivo    ; use Dispositivo;
pragma Elaborate_All(Dispositivo);
with Protegido      ; use Protegido;

with Matrices       ; use Matrices;
pragma Elaborate_All(Matrices);
with Paquete        ; use Paquete;

-- with Time_Measurements; use  Time_Measurements;
-- pragma Elaborate_All( Time_Measurements  );

with Full_Console_Management ; use Full_Console_Management;
pragma Elaborate_All(Full_Console_Management);

pragma Elaborate_All(Protegido);


package Pendulum  is
   P_I:Integer:=10;

   P_C:Integer:=8;

   T_Display  : Duration:=0.500 ;
   P_D:Integer:=6;

   T_Operario : Duration:=0.400;
   P_O:Integer:=4;


   -----------------------
   task   Inicio is
      pragma Priority(P_I);
   end  Inicio;

   task type  Control is
      pragma Priority(P_C);
   end Control;

   task type  Operario is
```

```
      pragma Priority(P_O);
   end Operario;

   task type  Display is
      pragma Priority(P_D);
   end Display;
   -----------------------
end Pendulum;
```

El cuerpo del paquete es el siguiente:

```
package body  Pendulum  is
   package Dur_Io is new Ada.Text_Io.Fixed_Io(Duration);
   use Dur_Io;

   ---------------------------------------
   task body Inicio  is
      Datoi : Float ;
   begin

      Datoi:= Lee_Operador;

      while Datoi /= 0.0  loop

         Accesos.Aplica_Fuerza(0.0);

         Datoi:=Lee_Operador;
      end loop;

      declare
         A: Control;
         B: Operario;
         C: Display;
      begin
         null;
      end;

   exception
      when Event: others =>
         Put(Exception_Name(Event));
         Put(Exception_Name(Exception_Identity(Event)));
         Put(Exception_Message(Event));
   end  Inicio;
   ----------------------------------------
   task body Operario  is
      Entrada : Float;
      Last_Start:Time;
      Salida: exception;
   begin
      Last_Start:=Clock;
      loop
         ----
         Entrada:=Lee_Operador;
         ----
         if Integer(Entrada)=99 then
            raise Salida;
         end if;
         ----
         Sistema.Cambia_Entrada(Entrada/10.0);
         ----
         Last_Start:=Last_Start + T_Operario;
```

```
      delay until (Last_Start);
   end loop;
   -------
exception
   when Salida=>
      Put("Finalizando Aplicacion");
      abort Control, Display;--, Operario;
   when Event: others =>
      Put(Exception_Name(Event));
      Put(Exception_Name(Exception_Identity(Event)));
      Put(Exception_Message(Event));
end  Operario;
----------------------------------------
task body Control is

   -- 100 Hz --- motor ----------------
   Nbar : Float := -2.0600 ;
   G:Vector(1..4):=(0.0005,0.0905,0.0011,0.2264 );
   F:Matrix(1..4,1..4):=((1.0000,0.0100,0.0001,0.0000),
                         (0.0,   0.9909,0.0266,0.0001),
                         (0.0,  -0.0001,1.0016,0.0100),
                         (0.0,  -0.0226,0.3117,1.0016));
   K:Vector(1..4):=(-2.0626,-1.6280,6.9029,1.3342);
   L:Matrix(1..4,1..2):=((  2.6208, -0.0105),
                         (171.3022, -1.3581),
                         ( -0.0318,  2.6333),
                         ( -5.2021,173.5720));


   U : Float    :=0.0;
   H:Matrix(1..2,1..4):=((1.0, 0.0, 0.0, 0.0),
                         (0.0, 0.0, 1.0, 0.0));
   Xs:Vector(1..4)  :=(others=>0.0);
   Y,Ys:Vector(1..2):=(others=>0.0);
   D : Datos_T;
begin
   loop
      ----
      D :=Sistema.Lee_Datos;

      Y(1) :=Float(Accesos.Lee_Posicion);

      Y(2) :=Float(Lee_Angulo);

      U :=D.R * Nbar -(D.Xs * K);     -- entrada

      Accesos.Aplica_Fuerza(U);

      Xs:=F * D.Xs +G * U +L*(Y-D.Ys);-- observer

      Ys:=H * Xs;                     -- observer

      Sistema.Actualiza_Datos(Xs,Y,Ys,U);

      ----
   end loop;
exception
   when Event: others =>
      Put(Exception_Name(Event));
      Put(Exception_Name(Exception_Identity(Event)));
      Put(Exception_Message(Event));
      Put("Pendulo Caido");
      abort Display,Operario;--,Control;
```

```
   end  Control;
   -----------------------------------------------
   task body Display is
      Last_Start:Time;
      P1,P2,P3,P4,P5,P6,P7,P8,P9,Media : Float :=0.0;
      D :Datos_T;
      F_Max : Float := 0.0;
   begin
      Last_Start:=Clock;
      loop
         ----
         D:=Sistema.Lee_Datos;
         Muestra_Datos(D.Y(1),D.Y(2),D.U);
         ----

         --- Para mostrar la media.
         P9:= P8;
         P8:= P7;
         P7:= P6;
         P6:= P5;
         P5:= P4;
         P4:= P3;
         P3:= P2;
         P2:= P1;
         P1:= D.Y(1);
         Media:= (P1+P2+P3+P4+P5+P6+P7+P8+P9)/9.0;
         Set_Cursor(6,32); Put("Media:");Put(Media,3,2,0);
         ---
         if abs(D.U) > F_Max then
            F_Max :=abs(D.U) ;
         end if;
         Set_Cursor(8,32); Put("F_max:");Put(F_Max,3,1,0);
         ---
         Last_Start:=Last_Start + T_Display;

         delay until (Last_Start);
      end loop;
   exception
      when Event: others =>
         Put(Exception_Name(Event));
         Put(Exception_Name(Exception_Identity(Event)));
         Put(Exception_Message(Event));
   end  Display;
   -----------------------------------------------
end Pendulum;
```

4.3 IMPLEMENTACIÓN DEL SIMULADOR DEL SISTEMA FÍSICO.

Su estructura es muy similar al 'Control', por lo que sólo se explicarán las diferencias más importantes.

4.3.1 PAQUETE DISPOSITIVOS

Este paquete implementa 5 funciones: una para imprimir datos por pantalla, otra para obtener entradas desde teclado y tres para operar sobre el driver de la tarjeta A/D D/A.

Las diferencias con Control estriban en las funciones de lectura / escritura en el driver. *Lee_Angulo* se cambia por *Escribe_Angulo*, *Lee_Posicion* por *Escribe_Posicion* y *Aplica_Fuerza* por *Lee_Fuerza*.

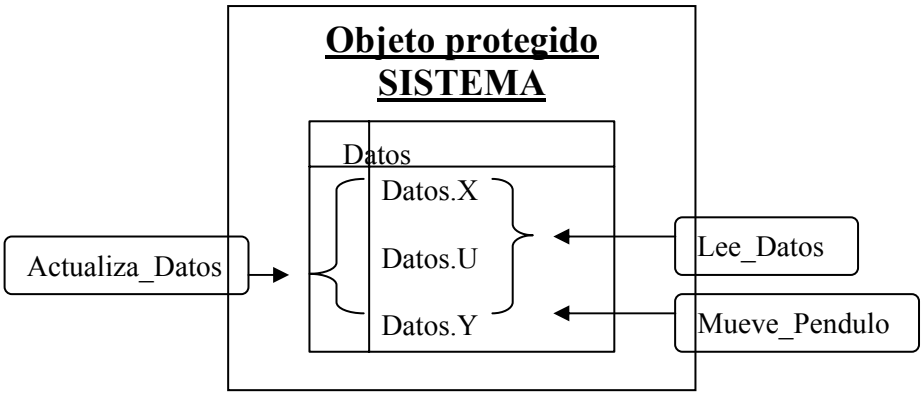
4.3.2 PAQUETE PROTEGIDO

Este paquete define y contiene la estructura de datos que albergará los principales datos del programa y proporcionará mecanismos para el acceso y manipulación de los mismos de forma exclusiva.

Respecto al control, los datos almacenados varían ligeramente, ya que el sistema sólo debe almacenar su salida Y, sus variables de estado X y la entrada de control U.

En cuanto a las operaciones sobre el objeto protegido, destaca el cambio de *Cambia_Entrada* por *Mueve_Pendulo* que se encargará de provocar una 'perturbación' sobre el sistema moviendo el péndulo adrede un número de grados hacia uno u otro lado.

La estructura del objeto protegido, los datos que almacena y las funciones que acceden a él, son las siguientes:



La función `Lee_Datos` es utilizada por las tareas *Control* y *Display*, el procedimiento `Mueve_Pendulo` por la tarea *Operario* y el procedimiento `Actualiza_Datos` por la tarea *Control*.

4.3.3 PAQUETE PENDULUM

Contiene al igual que la aplicación de Control, tres tareas llamadas Control, Operario y Display. Display realiza un trabajo idéntico, mientras que Operario se encarga de leer el teclado no para modificar la referencia sino para introducir perturbaciones al sistema. La tarea Control se encarga de leer a través del driver la señal de control y, en función de su valor y a través de las ecuaciones que definen el sistema físico, mover el péndulo y el carro y trasladar esos cambios al exterior a través de la tarjeta. A continuación, se muestra el pseudocódigo y el código de las tareas Operario y Control:

Control

Lee_Datos del objeto protegido
Lee_Fuerza (aquí se queda bloqueado hasta que llegue un nuevo muestreo)
Calcula respuesta del sistema físico.
Si posicion > + 0.5 m → posicion = + 0.5 m
Si posicion < - 0.5 m → posicion = - 0.5 m
Escribe_Posicion en el driver.
Escribe_Angulo en el driver.
Actualiza datos en el objeto protegido

Operario

Lee entrada de teclado
Si entrada = 99 → terminar aplicación
Si no → mover pendulo
Se bloquea hasta próxima activación.

```
-----------------------------------------
task body Control is

   -- 100 Hz ------------------
   G:Vector(1..4):=(  0.0005,0.0905,0.0011,0.2264 );
   F:Matrix(1..4,1..4):=((1.0000,0.0100,0.0001,0.0000  ),
                         (0.0,   0.9909,0.0266,0.0001  ),
                         (0.0,  -0.0001,1.0016,0.0100  ),
                         (0.0,  -0.0226,0.3117,1.0016  ));

   U : Float    :=0.0;
   H:Matrix(1..2,1..4):=((1.0,0.0,0.0,0.0),
                         (0.0,0.0,1.0,0.0));
   X:Vector(1..4):=(others=>0.0);
   Y:Vector(1..2):=(others=>0.0);
   D : Datos_T;
   Last_Start:Time;
begin

   loop
      ----
      D :=Sistema.Lee_Datos;
      U :=Float(Accesos.Lee_Fuerza);

      X :=F * D.X  +G * U;  -- planta
      Y :=H * X;            -- planta
      ----
      if (Y(1)>0.5) then
         Y(1):=0.5;
      end if;
      if (Y(1)<-0.5) then
         Y(1):=-0.5;
      end if;
      ----
      Accesos.Escribe_Posicion(Posicion_T(Y(1)));
      Escribe_Angulo(Y(2));
      Sistema.Actualiza_Datos(X,Y,U);
      ----
   end loop;
exception
   when Event: others =>
      Put(Exception_Name(Event));
      Put(Exception_Name(Exception_Identity(Event)));
      Put(Exception_Message(Event));
      Put("Pendulo Caido");
      abort Display,Operario;
      exit;
end  Control;
-----------------------------------------------

--------------------------------------
task body Operario  is
   Entrada : Float;
   Last_Start:Time;
   Salida: exception;
begin
   Last_Start:=Clock;
   loop
      ----
      Entrada:=Lee_Operador; --Aqui se queda bloqueado.
```

```
      if Integer(Entrada)=99 then
         raise Salida;
      end if;
      Sistema.Mueve_Pendulo(Entrada);
      ----
      Last_Start:=Last_Start + T_Operario;
      delay until (Last_Start);
   end loop;
exception
   when Salida=>
      Put("Finalizando Aplicacion");
      abort Control, Display;
   when Event: others =>
      Put(Exception_Name(Event));
      Put(Exception_Name(Exception_Identity(Event)));
      Put(Exception_Message(Event));
end  Operario;
-----------------------------------------
```

5. DRIVER PARA LA TARJETA A/D D/A

5.1 INTRODUCCIÓN

Un driver para un dispositivo es una interfaz entre el sistema operativo y el hardware del dispositivo. Los drivers forman parte del núcleo del sistema operativo y tienen acceso restringido a las estructuras del sistema operativo. El objetivo de un driver debe ser flexibilizar el uso de los dispositivos proporcionando un mecanismo común de uso para todas las aplicaciones.

El driver de la tarjeta conversora A/D D/A se desarrolló en un primer momento en Linux por que la forma de construcción de drivers es muy similar a la estructura prevista para MaRTE O.S. y porque resulta un entorno de desarrollo más cómodo.

5.2 TARJETA CONVERSORA A/D D/A

Para tomar medidas de la posición y el ángulo del sistema, se necesitaba convertir voltajes en datos. Para ello, se tomó una tarjeta conversora A/D D/A (modelo AX5411 de AXIOM) [30]. Sus características más importantes son las siguientes:

- Posee 16 canales analógicos de entrada y 2 de salida.
- Resolución: 12 bits
- Frecuencia máxima de muestreo: 60 Khz
- Frecuencia máxima de salida: 33 Khz
- Rangos de entrada: +/- 5 V
- Rangos de salida: [0, +5] V

La tarjeta está dotada de los recursos hardware necesarios para que las operaciones de entrada / salida puedan ser controladas mediante programa, mediante procedimientos de gestión de interrupciones y mediante canales de DMA (acceso directo a memoria). La tarjeta está dotada de un temporizador propio que permite temporizar localmente los procesos de adquisición y suministro de datos. Ofrece 16 registros mapeados sobre el bus I/O, a través de los que se accede a todos sus recursos hardware.

5.3 DRIVER EN LINUX

5.3.1 FORMATO DE DRIVERS EN LINUX

En Linux el concepto de fichero se utiliza como marco genérico para implementar muchos de los objetos internos del sistema operativo, es decir, casi todo puede ser tratado como un fichero. El núcleo construye un sistema de ficheros estructurado encima del hardware desestructurado, para lo cual soporta varios tipos de ficheros. Las operaciones de control de los dispositivos se realizan mediante drivers de dispositivos, representados mediante ficheros especiales; así cada dispositivo presente en el sistema debe tener su driver. Cada segmento de código que se añade al núcleo se denomina módulo (module), y el propio núcleo ofrece servicios para la instalación y desinstalación de módulos, aun en tiempo de ejecución [10].

Linux distingue tres tipos de dispositivos: de caracteres, de bloques y de red. Nuestro dispositivo, por sus características, pertenece al primer grupo, dispositivo de caracteres. El acceso a estos dispositivos se realiza a través de nodos (nodes) del sistema de ficheros y se utiliza como si se tratara de un fichero, con llamadas al sistema como open, close, read y write. La diferencia fundamental con los ficheros normales es que en ellos te puedes desplazar hacia delante y hacia atrás, mientras que los drivers son normalmente canales de datos a los que se accede secuencialmente.

Los diferentes tipos de drivers serán tratados al igual que otro tipo de entidad en Linux llamado módulo (module). El código de un módulo sólo debe llamar a funciones incluidas en el núcleo y no en librerías, ya que el módulo se enlaza directamente con el núcleo. El módulo ejecuta sus instrucciones en el espacio del núcleo mientras que las aplicaciones lo hacen en el de usuario, con diferentes privilegios y con espacios de direcciones diferentes.

Linux cambia del espacio de usuario al del núcleo cuando se produce una llamada al sistema o cuando se atiende una interrupción del hardware.

La ejecución de una llamada al sistema se ejecuta en el contexto del proceso que llama y puede acceder a su espacio de direcciones. Una excepción a esta norma son los manejadores de interrupción, que no ejecutan en el contexto de ningún proceso en particular.

En Linux, drivers y dispositivos se identifican mediante una serie de números de identificación llamados números mayores (major numbers) y números menores (minor numbers). Un dispositivo mayor se corresponde con una unidad funcional separada que a menudo dispone de sus propios registros de control, direcciones de I/O, y vector de interrupción. En cambio, un dispositivo menor identifica subunidades o subfunciones de un dispositivo mayor. Los dispositivos menores de un mismo dispositivo mayor, normalmente, comparten algunos recursos como los registros de control y el vector de interrupción. Los números menores de un dispositivo también se pueden usar para seleccionar diferentes modos de operación o funciones del dispositivo.

5.3.2 DESARROLLO

Estudiando el manual del dispositivo y definiendo una política de adquisición de datos se elaboró un driver. Este driver constaba de las funciones básicas que abstraerían el funcionamiento del dispositivo de forma que su uso fuese idéntico al de un fichero. Dichas funciones eran las siguientes:

- init_module : instala el módulo e inicializa los registros de la tarjeta.
- cleanup_module : desinstala el módulo.
- open_tarjeta y release_tarjeta : arranca / paraliza el ciclo de muestreo abriendo o cerrando el fichero de dispositivo de la tarjeta.
- read_tarjeta y write_tarjeta : lee / escribe en los canales de la tarjeta mediante la lectura / escritura en el fichero de dispositivo.

Init_module:

El punto de entrada *init_module* debe preparar el módulo para la posterior ejecución del resto de las funciones o puntos de entrada al mismo. Es la función que se ejecuta cada vez que el módulo se instala en el sistema, adosándose al núcleo de Linux. En él se hacen las siguientes acciones:

- Se comprueba la capacidad para registrar el módulo con el número mayor elegido. Hay que decir que en Linux este módulo es de tipo carácter, al que se le dio el número mayor 120, y número menor 0 y 1, uno para cada canal de lectura y escritura en la tarjeta.
- Se testea el hardware de la tarjeta para saber si está bien conectada y la comunicación con la misma es correcta.
- Se inicializan las variables de las estructuras en las que se registran los datos almacenados, tanto para el canal 0 como para el 1.
- Se instala el manejador de interrupción solicitando la IRQ = 5 que por omisión esta asignada al puerto paralelo 2, determinando además cual es la función manejadora de la interrupción.
- Se establece en la tarjeta cuál es el canal inicial y final del ciclo de muestreo del multiplexor, así como la ganancia del amplificador de entrada y el periodo de muestreo.
- Se programa la tarjeta en modo de disparo periódico, con reloj interno y generación de interrupción.

Cleanup_module:

El punto de entrada *cleanup_module* es el que se invoca en la desinstalación del módulo y debe eliminar todo rastro del mismo. Básicamente realiza estas tres acciones:
- Desprograma en la tarjeta la realización de muestreos periódicos y la generación de interrupciones.
- Libera el nivel de interrupción que reservamos en el sistema.
- Desregistra el dispositivo con el número mayor 120.

Open_tarjeta:

Este punto de entrada es el que se ejecuta cuando desde una aplicación cualquiera se abre el fichero de dispositivo. Éste enmascara las funciones de utilización de la tarjeta como si fueran las mismas que cualquier fichero. Sus acciones son:
- Arranca el reloj de la tarjeta que generara las interrupciones periódicas.
- Incrementa el contador de uso del módulo.

El sistema guarda un contador de utilización para cada módulo de manera que se pueda eliminar el módulo de forma segura.

Release_tarjeta:

Operación que se invoca cuando se cierra el nodo, y por tanto, la estructura de fichero. Esta función es llamada por el sistema cuando se cierra el fichero de dispositivo, ya sea adrede desde una aplicación o de forma forzosa cuando una aplicación que tenia abierto dicho fichero termina. Sus acciones son:
- Paraliza la ejecución de muestreos periódicos y da la orden a la tarjeta de que retire la petición de interrupción, por si se hubiese producido alguna.
- Decrementa el contador de uso del módulo.

Read_tarjeta:

Localiza el número menor del fichero desde el que se ha llamado a la función y en función de cuál sea:

0 →

Espera en la cola de lectura hasta que llegue una interrupción para el canal 1.
Coge el último dato muestreado por este canal y lo pasa al espacio de usuario.
Retorna el número de datos (en bytes) devuelto.

1 →

Coge el último dato muestreado por este canal y lo pasa al espacio de usuario.
Retorna el número de datos (en bytes) devuelto.

Write_tarjeta:

- Realiza las transformaciones necesarias del dato adquirido para introducirlo en los registros de la tarjeta.
- Introduce los datos en los registros del canal adecuado en función de cuál sea el número menor.
- Retorna el número de bytes escritos.

Funcion_manejadora:

Un manejador de interrupción no ejecuta en el contexto de ningún proceso en particular, por lo que no puede transferir datos desde o hacia el espacio de usuario. Realiza las siguientes acciones:

- Coge el dato muestreado del canal de entrada analógico.
- Obtiene el canal del que viene y su valor.
- Lo deposita en el buffer adecuado al canal que corresponda para que pueda ser recogido por la orden de read_tarjeta.
- Si el canal es el 0, despierta los procesos en la cola de espera.
- Da orden a la tarjeta de interrupción atendida.

El código del driver para Linux es el siguiente:

```
// ====================================================
//    DRIVER PARA EL MANEJO DE LA TARJETA AX5411 DE AXIOM
//      fichero:  drv_tarjeta.c
//    Ruben Miguelez Garcia @ 2001/2002
// ====================================================

#include <linux/config.h>

#include <linux/module.h>// para MOD_,,,_USE_COUNT
#include <linux/version.h>

#include <linux/types.h>
#include <linux/fs.h>
#include <linux/mm.h>
#include <asm/uaccess.h> // para usar 'put_user' y 'copy_from_user'
#include <linux/errno.h> // para usar 'EBUSY'
#include <asm/segment.h> // para usar 'put_user'

#include <linux/sched.h>
#include <asm/io.h>      // para out_p y inb_p
#include <linux/wait.h>  // para read/write bloqueante
//-------------------------------------------
//-------------------------------------------
#define DRIVER    "tarjeta"
#define PRU_MAJOR 120
#define MAJORX    MAJOR(file->f_dentry->d_inode->i_rdev)
#define MINORX    MINOR(file->f_dentry->d_inode->i_rdev)
#define MINORI    MINOR(inode->i_rdev)
#define MAJORI    MAJOR(inode->i_rdev)

//-------------------------------------------
// ------ CONSTANTES DE LA TARJETA ---------------
#define BASE      0x300    // en /proc/ioports vemos q esta libre
#define ADL       BASE+0
#define ASTA      BASE+0
#define ADH       BASE+1
#define CGA       BASE+1
#define MUC       BASE+2
#define DIN       BASE+3
#define DOUT      BASE+3
#define DAL0      BASE+4
#define DAH0      BASE+5
#define DAL1      BASE+6
#define DAH1      BASE+7
#define STAT      BASE+8
#define CLI       BASE+8
#define CNTR      BASE+9
#define DIL       BASE+10
#define DOL       BASE+10
#define DIH       BASE+11
#define DOH       BASE+11
#define CONT0     BASE+12
#define CONT1     BASE+13
#define CONT2     BASE+14
#define CCTR      BASE+15
#define CSTA      BASE+15
//----- ------------------------------
//---- PARAMETROS INICIALIZACION TARJETA------
#define TARJET_IRQ  0x50 // 5    en /proc/interrupts vemos q no se usa
```

```
#define DRIVER_IRQ     5 //      ojo
#define CANAL_INI   0x00 // 0
#define CANAL_FIN   0x01 // 0
#define GANANCIA    0x01 // 1
#define FREQ_MUESTREO 100  // Hhz
//#define D_CONT1   0x07a8 //10Hz=f*1a0a,100Hz=f*29a,500Hz=f*85,1000Hz=f*42
//#define D_CONT2   0x00ff //5KHz=f*d,10KHz=f*6,1Hz=ff*f51,2Hz=ff*7a8,
#define INT_ON      0x80      //4Hz=ff*3d4
#define INT_OFF     0x00
#define CONT_ON     0x03
#define CONT_OFF    0x00

// ----- VAR'S GLOBALES ------
#define SSHORT sizeof(short)
#define TAM   2

short circular[TAM];
short circular1[TAM];
int r,w;
int r1,w1;


DECLARE_WAIT_QUEUE_HEAD(rq);


//--------- READ_TARJETA ----------
static ssize_t read_tarjeta(struct file * file,char * buf, size_t count,
loff_t *offp){


  if (MINORX==0){
    interruptible_sleep_on(&rq);
    copy_to_user(buf,circular+r,2);
  }
  else
    copy_to_user(buf,circular1+r1,2);
  return(2);
}

//--------- WRITE_TARJETA  ---------
static ssize_t write_tarjeta(struct file * file,
                      const char * buf, size_t count, loff_t *offp){
  char alto,bajo;
  short wtemp;

  copy_from_user(&wtemp,buf,2);
  alto=(char) (wtemp>>4);
  bajo=(char) ((wtemp & 0x000f)<<4);

  if(MINORX==0){
    outb( bajo ,DAL0);  //byte bajo OJO!!PRIMERO EL BAJO
    outb( alto ,DAH0);  //byte alto
  }
  else {
    outb( bajo ,DAL1);  //byte bajo OJO!!PRIMERO EL BAJO
    outb( alto ,DAH1);  //byte alto
  }

  return (2);
}

//-------- OPEN_TARJETA -------------
static int open_tarjeta(struct inode *inode, struct file * file){
```

```
  outb_p(INT_ON | TARJET_IRQ | CONT_ON, CNTR);    // CONT_ON
  MOD_INC_USE_COUNT;
  return 0;
}

//-------- RELEASE_TARJETA ----------
static int release_tarjeta(struct inode *inode, struct file * file){

  outb_p(INT_ON | TARJET_IRQ | CONT_OFF, CNTR);   // CONT_OFF
  outb_p(0x01,CLI);                               // CLI
  MOD_DEC_USE_COUNT;
  return 0;
}

//----INICIALIZAR FILE_OPERATIONS---
static struct file_operations prueba_fops = {
  read:    read_tarjeta,
  write:   write_tarjeta,
  open:    open_tarjeta,
  release: release_tarjeta
};

//--------FUNCION_MANEJADORA--------
void funcion_manejadora (int irq,void *dev_id,struct pt_regs *regs){
  int   ptemp,ptemp1,i;
  short din;
  char  canal;
  canal=(inb(ADL) & 0x0f);
  din=( (inb(ADH)<<4) | (inb(ADL))>>4 );//ADL $ D*  , ADH $ DD

  if (canal==0){
    circular[w]=din;    ptemp=w;    w=r;    r=ptemp;

    if (CANAL_FIN==1){
      // Provocamos el muestreo del canal 1 por soft
      outb(1,ASTA );
      // Esperamos EOC
      while ( (inb(STAT) & 0x80 ) != 0x00 )
        i=i+1;
      // Recoger dato
      din=( (inb(ADH)<<4) | (inb(ADL))>>4 );
      circular1[w1]=din;    ptemp1=w1;    w1=r1;    r1=ptemp1;
    }
    wake_up_interruptible(&rq);
  }
  else {
    circular1[w1]=din;    ptemp1=w1;    w1=r1;    r1=ptemp1;
  }
  outb(1,CLI); // int atendida
}

//-------- INIT_MODULE -------------
int init_module(void){

int D_CONT1,D_CONT2;

  //--- Número mayor
  if (register_chrdev(PRU_MAJOR, "tarjeta", &prueba_fops)) {
    printk("ERROR: no se puede usar %d como numero mayor!!\n", PRU_MAJOR);
    return -EIO;
  };

  //---  CHECK del Hardware
```

```
  outb_p(0x50,CNTR);
  inb_p (CNTR);
  if((inb(CNTR)) != 0x50 ){
    printk("-- ERROR -- FALLO HARDWARE --\n");
    return(-1);
  }
  //--- Inicializar buffer circular
  r =0;w =1;
  r1=0;w1=1;

  //-- Instalacion manejador de interrupcion

if(request_irq(DRIVER_IRQ,funcion_manejadora,SA_INTERRUPT,"int_tarjet",0)){
    printk("No va el request_irq\n");
    return(-1);
  }
  else
    printk("Interrupcion Habilitada\n");

  //--- Inicializar tarjeta
  outb_p(1,CLI);                        // CLI
  outb_p((char)((CANAL_FIN<<4) | CANAL_INI), MUC); // SET_CH
  outb_p(GANANCIA, CGA);                // SET_GAIN
  outb_p(0x54,CCTR);                    // SET_TIMER
  D_CONT1=1000000/FREQ_MUESTREO;
  D_CONT2=1;
  while ( D_CONT1 > 65535 ){
      D_CONT1 = D_CONT1 / 10;
      D_CONT2 = D_CONT2 * 10;
      }
  outb_p((char)(D_CONT1 & 0x00ff),CONT1);//bajo
  outb_p(0x64,CCTR);
  outb_p((char)(D_CONT1 >> 8    ),CONT1);//alto
  outb_p(0xa4,CCTR);
  outb_p((char)(D_CONT2 >> 8    ),CONT2);//alto
  outb_p(0x94,CCTR);
  outb_p((char)(D_CONT2 & 0x00ff),CONT2);//bajo

  outb_p(INT_ON | TARJET_IRQ | CONT_OFF   ,CNTR );
  //int + irq + reloj interno + no dma + trigger por contador(no activo)
  return(0);
}
//--------- CLEANUP_MODULE ----
void cleanup_module(void){
  // Deshabilitar int's y contador
  outb_p(INT_OFF | TARJET_IRQ | CONT_OFF   ,CNTR );
  outb_p(1,CLI);
  // Liberamos interrupcion
  free_irq(DRIVER_IRQ,0); // Liberamos interrupcion
  // Desintalo módulo
  unregister_chrdev(PRU_MAJOR, "tarjeta");
}
```

5.4 DRIVER EN MARTE O.S.

Como hemos dicho anteriormente, el driver quedó configurado en Linux bajo el número mayor 120 y menores 0 y 1 (canal 0 y 1). Sus ficheros asociados eran /dev/fichero_dispositivo y /dev/fichero_dispositivo1 .

Posteriormente, a la hora de trasladar toda la aplicación a MaRTE O.S., como en este sistema operativo todavía no estaba implementada la funcionalidad de los ficheros de dispositivo se tuvo que

hacer un parche al driver para que funcionase en MaRTE O.S.. Lo primero que se tuvo que hacer fue transformar todo el driver de Linux en simples funciones de C. También se tuvieron que realizar algunas modificaciones como cambiar las instrucciones printk por printf y los parámetros de las sentencias outb, así como la forma de implementar la cola de espera en el read_tarjeta entre otros. Seguidamente, se construyó un paquete Ada que se encargaría de importar a Ada las funciones C.

```
package      Driver_Marte is
      --------------------------------
         type Short is mod 4096;
         for  Short'Size use 16;
      --------------------------------
      function Init_Module (Freq_Muestreo : in Integer)  return Integer ;
      pragma Import (C, Init_Module,"init_module");
      --------------------------------
      procedure Cleanup_Module;
      pragma Import (C, Cleanup_Module,"cleanup_module");
      --------------------------------
      procedure Open_Tarjeta;
      pragma Import (C,Open_Tarjeta,"open_tarjeta");
      --------------------------------
      procedure Release_Tarjeta;
      pragma Import (C,Release_Tarjeta,"release_tarjeta");
      --------------------------------
      function Read_Short (Minor :in Integer) return Short;
      pragma Import (C,Read_Short,"read_short" );
      --------------------------------
      procedure Write_Short(Minor :in Integer;Dato : in Short);
      pragma Import (C,Write_Short,"write_short");
      --------------------------------
end Driver_Marte;
```

6. ANÁLISIS DE PLANIFICABILIDAD

6.1 INTRODUCCIÓN

En los sistemas de tiempo real es muy difícil comprobar mediante pruebas si el diseño es completamente válido desde el punto de vista práctico, ya que los programas se ejecutan en la mayoría de los casos de forma concurrente y para que aparezca un posible error tienen que coincidir muchas circunstancias.

El test de planificabilidad es un conjunto de condiciones matemáticas que, si se cumplen, garantizan que el conjunto de tareas no violará ninguno de sus requerimientos temporales.

El algoritmo de planificación utilizado para las tareas que ejecutan concurrentemente es el RM (Rate Monotonic). En el cual a cada tarea se le asigna una prioridad fija, proporcional a la frecuencia con que se ejecuta (es decir, inversamente proporcional a su periodo Ti). En tiempo de ejecución, se selecciona siempre de entre todas las tareas que estén listas, la de mayor prioridad.

Antes de proceder con los cálculos conviene introducir algunas notaciones:
-Ci = Tiempo de ejecución para la tarea_i
-Ti = Periodo de la tarea_i
-Pi = Prioridad de la tarea_i
-Di = Plazo (Deadline) de la tarea_i
-Ri = Tiempo de respuesta de la tarea_i
-Bi = Tiempo que la tarea_i está produciendo un bloqueo a las tareas de mayor prioridad.
-Cs = Tiempo de cambio de contexto (propio del S.O. y del procesador).
-Csint = Tiempo de cambio de contexto de interrupción (propio del S.O. y del procesador).

Cs = 2.5 us para un Pentium II a 200Mhz con MaRTE O.S.
Csint = 1.5 us para un Pentium II a 200Mhz con MaRTE O.S.

Este planificador, de entre los que utilizan prioridades fijas, es óptimo si los periodos coinciden con los plazos máximos de finalización (Ti = Di). Existe una condición necesaria, pero no suficiente, que debería cumplir un conjunto de tareas para ser planificable bajo RM. Esta condición dice que el factor de utilización debe ser: $U <= n\,(2^{(1/n)}-1)$ siendo n el número de tareas [18].

Existe además, una condición necesaria y suficiente para la planificabilidad. Para verificarla hay que calcular iterativamente el tiempo de finalización de cada tarea (Ri). Si sucede que el factor de utilización es menor del 100%, la resolución iterativa de Ri converge, y entonces la condición necesaria y suficiente consiste en comprobar si Ri <= Di, en cuyo caso significa que la tarea Ti cumple con su plazo [18]. Su fórmula es:

$$W_i(t) = \lceil t/T_1 \rceil \cdot C_1 + \ldots + \lceil t/T_{i-1} \rceil \cdot C_{i-1} + C_i$$; Siendo $\lceil x \rceil$ la función techo.

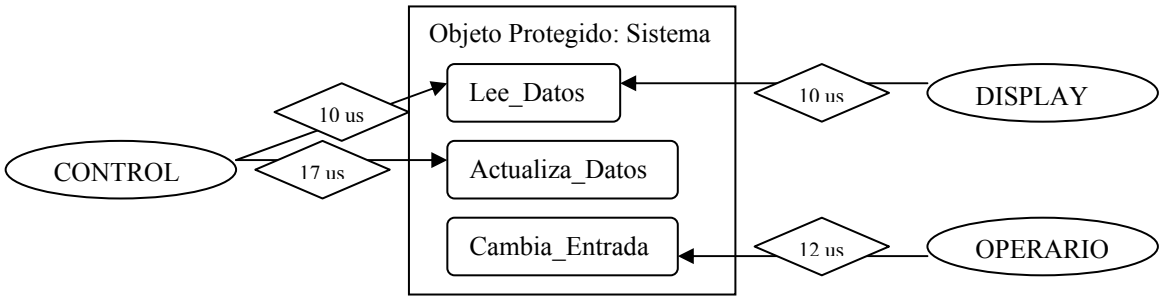
6.2 MAST

MAST son las siglas de "Modeling and Analysis Suite for Real-Time Applications". MAST es un conjunto de herramientas de código fuente abierto ("open source") que permiten modelar aplicaciones de tiempo real y representar análisis temporales de la ejecución de esas aplicaciones. El modelo de MAST se puede utilizar en un ambiente de diseño UML para diseñar aplicaciones de tiempo real, representando todo el comportamiento y requisitos de tiempo real junto con la información del diseño, y permitiendo un análisis automático de planificabilidad [26].

Para utilizar MAST hubo que realizar unos ficheros de entrada (uno para cada aplicación) en los que se introducían los datos relativos a las tareas, objetos compartidos, procesador, etc. Después se ejecutaba MAST con dichos ficheros como entrada y obteníamos la salida. Esta salida nos mostraba información sobre los tiempos de peor caso y los máximos bloqueos entre tareas por inversión de prioridad. Los resultados se compararon con los obtenidos a mano (ya que la aplicación bajo estudio es muy simple), viendo que eran idénticos. Seguidamente, mostramos los ficheros de entrada tanto para la aplicación Control como para Sistema.

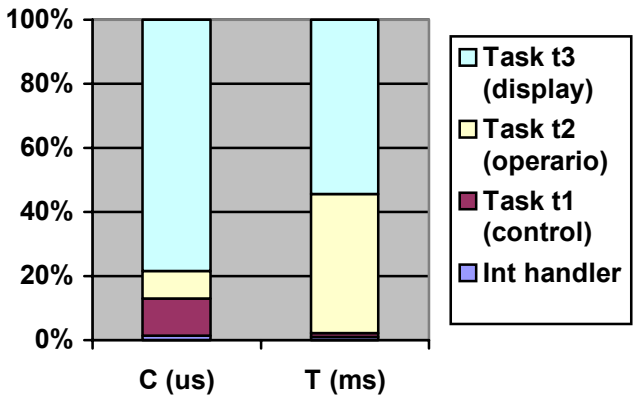
6.3 ANÁLISIS DE PLANIFICABILIDAD: CONTROL

A continuación, mostramos un pequeño dibujo que muestra las tareas y las funciones que utilizan cada una de ellas del objeto protegido y durante cuanto tiempo, lo que puede causar bloqueos entre estas tareas. Aquí no interviene el objeto protegido Accesos, ya que aunque su acceso también se realiza en exclusión mutua, no hay posibilidad de que se produzcan bloqueos entre las tareas porque sólo lo utiliza una de ellas.



Las medidas tomadas para el programa de control son:

	C (μs)	T (ms)
Int handler	22	10
Task t1 (control)	182	10
Task t2 (operario)	132	400
Task t3 (display)	1224	500



Test de utilización (si se cumple, garantiza que el sistema es planificable pero no que no lo es):

F1= (Cint+2Csint)/Ti < U(1),1
F2= (Cint+2Csint)/Ti + (C1 +2Cs)/T1 < U(2),2
F3= (Cint+2Csint)/Ti + (C1 +2Cs)/T1 + (C2+2Cs)/T2 < U(3),3
F4= (Cint+2Csint)/Ti + (C1 +2Cs)/T1 + (C2+2Cs)/T2 + (C3+2Cs)/T3 < U(4),4

Cálculo de tiempos de respuesta de peor caso:

R0=Cint + 2Csint
R1=[t/Ti]*(Cint+2Csint) + (C1+2Cs) + max {B2, B3}
R2=[t/Ti]*(Cint+2Csint) + [t/T1]*(C1+2Cs) + (C2+2Cs) + B3
R3=[t/Ti]*(Cint+2Csint) + [t/T1]*(C1+2Cs) + [t/T2]*(C2+2Cs) + (C3+2Cs)

Seguidamente mostramos el fichero de entrada para MAST con los datos relativos a la aplicación Control:

```
-- Aplicacion del pendulo invertido: Control

-- Processing Resources
Processing_Resource (
      Type              => Fixed_Priority_Processor,
      Name              => Processor_1,
        Worst_Context_Switch  => 2.5,
        Worst_ISR_Switch        => 1.5);

-- Scheduling Servers
Scheduling_Server (
      Type                    => Fixed_Priority,
      Name                    => Int_Handler,
      Server_Sched_Parameters       => (
            Type        => Interrupt_FP_Policy,
            The_Priority            => 100),
      Server_Processing_Resource    => Processor_1);

Scheduling_Server (
      Type                    => Fixed_Priority,
      Name                    => Control,
      Server_Sched_Parameters       => (
            Type        => Fixed_Priority_policy,
            The_Priority            => 10),
      Server_Processing_Resource    => Processor_1);

Scheduling_Server (
      Type                    => Fixed_Priority,
      Name                    => Operario,
      Server_Sched_Parameters       => (
            Type        => Fixed_Priority_policy,
            The_Priority            => 8),
      Server_Processing_Resource    => Processor_1);

Scheduling_Server (
      Type                    => Fixed_Priority,
      Name                    => Display,
      Server_Sched_Parameters       => (
            Type        => Fixed_Priority_policy,
            The_Priority            => 4),
      Server_Processing_Resource    => Processor_1);

-- Resources
```

```
Shared_Resource (
      Type  => Immediate_Ceiling_Resource,
      Name  => Sistema);

-- Operations

-- Critical Sections
Operation (
      Type                    => Simple,
      Name                    => Lee_Datos,
      Worst_Case_Execution_Time    => 10,
            Shared_Resources_List        => (Sistema));

Operation (
      Type                    => Simple,
      Name                    => Actualiza_Datos,
      Worst_Case_Execution_Time    => 17,
            Shared_Resources_List        => (Sistema));

Operation (
      Type                    => Simple,
      Name                    => Cambia_Entrada,
      Worst_Case_Execution_Time    => 12,
            Shared_Resources_List        => (Sistema));

-- Enclosing operations
Operation (
      Type  => Simple,
      Name  => Int_Handler,
      Worst_Case_Execution_Time    => 22);

Operation (
      Type  => Enclosing,
      Name  => Control,
      Worst_Case_Execution_Time    => 182,
      Composite_Operation_List     =>
            (Lee_Datos, Actualiza_Datos));

Operation (
      Type  => Enclosing,
      Name  => Operario,
      Worst_Case_Execution_Time    => 132,
      Composite_Operation_List     =>
            (Cambia_Entrada));

Operation (
      Type  => Enclosing,
      Name  => Display,
      Worst_Case_Execution_Time    => 1224,
      Composite_Operation_List     =>
            (Lee_Datos));

-- Transactions
Transaction (
      Type  => Regular,
      Name  => Int_Handler,
      External_Events => (
            (Type             => Periodic,
             Name             => E1,
             Period     => 10000)),
      Internal_Events => (
            (Type       => regular,
             name       => O1,
```

```
                Timing_Requirements => (
                  Type              => Hard_Global_Deadline,
                  Deadline     => 10000,
                  referenced_event => E1))),
      Event_Handlers => (
            (Type                   => Activity,
             Input_Event            => E1,
             Output_Event           => O1,
             Activity_Operation     => Int_Handler,
             Activity_Server  => Int_Handler)));

Transaction (
      Type  => Regular,
      Name  => Control,
      External_Events => (
            (Type             => Periodic,
             Name             => E2,
             Period     => 10000)),
      Internal_Events => (
            (Type       => regular,
             name       => O2,
                 Timing_Requirements => (
                  Type              => Hard_Global_Deadline,
                  Deadline     => 10000,
                  referenced_event => E2))),
      Event_Handlers => (
            (Type                   => Activity,
             Input_Event            => E2,
             Output_Event           => O2,
             Activity_Operation     => Control,
             Activity_Server  => Control)));

Transaction (
      Type  => Regular,
      Name  => Operario,
      External_Events => (
            (Type             => Periodic,
             Name             => E3,
             Period     => 400000)),
      Internal_Events => (
            (Type       => regular,
             name       => O3,
                 Timing_Requirements => (
                  Type              => Hard_Global_Deadline,
                  Deadline     => 400000,
                  referenced_event => E3))),
      Event_Handlers => (
            (Type                   => Activity,
             Input_Event            => E3,
             Output_Event           => O3,
             Activity_Operation     => Operario,
             Activity_Server  => Operario)));

Transaction (
      Type  => Regular,
      Name  => Display,
      External_Events => (
            (Type             => Periodic,
             Name             => E4,
             Period     => 500000)),
      Internal_Events => (
            (Type       => regular,
             name       => O4,
```

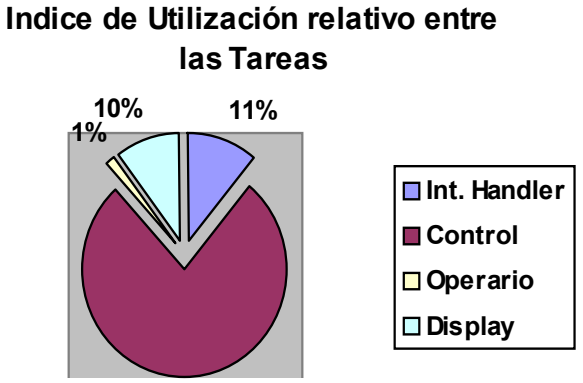
```
            Timing_Requirements => (
              Type                => Hard_Global_Deadline,
              Deadline     => 500000,
              referenced_event => E4))),
    Event_Handlers => (
          (Type                    => Activity,
           Input_Event             => E4,
           Output_Event            => O4,
           Activity_Operation      => Display,
           Activity_Server  => Display)));
```

Los cálculos obtenidos ofrecen los siguientes resultados:

	B (μs)
Int handler	0
Task t1 (control)	0
Task t2 (operario)	12
Task t3 (display)	10

F1= 0.0025 < U(1),1 = 1.000 R0 = 25 us
F2= 0.0212 < U(2),2 = 0.828 R1 = 224 us
F3= 0.0215 < U(3),3 = 0.779 R2 = 359 us
F4= 0.0240 < U(4),4 = 0.756 R3 = 1578 us

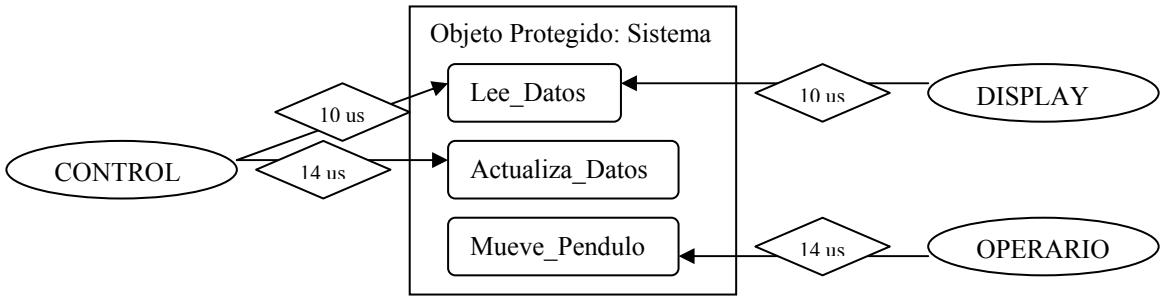
Como es visible, el conjunto de tareas de la aplicación es totalmente planificable.



Como se puede ver en el gráfico, la tarea control es la que mayor índice de utilización tiene. Pero hay que señalar que todas las tareas juntas suman tan solo un índice de utilización del 2.4%

6.4 ANALISIS DE PLANIFICABILIDAD: SISTEMA

Más abajo mostramos un pequeño dibujo que muestra las tareas y las funciones que utilizan cada una de ellas del objeto protegido, lo que puede causar bloqueos entre las mismas.



Las medidas tomadas para el sistema son:

	C (µs)	T (ms)
Int handler	6	10
Task t1 (control)	77	10
Task t2 (operario)	134	400
Task t3 (display)	1213	500

Posteriormente, se realizarán para la aplicación sistema los mismos cálculos que se realizaron para la aplicación control.

Test de utilización (garantiza si el sistema es planificable pero no que no lo es):

F1= (Cint+2Csint)/Ti < U(1),1
F2= (Cint+2Csint)/Ti + (C1 +2Cs)/T1 < U(2),2
F3= (Cint+2Csint)/Ti + (C1 +2Cs)/T1 + (C2+2Cs)/T2 < U(3),3
F4= (Cint+2Csint)/Ti + (C1 +2Cs)/T1 + (C2+2Cs)/T2 + (C3+2Cs)/T3 < U(4),4

Cálculo de tiempos de respuesta de peor caso:

R0=Cint + 2Csint
R1=[t/Ti]*(Cint+2Csint) + (C1+2Cs) + max {B2, B3}
R2=[t/Ti]*(Cint+2Csint) + [t/T1]*(C1+2Cs) + (C2+2Cs) + B3
R3=[t/Ti]*(Cint+2Csint) + [t/T1]*(C1+2Cs) + [t/T2]*(C2+2Cs) + (C3+2Cs)

Seguidamente, mostramos el fichero de entrada para MAST con los datos relativos a la aplicación Control:

```
-- Aplicacion del pendulo invertido: Sistema

-- Processing Resources
Processing_Resource (
      Type              => Fixed_Priority_Processor,
      Name              => Processor_1,
        Worst_Context_Switch  => 2.5,
        Worst_ISR_Switch        => 1.5);

-- Scheduling Servers
Scheduling_Server (
      Type                    => Fixed_Priority,
      Name                    => Int_Handler,
      Server_Sched_Parameters       => (
            Type        => Interrupt_FP_Policy,
            The_Priority            => 100),
      Server_Processing_Resource    => Processor_1);

Scheduling_Server (
```

```
      Type                    => Fixed_Priority,
      Name                    => Control,
      Server_Sched_Parameters       => (
            Type        => Fixed_Priority_policy,
            The_Priority            => 10),
      Server_Processing_Resource    => Processor_1);

Scheduling_Server (
      Type                    => Fixed_Priority,
      Name                    => Operario,
      Server_Sched_Parameters       => (
            Type        => Fixed_Priority_policy,
            The_Priority            => 8),
      Server_Processing_Resource    => Processor_1);

Scheduling_Server (
      Type                    => Fixed_Priority,
      Name                    => Display,
      Server_Sched_Parameters       => (
            Type        => Fixed_Priority_policy,
            The_Priority            => 4),
      Server_Processing_Resource    => Processor_1);

-- Resources
Shared_Resource (
      Type  => Immediate_Ceiling_Resource,
      Name  => Sistema);

-- Operations

-- Critical Sections
Operation (
      Type                    => Simple,
      Name                    => Lee_Datos,
      Worst_Case_Execution_Time     => 10,
            Shared_Resources_List         => (Sistema));

Operation (
      Type                    => Simple,
      Name                    => Actualiza_Datos,
      Worst_Case_Execution_Time     => 14,
            Shared_Resources_List         => (Sistema));

Operation (
      Type                    => Simple,
      Name                    => Mueve_Pendulo,
      Worst_Case_Execution_Time     => 14,
            Shared_Resources_List         => (Sistema));

-- Enclosing operations
Operation (
      Type  => Simple,
      Name  => Int_Handler,
      Worst_Case_Execution_Time     => 6);

Operation (
      Type  => Enclosing,
      Name  => Control,
      Worst_Case_Execution_Time     => 77,
      Composite_Operation_List      =>
            (Lee_Datos, Actualiza_Datos));

Operation (
```

```
      Type  => Enclosing,
      Name  => Operario,
      Worst_Case_Execution_Time     => 134,
      Composite_Operation_List      =>
            (Mueve_Pendulo));

Operation (
      Type  => Enclosing,
      Name  => Display,
      Worst_Case_Execution_Time     => 1213,
      Composite_Operation_List      =>
            (Lee_Datos));

-- Transactions
Transaction (
      Type  => Regular,
      Name  => Int_Handler,
      External_Events => (
            (Type             => Periodic,
             Name             => E1,
             Period     => 10000)),
      Internal_Events => (
            (Type       => regular,
             name       => O1,
                 Timing_Requirements => (
                   Type               => Hard_Global_Deadline,
                   Deadline     => 10000,
                   referenced_event => E1))),
      Event_Handlers => (
            (Type                   => Activity,
             Input_Event            => E1,
             Output_Event           => O1,
             Activity_Operation     => Int_Handler,
             Activity_Server  => Int_Handler)));

Transaction (
      Type  => Regular,
      Name  => Control,
      External_Events => (
            (Type             => Periodic,
             Name             => E2,
             Period     => 10000)),
      Internal_Events => (
            (Type       => regular,
             name       => O2,
                 Timing_Requirements => (
                   Type               => Hard_Global_Deadline,
                   Deadline     => 10000,
                   referenced_event => E2))),
      Event_Handlers => (
            (Type                   => Activity,
             Input_Event            => E2,
             Output_Event           => O2,
             Activity_Operation     => Control,
             Activity_Server  => Control)));

Transaction (
      Type  => Regular,
      Name  => Operario,
      External_Events => (
            (Type             => Periodic,
             Name             => E3,
             Period     => 400000)),
```

```
      Internal_Events => (
            (Type       => regular,
             name       => O3,
                 Timing_Requirements => (
                   Type               => Hard_Global_Deadline,
                   Deadline     => 400000,
                   referenced_event => E3))),
      Event_Handlers => (
            (Type                   => Activity,
             Input_Event            => E3,
             Output_Event           => O3,
             Activity_Operation     => Operario,
             Activity_Server  => Operario)));

Transaction (
      Type  => Regular,
      Name  => Display,
      External_Events => (
            (Type             => Periodic,
             Name             => E4,
             Period     => 500000)),
      Internal_Events => (
            (Type       => regular,
             name       => O4,
                 Timing_Requirements => (
                   Type               => Hard_Global_Deadline,
                   Deadline     => 500000,
                   referenced_event => E4))),
      Event_Handlers => (
            (Type                   => Activity,
             Input_Event            => E4,
             Output_Event           => O4,
             Activity_Operation     => Display,
             Activity_Server  => Display)));
```

Los cálculos ofrecen los siguientes resultados:

	B (μs)
Int handler	0
Task t1 (control)	0
Task t2 (operario)	14
Task t3 (display)	10

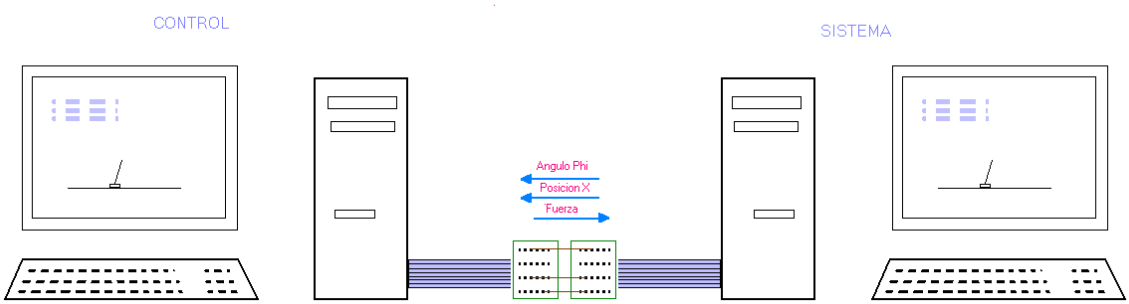
F1= 0.0009 < U(1),1 = 1.000 R0 = 9 us
F2= 0.0082 < U(2),2 = 0.828 R1 = 105 us
F3= 0.0086 < U(3),3 = 0.779 R2 = 240 us
F4= 0.0110 < U(4),4 = 0.756 R3 = 1448 us

Aquí también el análisis de planificabilidad es positivo como se esperaba, ya que la estructura de los programas y su código es muy similar.

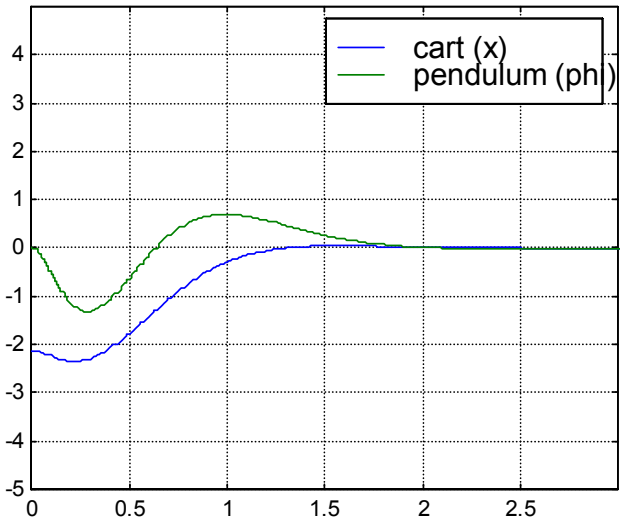
Para realizar las medidas de los tiempos, se utilizaron varios paquetes. Para las medidas en Ada relativas a las tareas y a los bloqueos por los objetos protegidos, se utilizó time_measurements. Para medir el tiempo consumido por la rutina de atención a la interrupción se usó medidas_tiempos.

7. PRUEBAS

Para comprobar la validez del diseño realizado se realizaron pruebas uniendo las aplicaciones Control y Sistema mediante la conexión de unas placas de prototipos que estaban conectadas a los convertidores A/D D/A de cada equipo y ejecutándolos conjuntamente de una forma similar a la figura.



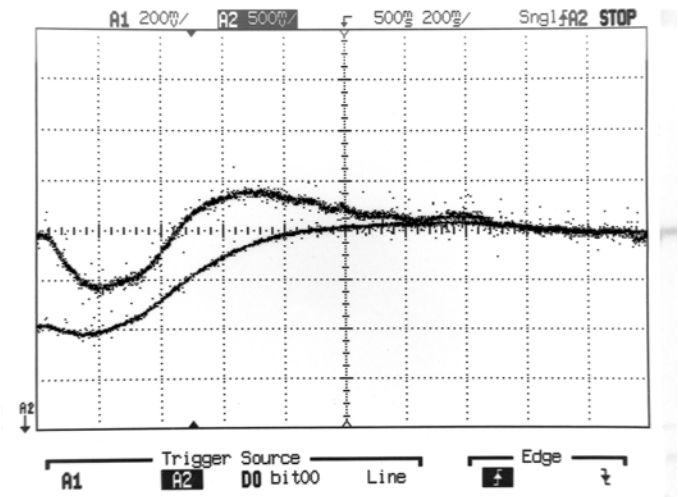
Posteriormente, la consigna de posición del péndulo invertido se sometió a un desplazamiento de 20 centímetros (ya que el diseño se hizo tomando este escalón como referencia). Los resultados del mismo podían ser vistos en el osciloscopio, en el que se tomaban las señales de voltaje relativas a la posición y el ángulo del sistema. Mediante una captura disparada por nivel era posible ver la evolución temporal de los valores hasta su estabilización. Seguidamente se tomaron fotografías de estas imágenes para a continuación poder compararlas con los cálculos teóricos y medir sus características temporales.



Simulación Matlab: Control = 100 Hz Sistema = 100 Hz

En la gráfica teórica obtenida con Matlab que se muestra arriba, el eje de tiempos está en segundos y el vertical está escalado para x y θ en la misma medida que en el osciloscopio.

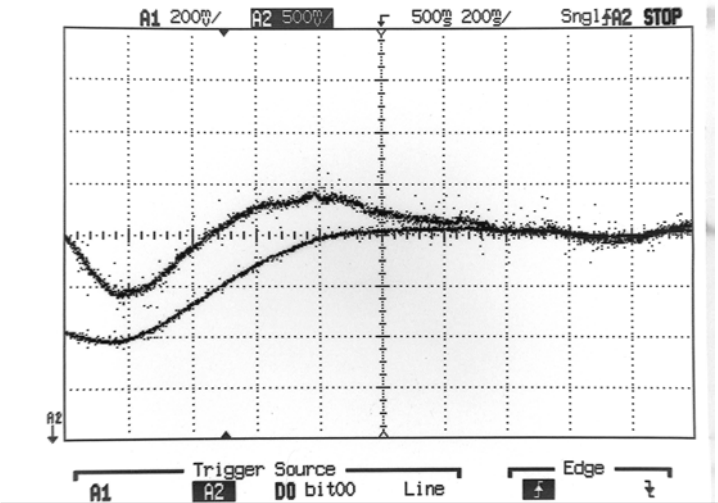
En el osciloscopio las medidas oscilan en el rango [0,5] voltios, que corresponden en posición del carro a valores entre [-0.5, +0.5] metros y en ángulo del péndulo a [-0.69, +0.69] radianes, lo que es igual a [-40, +40] grados.



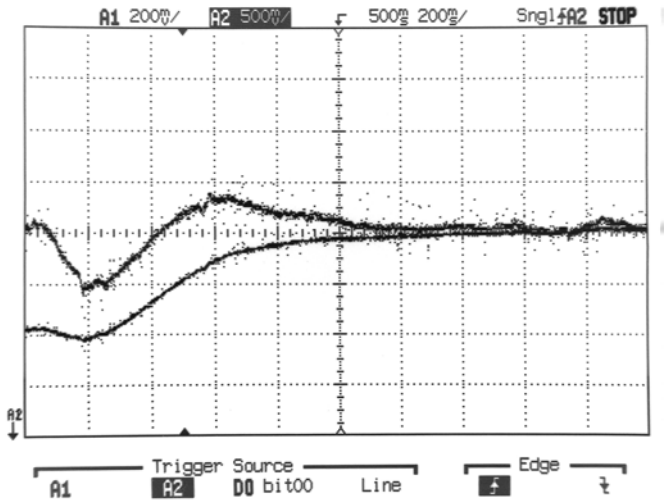
Captura Osciloscopio: Control = 100 Hz Sistema = 100 Hz

En la captura del osciloscopio podemos ver que el tiempo de establecimiento es aproximadamente ts=1.5 seg, similar al de la simulación de Matlab.

Después de realizar la prueba básica, se quiso comprobar la respuesta del Control ante un sistema cuyo modelo se acercase más a la realidad, lo cual se conseguía aumentando la frecuencia de funcionamiento de la aplicación sistema, y con ello su precisión. Para ello se recalcularon las matrices que definían el sistema cambiando algunos parámetros en el fichero de Matlab. A continuación tan solo había que introducir los nuevos valores de las matrices en el fichero de la aplicación, recompilarla y cargarla en un disquete. Así se generaron disquetes del sistema cuya frecuencia de funcionamiento era 200 Hz y 400 Hz.



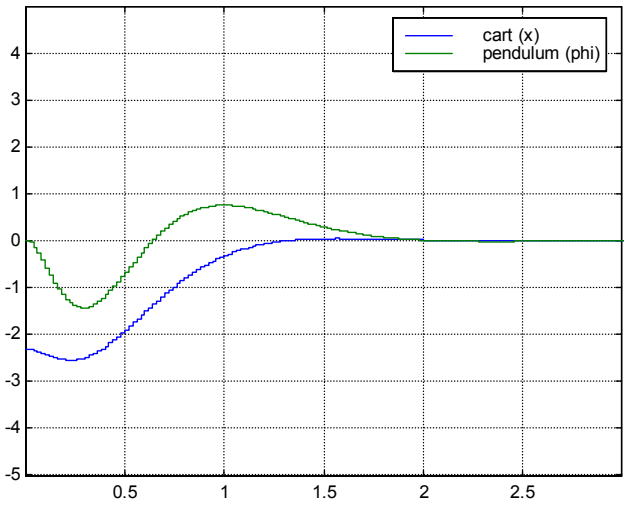
Captura Osciloscopio: Control = 100 Hz Sistema = 200 Hz



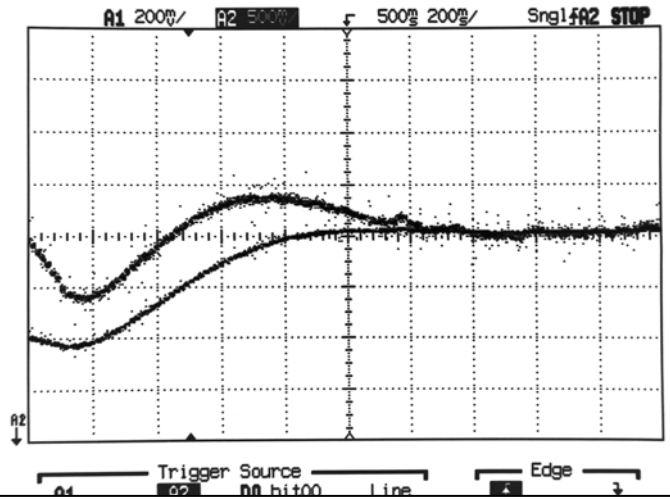
Captura Osciloscopio: Control = 100 Hz Sistema = 400 Hz

Podemos ver que la respuesta al aumentar la precisión del sistema, manteniendo fija la frecuencia del Control es muy similar en los tres casos, lo que nos sugiere que una frecuencia de muestreo en la aplicación Sistema que sea igual a la de Control es suficiente para simular el sistema real con suficiente precisión.

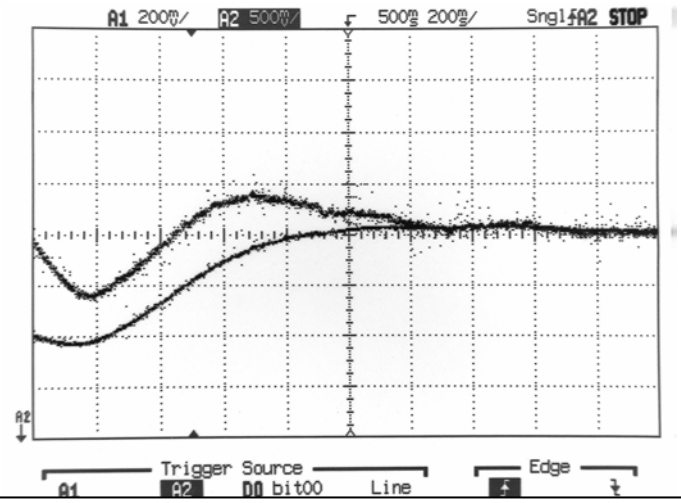
Un último experimento se hizo para comprobar la validez del sistema de Control a mayor frecuencia que la del diseño original, para observar si mejoraba la respuesta del sistema, y a menor frecuencia para ver si seguiría siendo estable. Para ello, se realizó el mismo procedimiento, recalcular las ecuaciones, recompilar los programas y cargarlos en disquetes. Se crearon disquetes de control a 50 Hz, 100 Hz y 400 Hz.



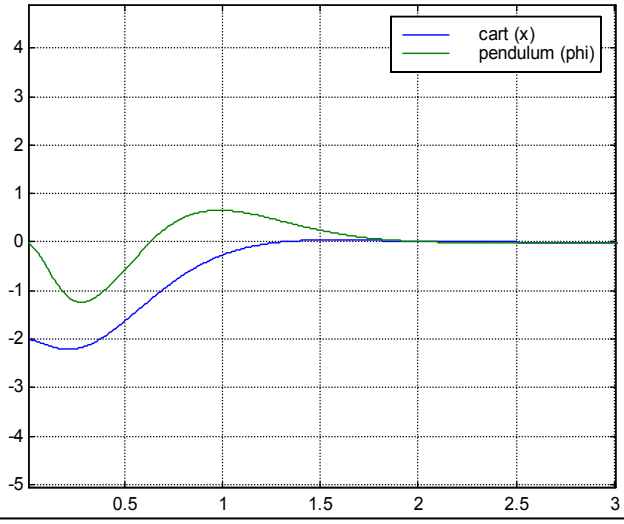
Simulación Matlab:Control = 50 Hz Sistema = 50 Hz



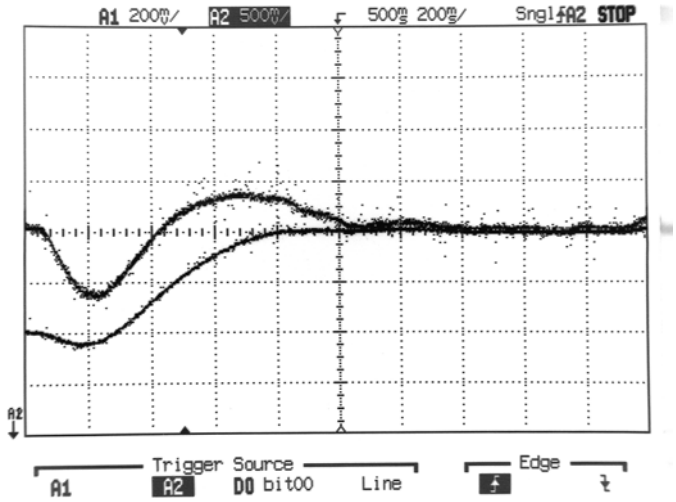
Captura Osciloscopio: Control = 50 Hz Sistema = 50 Hz



Captura Osciloscopio: Control = 50 Hz Sistema = 100 Hz



Simulación Matlab: Control = 400 Hz Sistema = 400 Hz



Captura Osciloscopio: Control = 400 Hz Sistema = 400 Hz

Con un Control a 400Hz podemos observar que el tiempo de establecimiento es ligeramente menor, manteniéndose el tiempo de pico y el máximo sobreimpulso.

Hay que tener en cuenta siempre que la frecuencia del sistema debe ser mayor o igual que la del control, de lo contrario los resultados no son los esperados, debido a que la simulación del sistema no sería suficientemente precisa.

Como conclusión, se puede extraer que el control puede ir a igual frecuencia que el sistema sin que aparezcan errores apreciables y que al aumentar la frecuencia de control el sistema se estabiliza ligeramente más deprisa, pero esa pequeña mejora no justifica ir a tanta frecuencia, con lo que se podría mantener un control a 50 Hz con una respuesta bastante aceptable. También hay que decir, que los resultados teóricos son muy similares a los prácticos, en los cuales, las pequeñas perturbaciones que se aprecian pueden ser debidas al ruido presente en las mediciones.

8. CONCLUSIONES

Como conclusión al trabajo desarrollado, haremos un breve resumen de su contenido indicando los logros conseguidos y los medios necesarios para alcanzarlos, así como las posibles líneas futuras del trabajo y las conclusiones que podemos obtener del mismo.

Fase de Estudio

En una primera fase, se realizó un estudio sobre los sistemas de tiempo real así como sobre la programación concurrente y el diseño de drivers. Se buscó una aplicación que utilizase los conocimientos adquiridos y se analizó en qué lenguajes era más conveniente realizar el diseño y sobre qué plataformas. Así se decidió que lo mas apropiado era escribir la aplicación de control y simulación en Ada y el driver en C, todo ello sobre Linux inicialmente como entorno de desarrollo más cómodo, para posteriormente trasladarlo a MaRTE O.S., y así poder garantizar el cumplimiento de todos los requisitos temporales.

Diseño

En la fase de diseño, se empezó por analizar el funcionamiento de la tarjeta conversora para realizar un driver de la misma en Linux.

Se adaptó un diseño de control existente para el problema del péndulo invertido mediante la técnica de las variables de estado. El diseño se realizó con Matlab, y posteriormente, el algoritmo obtenido se trasladó a Ada.

Se elaboraron las aplicaciones de Control y Simulación utilizando el algoritmo de control y las ecuaciones que modelaban el sistema.

Implementación

Una vez construidas las aplicaciones y el driver, se trasladó a MaRTE O.S. todo el desarrollo software, donde continuó el trabajo.

Se tomaron medidas de los tiempos de ejecución de las tareas, para posteriormente realizar con ellas el análisis de planificabilidad que nos diría si el sistema sería planificable o no.

Finalmente, se realizaron diversas pruebas modificando la frecuencia de funcionamiento de las aplicaciones Control y Sistema para verificar su validez y poder observar su rendimiento.

Conclusiones Finales

Algunas conclusiones que podemos extraer del trabajo desarrollado son las siguientes:

- Podemos trasladar fácilmente un driver de Linux a MaRTE O.S.
- Es posible realizar una aplicación en Linux, teniendo en cuenta que posee un entorno de desarrollo más cómodo para, posteriormente, trasladar la aplicación completa a MaRTE O.S. donde realizar las pruebas finales. Es decir, Linux y MaRTE O.S. son entornos entre los cuales la portabilidad es posible.
- Es sencillo trasladar a Ada un algoritmo de control diseñado en Matlab.
- La implementación de control del péndulo invertido es perfectamente realizable sobre MaRTE O.S. donde cumple de forma satisfactoria todos sus requisitos temporales, con un amplio margen para extender la aplicación a otras funciones.

Trabajo Futuro

Una vez realizada la aplicación de control del Péndulo Invertido, se sugiere como trabajo futuro, la construcción del sistema real y la adaptación del sistema de control por computador realizado al mismo. Para ello, habrá además que construir los sensores de posición y ángulo, los cuales pueden ser de muy diversos tipos (eléctricos, ópticos, mediante una cámara y procesado de imagen, etc.) y si fuera necesario, unos filtros para eliminar el ruido en las mediciones.

9. BIBLIOGRAFÍA

- Ada:

[1] Michael González. “Apuntes de Ada”. U. Cantabria, 2001.

[2] J. Barnes. “Programming In Ada 95”, first edition. Addison-Wesley, 1995.

[3] M. Feldman, E. Koffman “Ada 95: Problem Solving and Program Design”. Addison-Wesley, 1996.

- C:

[4] Michael González. “Una comparación entre C y Ada”. U. Cantabria, 2001.

[5] Rafael Menéndez de Llano.”Apuntes de Sistemas Operativos”. U. Cantabria.

[6] Márquez García F.M. “UNIX, programación avanzada”. Ra-Ma, 1996.

- Programación concurrente:

[7] J. M. Drake. “Programación concurrente”. U. Cantabria, 2001.

[8] Burns A. and Davies G. : “Concurrent Programming”. Addison Wesley, 1993.

[9] Burns A. and Wellings A. : “Concurrency in Ada”. Cambridge, 1995.

- Drivers:

[10] J. J. Gutiérrez. “Programación orientada al sistema: Drivers”. U. Cantabria, 2001.

[11] Alessandro Rubini y Jonathan Corbet : “LINUX Device Drivers”. O´Reilly, 2001.

[12] M. J. Bach: “The Design of the Unix Operating System”. Prentice Hall, 1986.

[13] IEEE Standard 1003.1-2001, “Standard for Information Technology Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API). IEEE, 2001.

- Posix:

[14] Michael González. “POSIX de Tiempo Real”. U. Cantabria, 2001.

[15] A. Burns and A. Wellings. “Real-Time Systems and Programming Languages”. Second Edition. Addison-Wesley, 2001.

[16] B. O. Gallmeister. “POSIX.4: Programming for the Real World”. O´Reilly & associates, Inc., 1996.

[17] B. Nichols, D. Buttlar and J. Proulx Farrell. "Pthreads Programming". O'Reilly & associates, Inc., 1996

- RMA:

[18] Michael González. “Análisis de Sistemas de Tiempo Real”. U. Cantabria, 2001.

[19] M.H. Klein, T. Ralya, B. Pollak, R. Obenza and M. Gonzalez Harbour. “A practitioner’s Handbook for Real-Time Analysis”. Kluwer Academic Pub., 1993.

[20] J.C. Palencia. “Análisis de planificabilidad de sistemas distribuidos de tiempo real basados en prioridades fijas”. Tesis Doctoral, 1999.

- Ingeniería de Sistemas

[21] Ogata K. Ingeniería de Control Moderna. Ed. Prentice Hall International, New Jersey. 1974.

- Páginas web:

[22] Modelado Péndulo Invertido: http://www.engin.umich.edu/group/ctm/examples/pend/invpen.html

[23] Modelado Motor: http://www.engin.umich.edu/group/ctm/examples/pend/digINVSS.html

[24] Control Digital: http://www.engin.umich.edu/group/ctm/examples/motor/motor.html

[25] MaRTE O.S.: http://marte.unican.es/

[26] MAST: http://mast.unican.es/

[27] Control sísmico de estructuras modeladas como un péndulo invertido: http://wueconb.wustl.edu:8089/eps/ge/papers/9805/9805002o.html

[28] Métodos de regulación fiscal modelados como un péndulo invertido:

http://www.construir.com/CIRSOC/muestra/document/103p1a.htm

- Otros:

[29] Manual de la tarjeta AX5411 de AXIOM. AXIOM Technology Co., Ltd., 1994.

[30] Apuntes de J. M. Drake de la tarjeta AX5411. Universidad de Cantabria.