

**Programa Oficial de Postgrado en Ciencias, Tecnología y
Computación**

Máster en Computación

Facultad de Ciencias - Universidad de Cantabria



**Modelos de Referencia en el diseño de
controladores industriales utilizando
MaRTE OS y ADA**

**Control de Posicionamiento Guiado por Láser en
Soldaduras de Virolas**

Sergio Martín Calvo

sergio.martin.calvo@gmail.com



Director:

Mario Aldea Rivas.

Grupo de Computadores y Tiempo Real

Departamento de Electrónica y Computadores.

Santander, Julio, 2008

AGRADECIMIENTOS

Me gustaría agradecer a mi director de Tesis de Máster Mario Aldea Rivas, por dejarme libertad para realizar este trabajo aunque siempre bajo su supervisión y apoyo, a Michael González Harbour con el que he realizado gran parte del proyecto para ENSA y del que he aprendido muchísimo a lo largo de este proyecto y de manera especial a todo el equipo de trabajo de ENSA, a los que no nombro uno a uno porque seguro que me olvidaría de alguien, donde he pasado mucho tiempo desarrollando, probando y poniendo a punto el prototipo y me han tratado como a uno más de su equipo de trabajo.

INDICE:

1.- Introducción	5
1.1.- MaRTE OS	6
1.2.- Lenguaje de Programación Ada	9
1.2.1.- Características principales	9
1.3.- Objetivos	10
2.- Análisis de los Módulos Software necesarios	11
2.1.- Módulos Software necesarios	11
2.2.- Estructura Global	12
2.3.- Descripción de los Módulos	12
2.3.1.- Relaciones funcionales entre los Módulos	13
3.- Control de posicionamiento guiado por Láser en soldadura de virolas	14
3.1.- Descripción del sistema de soldadura	14
3.1.1.- Descripción del proceso	14
3.2.- Descripción del controlador	16
3.2.1.- Señales de entrada/salida	16
3.3.- Resumen de las funciones del sensor láser Seampilot	18
3.3.1.- Datos generales	18
3.3.2.- Prestaciones	18
3.4.- Descripción funcional	19
3.4.1.- Modo manual	19
3.4.2.- Modo automático	20
3.4.3.- Funciones comunes a los modos de operación manual y automático	21
3.4.4.- Modo emergencia	21
3.5.- Interfaz hombre-máquina	21
3.6.- Estructura global del controlador	23
3.7.- Descripción de los módulos	25
3.7.1.- Configuración	25
3.7.1.1.- Descripción	25
3.7.1.1.- Operaciones que soporta	25
3.7.1.1.2.- Entorno	25
3.7.1.1.3.- IO Digital	26
3.7.1.1.4.- IO Analógica	26
3.7.2.- Mandos	27
3.7.2.1.- Descripción	27
3.7.2.2.- Operaciones que soporta	27
3.7.2.2.1.- Botones	27
3.7.2.2.2.- Luces	27
3.7.2.2.3.- Contadores	28
3.7.2.3.- Descripción Detallada	28
3.7.3.- Gestor de Mandos	29
3.7.3.1.- Descripción	29
3.7.3.2.- Operaciones que soporta	29
3.7.3.3.- Tratamiento de Alarmas	29
3.7.4.- Máquina	30
3.7.4.1.- Descripción	30
3.7.4.1.- Operaciones que soporta	30
3.7.4.1.1.- Sensores	30
3.7.4.1.2.- Actuadores	30

3.7.5.- Alarmas	31
3.7.5.1.- Descripción	31
3.7.5.2.- Operaciones que soporta	31
3.7.6.- Planificador	32
3.7.6.1.- Descripción	32
3.7.6.2.- Tratamiento de Alarmas	32
3.7.6.3.- Descripción detallada	32
3.7.7.- Virola	33
3.7.7.1.- Descripción	33
3.7.7.2.- Operaciones que soporta	33
3.7.7.3.- Descripción Detallada	33
3.7.8.- Seampilot	33
3.7.8.1.- Descripción	33
3.7.8.2.- Operaciones que soporta	33
3.7.8.3.- Tratamiento de Alarmas	33
3.7.8.4.- Descripción Detallada	34
3.7.8.5.- Módulo Protocolo_SeamPilot	38
3.7.9.5.1.- Descripción	38
3.7.8.6.- Módulo Envía_Recibe	39
3.7.8.6.1.- Descripción	39
3.7.10.- Reportero	40
3.7.10.1.- Descripción	40
3.7.10.2.- Descripción Detallada	40
3.7.11.- Drivers de Entrada/Salida	41
3.7.11.1.- Entradas y Salidas Digitales	41
3.7.11.1.1.- Descripción	41
3.7.11.1.2.- Operaciones que soporta	41
3.7.11.1.3.- Excepciones que eleva	41
3.7.11.2.- Salidas Analógicas	41
3.7.11.2.1.- Descripción	41
3.7.11.2.2.- Operaciones que soporta	41
4.- Brazo Telemanipulado	42
4.1.- BTM	42
4.1.1- Estructura global del controlador	42
Conclusiones	45
Bibliografía y Referencias	46

INDICE de ILUSTRACIONES:

Figura 1.1: Arquitectura de MaRTE OS..... 7

Figura 1.2: Desarrollo de Aplicaciones en MaRTE OS..... 7

Figura 2.1: Módulos sin relaciones..... 12

Figura 2.2: Módulos con relaciones..... 13

Figura 3.1: Piezas a soldar..... 14

Figura 3.2: Detalle de la poceta de soldadura..... 14

Figura 3.3: Imagen de una soldadura..... 15

Figura 3.4: Arquitectura hardware del controlador..... 17

Figura 3.5: Modos de funcionamiento..... 19

Figura 3.6: Interfaz hombre-máquina..... 22

Figura 3.7: Arquitectura del controlador..... 24

Figura 3.8: Diagrama de estados de las luces..... 28

Figura 3.9: Estado de las alarmas..... 31

Tabla I: Patrones de Identificación de Poceta..... 34-35

Figura 3.10: Diagrama Envía-Recibe..... 39

Figura 3.11: Configuración de Pantalla..... 40

Figura 4.1: Arquitectura del controlador BTM..... 43

Figura 4.2: BTM Distribuido..... 44

1.- Introducción

1.1.- MaRTE OS

MaRTE OS es un sistema operativo empotrado de tiempo real distribuido bajo licencia GNU/GPL. Ha sido desarrollado principalmente por Mario Aldea Rivas, como uno de los resultados de su tesis doctoral, dentro del Grupo de Computadores y Tiempo Real, bajo la dirección de Michael González Harbour. Sus principales características son:

- Permite ejecutar aplicaciones en máquinas desnudas.
- Da y controla el acceso a los recursos hardware.
- Es bastante diferente de un sistema de propósito general como Linux ya que no soporta sistemas de ficheros ni tiene una consola de comandos.
- Sigue el subconjunto mínimo de tiempo real POSIX.13. Dentro de POSIX.13 MaRTE OS figura dentro de la categoría "Mínimo", destinada a sistemas empotrados pequeños, en la cual no es necesario ofrecer sistema de ficheros ni multiproceso. MaRTE OS no hace uso pues de la MMU de algunas arquitecturas, no usa disco duro sino que se aloja en memoria volátil y tampoco necesita de un terminal. Es lo que se denomina en jerga POSIX como controlador de un "Tostador".

Funcionalidad del perfil mínimo del subconjunto POSIX.13 incluida en MaRTE OS:

- **Threads:** también llamados *Hilos* o *Hebras*. Un *Tread* es una secuencia de instrucciones ejecutada en paralelo con otras secuencias. Los *Threads* son una forma de dividir un programa en varias tareas que se ejecutan de forma concurrente.
- **Mútex, Variables Condicionales, Semáforos:** estructuras utilizadas para la sincronización de varios *Threads*.
- **Señales:** servicio del núcleo que permite indicar la existencia de un evento.
- **Relojes y contadores:** ofrecen funcionalidad para controlar el tiempo de ejecución de cada *Thread*.
- **Suspensión de *Threads*,** retrasos absolutos y relativos.
- **"Ficheros" de dispositivo y entrada/salida** Permiten tratar los dispositivos desde un punto de vista abstracto como si fueran un fichero al que realizar llamadas del tipo **open, read, write, ...**
- Funcionalidad extra: Manejadores de interrupciones hardware, planificación definida por la aplicación.

Ofrece concurrencia a nivel de thread (tareas en Ada) pero no a nivel de procesos. En MaRTE OS solo hay un proceso (que será el encargado de crear los threads-tareas necesarios), y por tanto un único espacio de direcciones de memoria.

Todos los servicios tienen una respuesta y latencia temporal acotada, por lo que se puede utilizar para aplicaciones de tiempo real (tanto estricto como no).

El espacio de direcciones es compartido por el *núcleo (kernel)* y la aplicación con lo cual no se provee la protección de otros sistemas operativos y se debe probar a fondo el sistema final. La ventaja es que permite una mayor velocidad de ejecución.

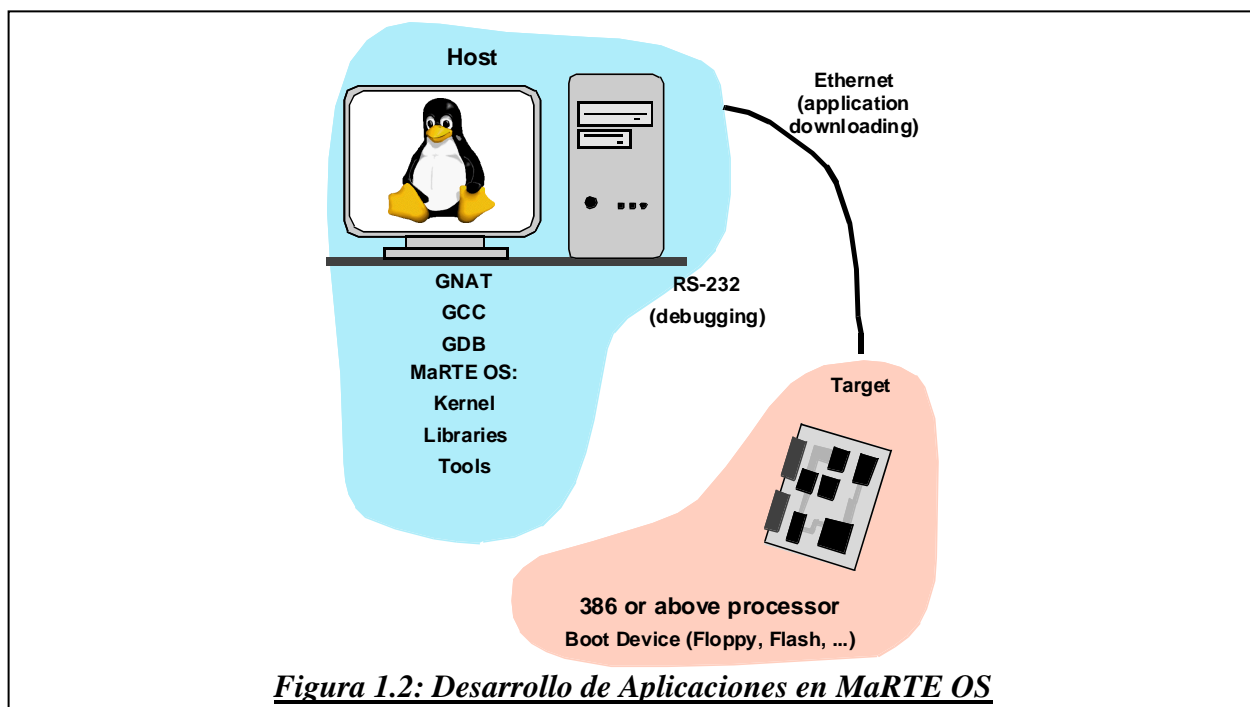
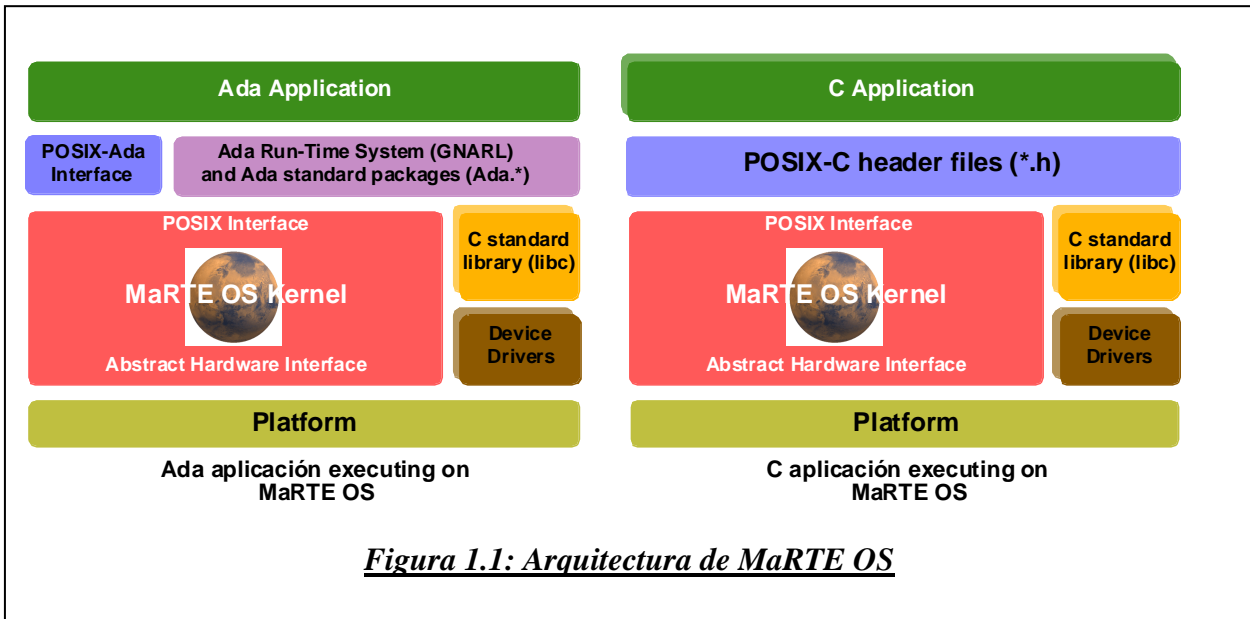
Es un *núcleo* monolítico. Un núcleo monolítico se caracteriza porque cuando se está ejecutando una parte de su código no hay otras partes ejecutándose concurrentemente. Permite simplificar su desarrollo y ser muy eficientes en sistemas monoprocesador, aunque es complicado portarlo a sistemas multiprocesador. Linux en sus inicios también era un núcleo monolítico aunque hoy en día es un híbrido.

Está escrito en Ada 95 (salvo algo de código en C y ensamblador) pero ofrece interfaces tanto para aplicaciones Ada como C, o incluso aplicaciones que mezclen threads con tareas Ada.

Es portable a diferentes plataformas gracias a una capa de abstracción hardware. Incluyendo microcontroladores, de gran importancia de en el mundo de los sistemas embebidos.

Actualmente se encuentran implementadas para:

- Arquitectura x86 PC (bien en máquinas desnudas o bien en emuladores). El *núcleo* es compatible con cargadores que usen el protocolo Multiboot.
- Arquitectura Linux y LinuxLib, donde MaRTE OS corre dentro de Linux. Principalmente destinada a investigación y enseñanza (no se puede alcanzar requerimientos de tiempo real estricto dentro de un sistema que no lo es, como Linux).



En la figura podemos ver la arquitectura en forma de capas de MaRTE OS para una aplicación escrita en Ada y en C. La principal diferencia entre ambas radica en la capa utilizada para comunicar la aplicación con el resto del sistema.

En el caso de las aplicaciones Ada, esta interacción se realiza en su mayor parte de forma indirecta a través de la librería de tiempo de ejecución del compilador GNAT, llamada GNARL, y en parte a través de la interfaz POSIX-Ada. Es decir, para crear un hilo de ejecución no usamos una función como en C, sino que creamos una tarea Ada, mediante la palabra **task**, y sería la librería GNARL la que se encargue de transformar adecuadamente esa instrucción según el sistema operativo donde se ejecute (o incluso puede ser una máquina desnuda). Sin embargo, si queremos sincronizar una tarea con un thread C, no podemos usar las utilidades de Ada (como los objetos protegidos) sino que necesitamos recurrir a servicios del sistema operativo, los **Mutexes** en este caso, mediante la interfaz POSIX-Ada. La librería de tiempo de ejecución es en su mayor parte independiente de la plataforma sobre la que se ejecuta. Sólo una pequeña parte, denominada GNULL (GNU Low-level Library), debe ser modificada para adaptarse a las diferentes plataformas de ejecución. Esta adaptación es necesaria para cada versión del compilador GNAT por lo que MaRTE OS es definitivamente dependiente del compilador GNAT. No podemos compilarlo con un compilador Ada cualquiera.

En el caso de las aplicaciones C, la interacción se realiza a través de la interfaz POSIX. 1, una capa muy delgada consistente únicamente en un conjunto de ficheros de cabecera que definen directamente las funciones exportadas por el núcleo de MaRTE OS y la librería estándar C.

Como puede apreciarse en ambas figuras, el núcleo incluye una interfaz abstracta de bajo nivel para acceder al hardware. En ella se define la visión que del hardware tienen las partes del núcleo que son independientes de la plataforma de ejecución (Ej: x86, PowerPC, ARM, MIPS,...). Esta interfaz constituye la **única parte del núcleo que es dependiente del hardware**, lo que facilita el portado de MaRTE OS a distintas plataformas. Por otro lado, los gestores de dispositivos (drivers) también presentarían dependencias respecto al hardware sobre el que se ejecutan. Además, la librería estándar C depende del hardware sobre el que se ejecuta en lo referente a las operaciones de entrada/salida por consola. La E/S por consola no está incluida dentro de la interfaz abstracta con el hardware porque no es común a todas las posibles arquitecturas (los sistemas empotrados normalmente no disponen de dispositivo de interfaz con el usuario).

1.2.- Lenguaje de Programación Ada

La programación de computadores, esto es, la creación de software para ellos, es un proceso de escritura. Así como los escritores de libros y artículos de revistas, los programadores (escritores de programas) deben expresar sus ideas por medio de una lengua escrita. Sin embargo, a diferencia de los escritores de novelas, los programadores deben expresar sus textos en lenguajes de propósito especial, basados en las matemáticas, conocidos como lenguajes de programación.

Ada, es uno de entre muchos posibles lenguajes de programación. Fue diseñado con un claro propósito en mente: la calidad del producto. Entendiéndose por calidad, la confianza que los usuarios van a poder depositar en el programa.

Si bien es posible escribir cualquier programa en Ada, éste ha sido utilizado principalmente, en el desarrollo de **software de control, de tiempo real y de misión crítica**. Típicamente, estos sistemas son responsables de procesos industriales y militares, muy costosos, y en los cuales incluso vidas humanas dependen del buen funcionamiento del software. Es vital en tales sistemas, utilizar un lenguaje que como Ada, ayuda en la creación de software de alta calidad.

Por otro lado, Ada, como lenguaje que promueve las buenas prácticas en ingeniería del software, es muy usado en la enseñanza de la programación en muchas universidades de todo el mundo.

Ada se usa principalmente en entornos en los que se necesita una gran seguridad y confiabilidad como la defensa, la aeronáutica (Boeing o Airbus), la gestión del tráfico aéreo (como Indra en España) y la industria aeroespacial entre otros.

1.2.1- Características principales

Legibilidad

Los programas profesionales se leen muchas más veces de las que se escriben, por tanto, conviene evitar una notación que permita escribir el programa fácilmente, pero que sea difícil leerlo excepto, quizás, por el autor original y no mucho tiempo después de escribirlo.

Tipado fuerte

Esto asegura que todo objeto tenga un conjunto de valores que esté claramente definido e impide la confusión entre conceptos lógicamente distintos. Como consecuencia, el compilador detecta más errores que en otros lenguajes.

Construcción de grandes programas

Se necesitan mecanismos de encapsulado para compilar separadamente y para gestionar bibliotecas de cara a crear programas transportables y mantenibles de cualquier tamaño.

Manejo de excepciones

Los programas reales raramente son totalmente correctos. Es necesario proporcionar medios para que el programa se pueda construir en capas y por partes, de tal forma que se puedan limitar las consecuencias de los errores que se presenten en cualquiera de las partes.

Abstracción de datos

Se puede obtener mayor transportabilidad y mejor mantenimiento si se pueden separar los detalles de la representación de los datos y las especificaciones de las operaciones lógicas sobre los mismos.

Procesamiento paralelo

Para muchas aplicaciones es importante que el programa se pueda implementar como una serie de actividades paralelas. Dotando al lenguaje de estos mecanismos, se evita tener que añadirlos por medio de llamadas al sistema operativo, con lo que se consigue mayor transportabilidad y fiabilidad.

Unidades genéricas

En muchos casos, la lógica de parte de un programa es independiente de los tipos de los valores que estén siendo manipulados. Para ello, se necesita un mecanismo que permita la creación de piezas de programa similares a partir de un único original. Esto es especialmente útil para la creación de bibliotecas.

1.3.- Objetivos

Los controladores industriales requieren implementar un conjunto de módulos software que son comunes a todos ellos: gestión de terminales de mando, drivers de dispositivos físicos, evaluación de trayectorias, control de servos, registro de eventos, tratamientos de acciones de emergencia, etc. Asimismo, hay que conseguir características que son comunes a todos ellos, como garantizar requisitos de tiempo real, ser implementados con recursos de memoria limitados o no disponer de sistemas de almacenamiento masivo.

El trabajo que se propone es identificar y seleccionar un conjunto básico de estos módulos, así como caracterizar su funcionalidad, los parámetros de configuración que los hacen reutilizables y los patrones de interacción básicos entre ellos, el sistema que controlan y el operador. Así como proponer un modelo de referencia optimizado para ser implementado en el lenguaje de programación Ada y sobre el sistema operativo de tiempo real Marte OS. El objetivo de este trabajo es proponer las bases para el diseño modular de controladores industriales, que reduzca el esfuerzo y el tiempo para su desarrollo, y la propuesta de un proceso que garantice niveles contrastados de calidad interna y externa del software que se genere.

Los resultados del trabajo se validarán aplicándolos al desarrollo del controlador de posicionamiento por perfil láser en la soldadura de virolas y comparándolos con otro controlador industrial, el BTM.

2.- Análisis de los Módulos Software necesarios

2.1.- Módulos Software necesarios

Los sistemas que vamos a modular son sistemas de entorno industrial en los que se pueden diferenciar una serie de paquetes básicos a los que podremos definir su funcionalidad y las relaciones básicas que van a tener entre ellos.

En estos sistemas vamos a tener uno o varios operadores a cargo de la aplicación por lo que será necesario un panel de control de la aplicación desde el cual el operario dará ordenes a la aplicación mediante controles como botones, joysticks... y ésta le responderá indicándole el estado del proceso que se encuentre haciendo mediante luces, displays..., para gestionar este módulo y toda la funcionalidad propia de él vamos a definir un primer paquete llamado “Gestor de Mandos”, que se encargara de aplicar la funcionalidad propia del sistema, además asociado directamente a este paquete habrá otro que se encargue del funcionamiento de todos esos elementos (Luces, Botones, Displays...) y se encargará de leer el estado y hacerle cambiar, a este otro paquete le denominaremos “Mandos”. También sería recomendable que el sistema tuviese una interfaz grafica, a la que podríamos denominar el “Reportero” de la aplicación, en la que el operador podría ver de forma directa el estado, las incidencias, posibles alarmas..., que sería conveniente que se tratasen de forma aislada en otro paquete que denominaremos “Alarmas”.

Hasta ahora hemos hablado de una parte del sistema que es la que se refiere a la interacción del operario con el sistema, tanto las ordenes que este le da, como las respuestas que obtiene de él, la otra parte a tratar es la funcionalidad del sistema, que en cada aplicación variara considerablemente en función de las características propias de la aplicación y del fin que se persiga, pero en este tipo de sistemas que estamos analizando podríamos hablar de que hay una parte en la que tenemos que controlar una maquina, por lo que podríamos usar una estructura similar a las del gestor de mando pero para la gestión del robot, podríamos tener un paquete “Control de Servos” con el que calcularíamos las consignas que hay que introducir a estos para que se muevan de la forma deseada, y otro que sería el paquete “Maquina” que sería el análogo a “Mandos” en el que podríamos actuar sobre los valores que manejase como podrían ser las consignas de los servos, relees... a los que denominaremos “Actuadores” y podríamos leer posibles valores que nos ofrezca el robot que estemos utilizando, a los que llamaremos “Sensores”, estos sensores podrían ser topes de recorrido, posiciones, o acciones que este llevando acabo el robot, además de esto crearemos un paquete en el que introduciremos todos los parámetros susceptibles de modificación para tenerlos agrupados, y que sea mas sencilla la configuración del sistema, a este paquete lo denominaremos “Configuración”.

El funcionamiento del sistema estará regido por un “Planificador”, que dependiendo del estado de sistema, las órdenes que haya introducido el operador y de otros estímulos exteriores se encargara de planificar las acciones a realizar por el controlador.

A parte de estos módulos introducidos, cada aplicación tendrá módulos específicos que describirán la funcionalidad de elementos propios del sistema.

2.2.- Estructura Global

La estructura básica del sistema estaría compuesta por estos módulos:

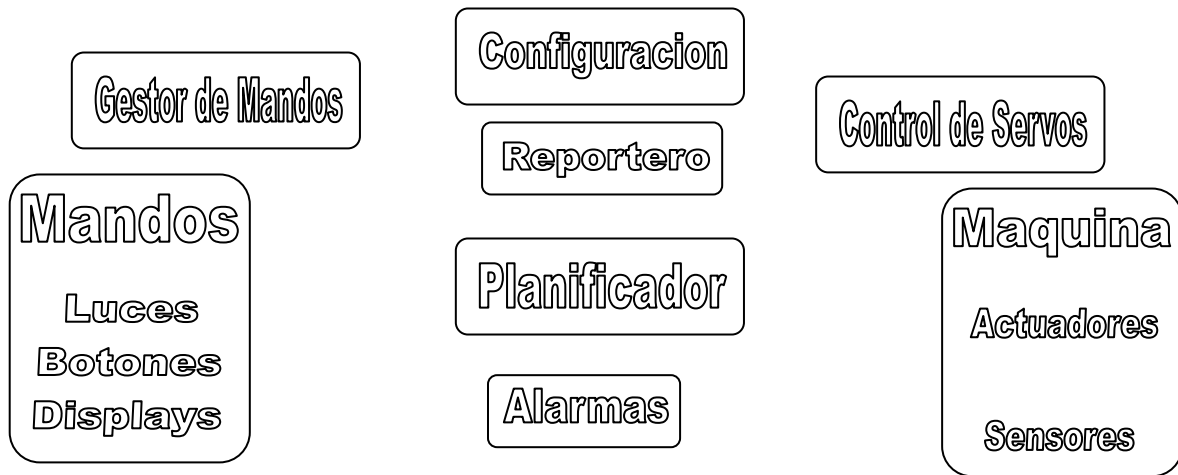


Figura 2.1: Módulos sin relaciones

2.3.- Descripción de los Módulos

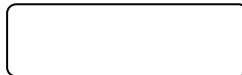
- **Configuración:** En este módulo se describen todos los parámetros del sistema que son configurables. Aunque muchos de estos parámetros corresponden a otros módulos del sistema, tales como los propios de los mandos o el robot, se ha preferido agruparlos todos en un mismo módulo para simplificar la labor de configurar el sistema. Casi todos los módulos del sistema leen los parámetros configurables del módulo configuración.
- **Mandos:** Este módulo engloba todos los elementos de la unidad de mandos del robot, en concreto la botonera, las luces de la botonera, la seta de emergencia, posibles displays. El módulo proporciona operaciones para leer el estado de los mandos, así como para poder encender o apagar las diferentes luces del cuadro de mandos, poner valor a los displays...
- **Máquina:** Este módulo englobaría los elementos de la máquina de sistema en cuestión. Los elementos se agrupan en dos tipos: sensores, cuyos valores se pueden leer, y actuadores, sobre los que se puede actuar. Los sensores serían los que permitirían extraer información que obtuviese la máquina, como podrían ser topes de recorrido. Los actuadores son los que permitirían especificar las consignas de los servos, la salida de habilitación de los mismos entre otras funcionalidades que podría tener la máquina en cada caso.
- **Gestor de Mandos:** Este módulo gestiona las órdenes que el usuario introduce a través de la unidad de mandos. En concreto determina el modo de funcionamiento del controlador, gestiona los cambios de modo de funcionamiento, y almacena órdenes que el planificador debe ejecutar. También gestiona las luces del cuadro de mandos, encendiéndolas y apagándolas, así como posibles displays en función del estado y de las alarmas existentes.
- **Planificador:** Este módulo se encarga de planificar las acciones a realizar por el controlador. Como entradas, recibirá las órdenes que provienen del gestor de mandos y del Control de Servos y de los módulos específicos de cada sistema. Como salida proporciona periódicamente las acciones a tomar por el controlador.
- **Reportero:** Este módulo se encarga de leer periódicamente el estado del sistema, y presentarlo.
- **Alarmas:** Este módulo se encarga de almacenar el estado de las alarmas del sistema. Otros módulos pueden modificar este estado, o leerlo.

2.3.1.- Relaciones funcionales entre los Módulos

Las principales relaciones entre los módulos serán:

- Los módulos deberán tener acceso para leer los datos de configuración que se encuentren en el paquete “Configuracion”.
- Los Módulos que tengan que tomar decisiones como el “Planificador”, “Gestor de Mandos”, “Control de Servos” deberán tener acceso tanto a leer como a escribir en “Alarmas”; el encargado de mostrar el estado del sistema será el “Reportero” y deberán tener acceso para leerlas también.
- Entre el “Gestor de Mandos” y los “Mandos”, en la que el primero leerá y escribirá el estado del segundo
- Entre el “Control de Servos” y la “Maquina” en la que el primero leerá los sensores y actuara sobre los actuadores que componen el segundo.
- El “Planificador” al que podríamos llamar la cabeza pensante del sistema deberá tener acceso a toda la información que le puedan ofrecer el resto de los paquetes para tomar las decisiones adecuadas en función de lo que se quiera llevar a cabo.

Configuracion



Gestor de Mandos

3.- Control de posicionamiento guiado por Láser en soldadura de virolas

3.1.- Descripción del sistema de soldadura

Esta sección describe la especificación funcional del sistema de soldadura de virolas con posicionamiento por levantamiento de perfiles láser. En primer lugar se describe el proceso de soldadura, y en segundo lugar las características del controlador hardware.

3.1.1.- Descripción del proceso

El proceso consiste en la soldadura de virolas cilíndricas mediante un proceso de soldadura de arco sumergido bajo flux. La figura 3.1 muestra un diagrama de las piezas a soldar, que están montadas sobre un virador que las hace girar, mientras que la máquina de soldadura está fija en la parte superior.

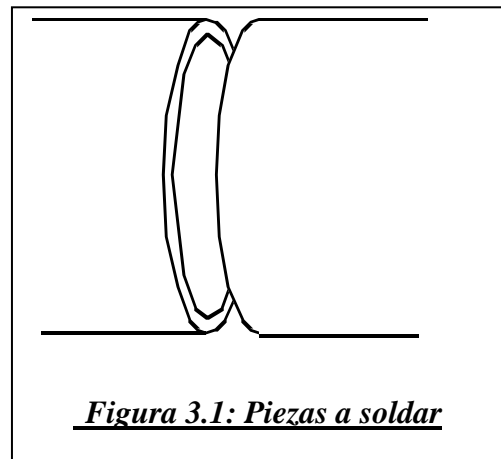


Figura 3.1: Piezas a soldar

La figura 3.2 muestra un detalle de la geometría de la soldadura, mediante un corte transversal de la pieza a soldar. La zona donde se deposita el material de soldadura, que denominaremos “poceta”, tiene una forma trapezoidal, aunque casi rectangular. La soldadura se hace mediante múltiples capas, habitualmente usando dos o tres cordones por capa. Al final de cada cordón la máquina debe cambiar la posición del cabezal de soldadura para dar el siguiente cordón. En este proceso, el fondo de la poceta va cambiando de forma y elevándose, a medida que se depositan cordones.

La figura 3.3 muestra una imagen de un corte transversal de una soldadura ya realizada. La pieza a soldar se desplaza sobre el virador y puede llegar a tener desplazamientos de más de 100 mm. Además, puede haber variaciones en la altura de la pieza por falta de cilindricidad. Por ello se requiere que el sistema de control de la soldadura sea capaz de detectar la posición de la pieza a soldar. En particular, es preciso detectar el centro geométrico de la poceta en el sentido transversal a la misma, así como la profundidad. Con ello se controlará la posición central de la antorcha de soldadura así como su altura con respecto al fondo de la poceta. El arco eléctrico de la antorcha tiene una longitud aproximada de unos 12 mm que no se controla directamente. El proceso de soldadura es a potencia constante. La antorcha tiene una posibilidad de movimiento lateral manual (con posibilidad de que esté actuado por el controlador en el futuro), que se usa por el operador para dar el cordón izquierdo, central, o derecho, así como para hacer pequeños ajustes en la trayectoria.

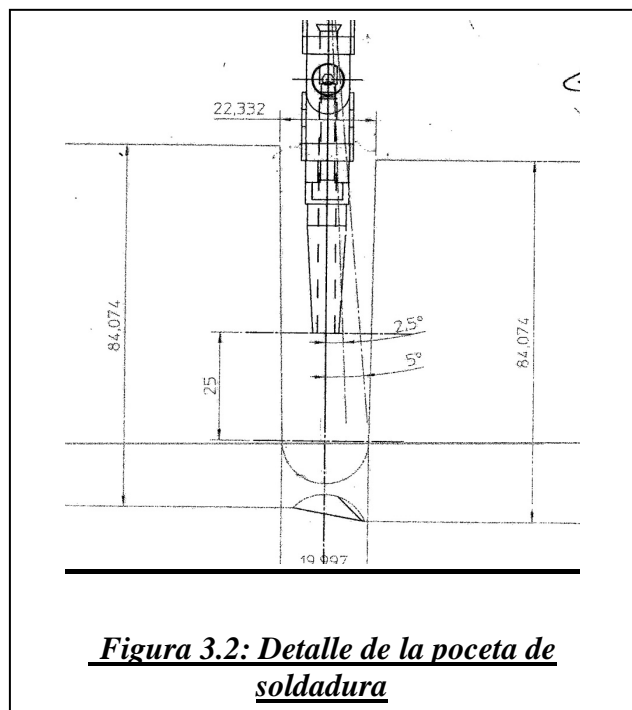


Figura 3.2: Detalle de la poceta de soldadura

Actualmente este movimiento lateral no es detectado por el controlador, aunque está previsto que en el futuro pueda serlo. También se prevé una entrada para informar al controlador sobre si la máquina está o no soldando, aunque en la primera versión esta entrada no se utilizará. Para determinar la altura de la poceta se calculará la altura media de la banda central del fondo de la poceta. Se eliminan los bordes ya que pueden tener una altura irregular. La banda central comprenderá entre el 60 y el 80% de la anchura total (configurable). La media se hará también sobre los últimos N perfiles (configurable). Uno de los problemas a resolver es la detección del principio de cada cordón, con objeto de poder comenzar el siguiente, después de permitir un cierto solapamiento entre los dos. Esto se hará detectando anticipadamente el inicio del cordón en el perfil obtenido del sensor láser. Se considera que se ha llegado al inicio del cordón cuando se produce un incremento de altura media equivalente al 60% de la altura del cordón, en una de las tres zonas centrales de cada posible cordón.

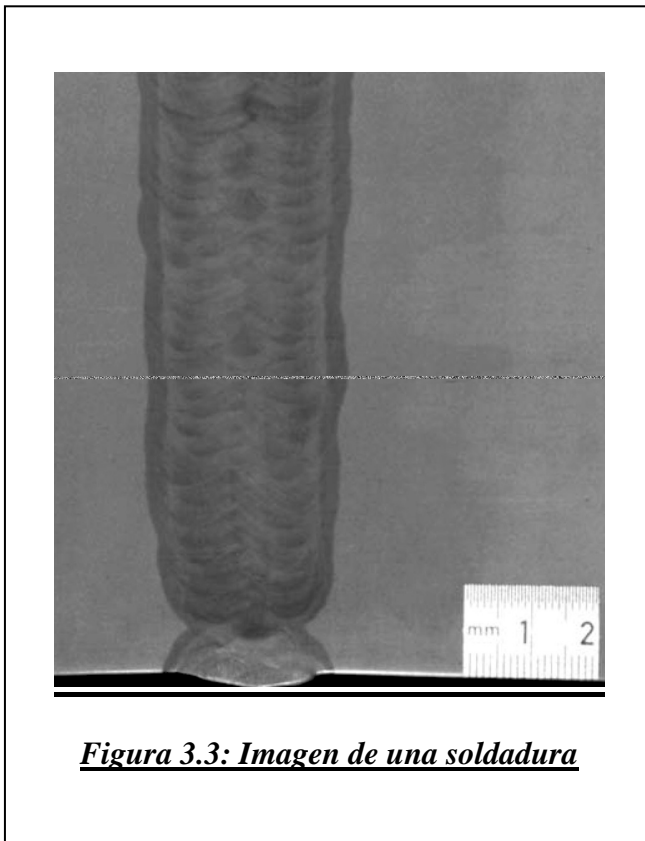


Figura 3.3: Imagen de una soldadura

La velocidad del virador es más o menos constante, pero ajustable, con valores de velocidad lineal que pueden oscilar entre 40mm/minuto y 400mm/minuto. El controlador la mide para calcular el tiempo que transcurre entre la detección del inicio del cordón y el instante en que éste llega a la antorcha de soldadura. La precisión requerida no es muy alta, en torno al 1%. Para ello se dispondrá de una o varias marcas en el exterior de la poceta de soldadura, que se detectarán por medio de un sensor de proximidad (por determinar), de modo que se mide el tiempo de paso desde una de esas marcas a la siguiente. La distancia de una de estas marcas a la siguiente puede ser del orden de 200 mm. Para definir la geometría de la poceta utilizaremos dos ejes de coordenadas. Aunque se definen en relación a la poceta de soldadura, la descripción es aproximada ya que la posición de la poceta respecto a la máquina de soldadura es muy relativa.

- El eje de desplazamiento (lateral) es horizontal, paralelo (aproximadamente) al eje del virador, y tiene su origen de coordenadas en un lugar que se sitúa aproximadamente en el centro de la poceta. Los desplazamientos serán de ± 150 milímetros. El sentido positivo es hacia la derecha si miramos al sensor láser desde la antorcha de soldadura.
- El eje de profundidad es vertical, con sentido positivo hacia abajo, y tiene el origen de coordenadas en un lugar que es aproximadamente la parte superior de la poceta. Tiene un rango de ± 150 milímetros. Como unidades de longitud se utilizarán los milímetros.

3.2.- Descripción del controlador

Para controlar la antorcha de soldadura el controlador será capaz de moverla en un plano. Los movimientos son de profundidad (en la dirección del fondo de la poceta), y desplazamiento lateral (trasversal a la poceta). Para ello contará con dos servomotores. El movimiento no tiene requisitos de velocidad alta. En particular será suficiente con alcanzar velocidades de 2 mm/s en profundidad, y 2 mm/s en desplazamiento. No es preciso contemplar rampas de aceleración constante. Para determinar la posición central de la poceta y su profundidad se cuenta con un sistema capaz de obtener perfiles de elevación mediante láser. El sistema se denomina SeamPilot y se describe en la sección 3.3, “Resumen de las funciones del sensor láser SeamPilot”.

3.2.1.- Señales de entrada/salida

El controlador se comunica con el SeamPilot mediante una línea RS422, ya que presenta mayor inmunidad al ruido que la RS-232. El controlador dispone de una línea de este tipo (COM2).

Salidas analógicas (2):

- Consigna del motor de desplazamiento
- Consigna del motor de profundidad

Entradas digitales (18)

- 2 botones de cambio de modo (manual, automático)
- 4 botones de movimiento (arriba, abajo, izquierda, derecha)
- 1 botón de prueba de luces
- 2 botones para subir/bajar display de distancia láser-antorcha
- 2 botones para subir/bajar display de solapamiento de cordón
- 1 botón de reconocimiento
- 1 sensor de proximidad para medida de velocidad
- 4 señales de topes de recorrido
- 1 seta de emergencia

Salidas digitales (19)

- 2 luces de modo de operación (manual, automático)
- 4 luces de errores (cancelado modo auto, profundidad fuera de rango, tope de recorrido, otros errores)
- 2 luces para indicaciones del seampilot (error SeamPilot, perfil adquirido)
- 1 luz de final de cordón
- 1 luz de reconocimiento
- 1 línea para inhibir/habilitar los servos
- 3 líneas (pulsos, reset y dirección) para el contador del display de distancia láser-antorcha
- 3 líneas (pulsos, reset y dirección) para el contador del display de solapamiento de cordón
- 1 zumbador de aviso/error.

Adicionalmente se prevé dejar espacio para las siguientes entradas y salidas, que de momento no se utilizarán:

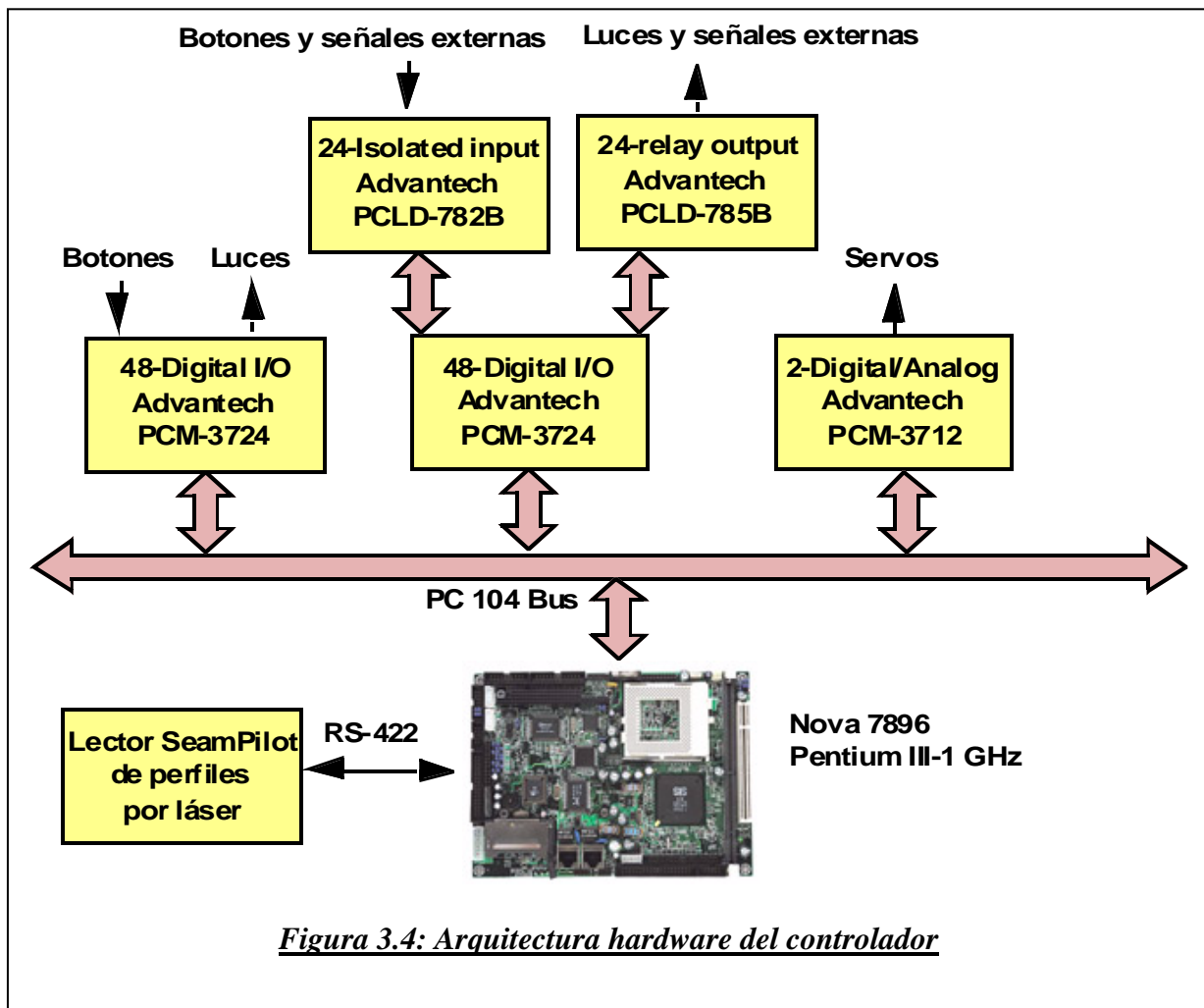
Entradas digitales (7)

- Realimentación de posición de cordón derecha
- Realimentación de posición de cordón centro
- Realimentación de posición de cordón izquierda
- Botón de posición de cordón derecha
- Botón de posición de cordón centro
- Botón de posición de cordón izquierda
- Entrada de máquina soldando

Salidas digitales (6)

- Actuación de posición de cordón derecha
- Actuación de posición de cordón centro
- Actuación de posición de cordón izquierda
- Luz de posición de cordón derecha
- Luz de posición de cordón centro
- Luz de posición de cordón izquierda

En total, serán necesarias pues 25 entradas y 25 salidas digitales (total 50).



3.3.- Resumen de las funciones del sensor láser Seampilot

3.3.1.- Datos generales

El sistema *SeamPilot* se compone de la cámara láser y la Unidad de Procesado de Señal (SPU):

- Cámara láser: Tiene un barrido de 40° de ancho y de 58mm de profundidad.
- SPU: se conecta directamente con la cámara y, a través de una línea serie RS-232 o RS-422, con el computador que hace de controlador del proceso.

La comunicación SPU-controlador se realiza utilizando un protocolo especial que permite transmitir mensajes (de 256 bytes como máximo) desde el controlador a la SPU o en sentido opuesto. La SPU puede presentar dos interfaces de comunicación, cada una con una funcionalidad distinta. Los mensajes enviados desde el controlador a la SPU permiten configurar el modo de funcionamiento de esta última (salvo la velocidad del puerto serie y la interfaz de comunicación que se configuran con “switches”) o solicitar un mensaje determinado. La SPU envía al controlador mensajes que permiten monitorizar su estado y otros que contienen los puntos obtenidos (con mayor o menor preproceso por su parte).

La SPU tiene dos puertos serie (AUX y MAIN), lo que permite monitorizar desde un segundo computador el funcionamiento del conjunto SPU-controlador.

3.3.2.- Prestaciones

Las principales características del sistema son:

- Capaz de realizar aproximadamente 2000 medidas (puntos) por segundo.
- Máxima frecuencia de barrido del haz: 10 “scans” por segundo (los otros valores posibles son 5 y 2.5 barridos por segundo). A la máxima frecuencia cada perfil se compone de 201 puntos.
- Máxima velocidad de transferencia por la línea serie: 38400 baudios, por lo tanto pueden transmitirse como máximo únicamente 3 mensajes de 201 puntos por segundo o 10 mensajes de 70 puntos.
- Los puntos obtenidos por el sistema consisten en las coordenadas Y y Z respecto al sistema de referencia de la cámara. Cada una de estas coordenadas es codificada como una palabra de 2 bytes con un resolución de 1/32 mm por bit o 1/1024 pulgadas por bit (según cuál sea la unidad de medida seleccionada).

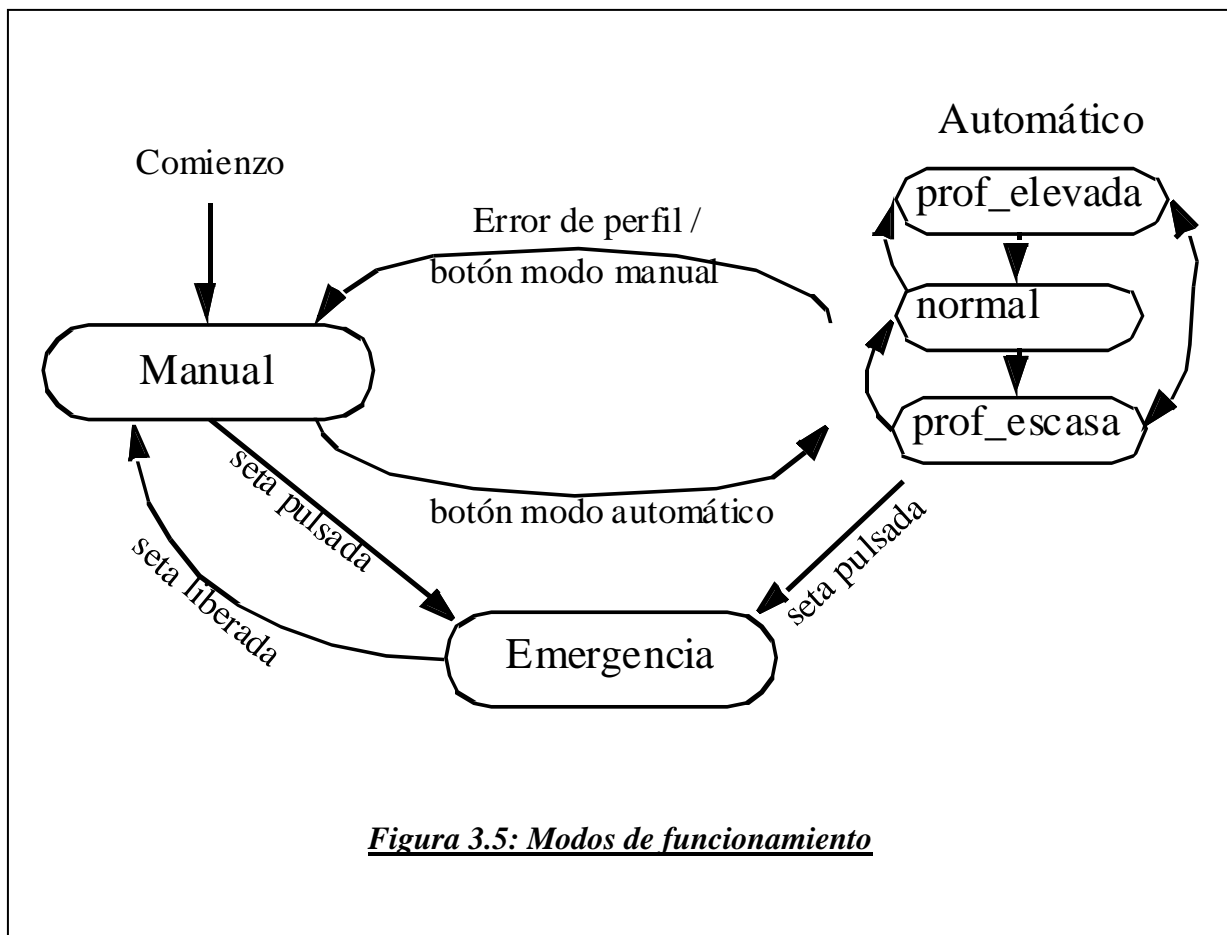
3.4.- Descripción funcional

El sistema al arrancar inicializa el software y luego pasa a modo *manual*, en el que se pueden mover los motores mediante botones. Desde ese modo se puede pasar al modo *automático*, en el que el controlador mueve los motores para tener la antorcha de soldadura colocada en la posición relativa que haya sido aprendida. La primera vez que se entra en el modo automático, la posición se aprende en ese momento y es aquella en la que esté el sistema en ese momento. En las sucesivas veces, se irá a la última posición aprendida. La posición aprendida se puede modificar en el modo automático con los botones de movimiento

3.4.1.- Modo manual

En este modo el operador puede mover los dos motores mediante los botones de “desplazamiento izquierda”, “desplazamiento derecha”, “profundidad abajo” y profundidad arriba”. En este modo la luz “modo manual” permanece encendida. Se sale de este modo hacia el modo automático cuando se pulsa el botón “modo automático” y se comprueba que el seampilot está adquiriendo perfiles correctamente y la profundidad no es elevada.

Cuando la profundidad sea elevada no se permitirá entrar en el modo automático, y se mantendrá la luz de “profundidad fuera de rango” encendida.



3.4.2.- Modo automático

El modo automático tiene tres submodos que dependen de la profundidad de la poceta. Si la profundidad es mayor al umbral establecido en la configuración, el sistema estará en el submodo “en poceta”, en el que se controlan tanto el desplazamiento como la profundidad, para situarse siempre en el punto aprendido. Si se detecta que la profundidad es escasa, se pasa al submodo “prof_escasa”, en el que sólo se controla la profundidad, para situarse en la profundidad aprendida. Al pasar al submodo “prof_escasa”, el aviso “profundidad fuera de rango” parpadea y suena el zumbador intermitentemente. Cuando se reconoce este aviso, “profundidad fuera de rango” permanece iluminado y se desconecta el zumbador. Cuando no se detecta el fondo de la poceta el submodo es profundidad elevada. En este caso no se controla la profundidad, pero sí el desplazamiento.

Al entrar en el modo automático por primera vez el sistema aprende la posición actual de la antorcha con respecto a la posición de la poceta (desplazamiento y profundidad), y a partir de ese momento controla los dos ejes para mantener esa posición aprendida.

Al entrar en el modo automático en ocasiones posteriores a la primera, el sistema se dirige al último punto aprendido.

El punto aprendido puede modificarse con los botones de desplazamiento, que lo irán moviendo a una velocidad prefijada.

La realimentación es mediante los perfiles obtenidos con la SeamPilot, teniendo en cuenta la distancia que hay entre el sensor láser y la antorcha de soldadura, que es un valor que el operador puede ajustar mediante dos botones y un display que muestra el valor actual.

Si después de un número configurable de perfiles no se ha obtenido un perfil correcto, se activa el error “cancelado modo auto” y se pasa al modo manual. En el error “cancelado modo auto” se activa la luz de ese error de forma intermitente, y además se activa el zumbador también de forma intermitente.

Mientras está en el modo automático el sistema está a la espera de detectar el inicio del cordón actual mediante los perfiles obtenidos del SeamPilot. El procedimiento de detección está descrito en 3.1.1, “Descripción del proceso”. Al detectar el inicio del cordón, el sistema espera el tiempo necesario para que la antorcha llegue al inicio del cordón más el solapamiento del cordón (distancia entre el láser y la antorcha más el solapamiento, todo dividido entre la velocidad de la virola), y luego inicia el proceso de cambio de cordón. El solapamiento es un valor que el operador puede ajustar mediante dos botones y un display que muestra el valor actual.

El procedimiento de cambio del cordón consiste de momento en avisar encendiendo la luz de “final de cordón”, y haciendo sonar el zumbador de modo continuo, hasta que al sistema se le informe de que se ha realizado el cambio por pulsación de uno de los botones que definen la zona del cordón. En el futuro se podrá realimentar la zona real del cordón con las entradas previstas al efecto.

En este modo la luz “modo automático” permanece encendida.

En principio el error de profundidad elevada sólo ocurre cuando la máquina está sin posicionar, y nunca en soldadura. De todos modos, si la soldadura ha comenzado se continúa, pero pasando al submodo “prof_elevada”, y señalizando el error y el zumbador, como indicación al operador de que la máquina no puede controlar la profundidad.

Se sale de este modo pasando al modo manual en los casos de error descritos arriba, o cuando el operador pulsa el botón de modo manual.

3.4.3.- Funciones comunes a los modos de operación manual y automático

El sistema obtiene perfiles de la SeamPilot. Si no hay comunicación correcta con el SeamPilot o éste indica que existe algún error se ilumina la luz “Error SeamPilot” durante un segundo, o mientras dure la condición de error, lo que más tarde ocurra.

Cuando el SeamPilot entra en la zona de alcance de la poceta y es capaz de medir perfiles, se enciende la luz “Perfil adquirido”. Esta luz se apaga durante un segundo si se detecta algún perfil incorrecto.

Si se detecta un tope hardware se activa la alarma correspondiente. Asimismo el controlador no intentará movimientos en la dirección de ese tope.

Cuando se detecta la señal del detector de proximidad, ésta se utilizará para medir y actualizar la velocidad de la virola, a no ser que el valor obtenido esté fuera de los valores normales (configurables).

Si se detecta la pulsación de uno de los botones de subir o bajar el valor de alguno de los displays/contadores, el de distancia láser-antorcha o el de solapamiento del cordón, se realizará la modificación solicitada.

Las luces de “zona de cordón” se iluminarán de acuerdo con lo indicado por las señales de realimentación.

Mientras se mantenga pulsado el botón “prueba luces”, se encenderán todas las luces.

Los errores “cancelado modo auto”, “profundidad escasa”, “tope de recorrido”, y “otros errores” deben ser reconocidos cuando se producen. En estos casos, la luz del error y la de reconocimiento parpadean. Cuando se pulsa el botón de reconocimiento, la luz del error permanece encendida mientras dure la condición del error, y la luz de reconocimiento se apaga.

El error “otros errores” se utiliza para indicar errores que el software detecte.

3.4.4.- Modo emergencia

En este modo se inhiben los servos y el sistema permanece inactivo hasta que la seta de emergencia se libera. En ese momento se habilitan los servos y se pasa a modo manual. En este modo las luces de modo manual y automático permanecen apagadas.

3.5.- Interfaz hombre-máquina

Se detallan en la figura 3.6 las luces y botones que constituyen la interfaz hombre máquina.

Los interruptores no son leídos por el controlador. El interruptor de los displays habilita o inhibe por hardware los botones de cambio. El interruptor del zumbador lo habilita o inhibe también por hardware.

El sistema podrá tener una pantalla conectada donde se reflejen datos del proceso y posibles mensajes de error. La información a presentar en esta pantalla será:

- Posición (profundidad y desplazamiento) de la poceta
- Consigna de posición (profundidad y desplazamiento)
- Modo de operación y zona del cordón
- Velocidad del virador
- Puntos significativos (profundidad y desplazamiento, profundidades de las tres zonas de cordón) del perfil obtenido por el seampilot.
- Últimos mensajes de error

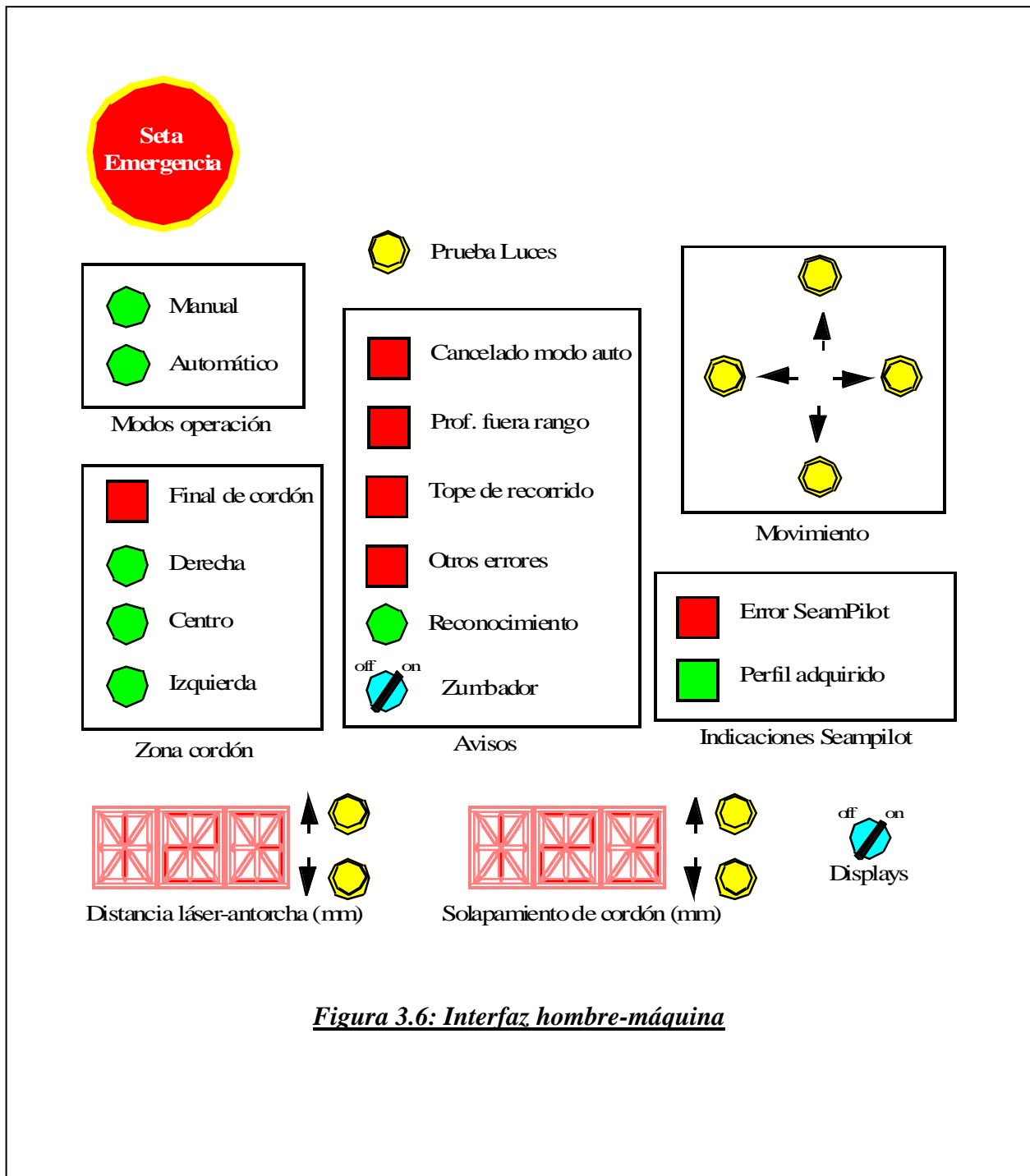


Figura 3.6: Interfaz hombre-máquina

3.6.- Estructura global del controlador

La figura 3.7 muestra un esquema general de los módulos en que se ha dividido el software del controlador, así como las principales relaciones entre esos módulos.

A continuación se describen brevemente cada uno de los módulos del primer nivel de descomposición del sistema, posteriormente se describirá cada uno de ellos de forma más detallada.

- **Configuración:** En este módulo se describen todos los parámetros del sistema que son configurables. Aunque muchos de estos parámetros corresponden a otros módulos del sistema, tales como los mandos o los motores, se ha preferido agruparlos todos en un mismo módulo con objeto de simplificar la labor de configurar la aplicación del láser. Casi todos los módulos del sistema leen los parámetros configurables del módulo configuración.
- **Mandos:** Este módulo engloba todos los elementos de la unidad de mandos de la aplicación, en concreto la botonera, las luces de la botonera, y la seta de emergencia. El módulo proporciona operaciones para leer el estado de los mandos, así como para poder encender o apagar las diferentes luces del cuadro de mandos.
- **Máquina:** Este módulo engloba los elementos de la máquina de sistema de soldadura. Los elementos se agrupan en dos tipos: sensores, cuyos valores se pueden leer, y actuadores, sobre los que se puede actuar. Los sensores son los topes de recorrido de cada eje, la indicación de máquina soldando, y la realimentación de la zona del cordón actual. Los actuadores son los que permiten especificar las consignas de los servos, la salida de habilitación de los mismos, y la actuación de la zona del cordón.
- **Seampilot:** Este módulo contiene las operaciones de comunicación con el Seampilot que permiten obtener los perfiles y, a partir de ellos, el desplazamiento y profundidad actuales de la soldadura.
- **Virola:** El módulo gestiona el sensor de proximidad que sirve para medir la velocidad de giro.
- **Gestor de Mandos:** Este módulo gestiona las órdenes que el usuario introduce a través de la unidad de mandos. En concreto determina el modo de funcionamiento del controlador, gestiona los cambios de modo de funcionamiento, y almacena órdenes que el planificador debe ejecutar. También gestiona las luces del cuadro de mandos, encendiéndolas y apagándolas en función del estado y de las alarmas existentes.
- **Planificador:** Este módulo se encarga de planificar las acciones a realizar por el controlador. Como entradas, recibe el estado actual y las órdenes —que provienen del gestor de mandos— y la posición actual de la antorcha de soldadura que proviene del Seampilot. Como salida proporciona periódicamente consignas a los motores.
- **Reportero:** Este módulo se encarga de leer periódicamente el estado del sistema, y presentarlo en un monitor.
- **Alarmas:** Este módulo se encarga de almacenar el estado de las alarmas del sistema. Otros módulos pueden modificar este estado, o leerlo.

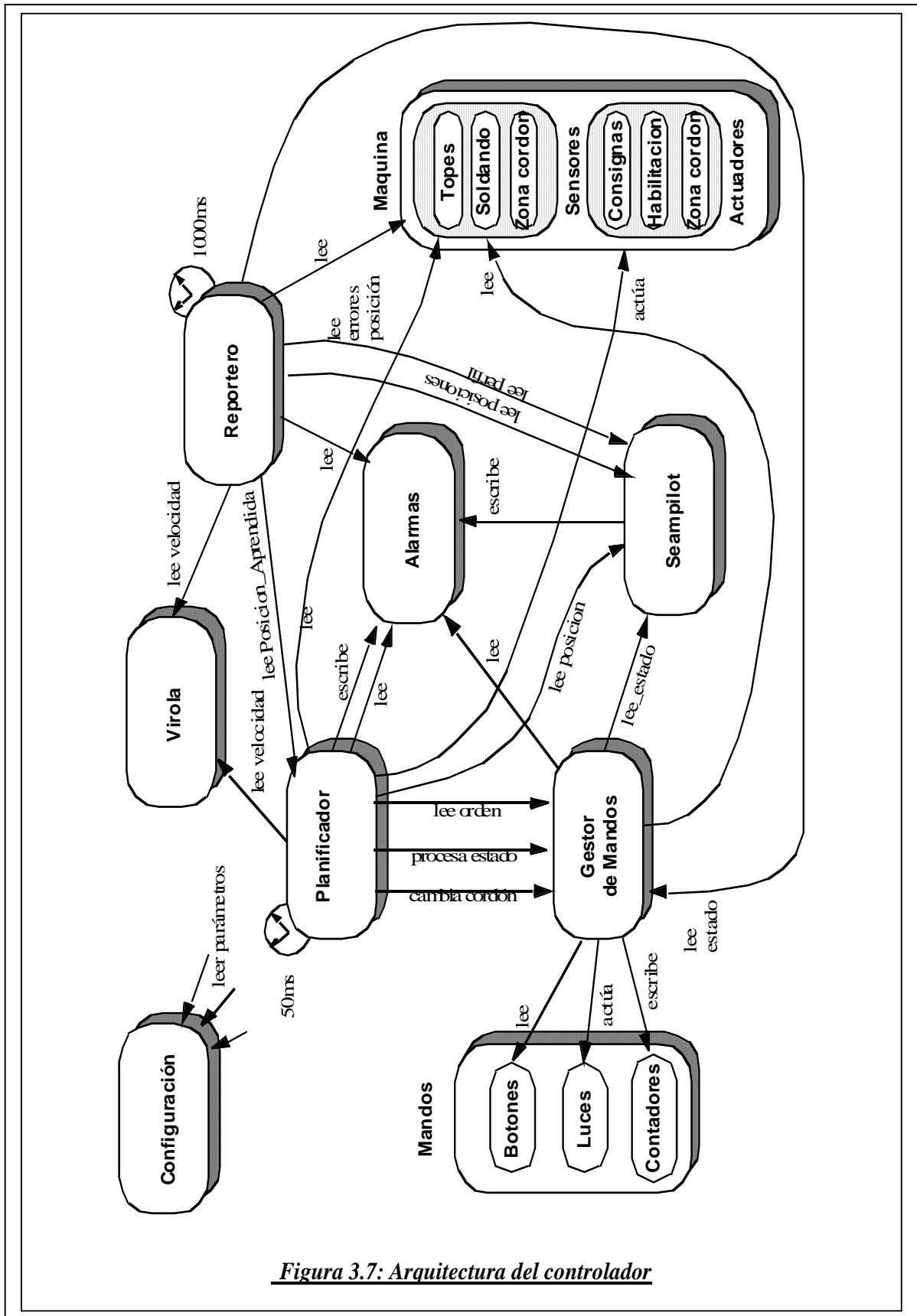


Figura 3.7: Arquitectura del controlador

3.7.- Descripción de los módulos

3.7.1.- Configuración

3.7.1.1.- Descripción

El módulo de configuración permite definir en un lugar único y fácilmente modificable todos los parámetros del sistema que se consideran susceptibles de modificación. Cada parámetro se define mediante una variable con un valor por defecto asignado. Esta variable no es directamente visible desde el exterior, sino que se suministran operaciones para leer o utilizar su valor. Los parámetros de configuración no cambian una vez que el programa ha arrancado.

El módulo de configuración define, además, los tipos más comúnmente utilizados en el sistema. Se divide en tres partes:

- Contiene las medidas y parámetros configurables del sistema de soldadura y de su entorno
- Contiene las operaciones y tipos básicos relacionados con las señales digitales
- Contiene las operaciones y tipos básicos relacionados con las señales analógicas

3.7.1.1.- Operaciones que soporta

3.7.1.1.2.- Entorno

- `Profundidad_Minima`: Retorna la profundidad de la poceta para seguir en modo automático.
- `Max_Perfiles_Erroneos`: Retorna el máximo número de perfiles con error
- `Vel_Max`: Retorna las velocidades máximas de los motores
- `Vel_Movimiento`: Retorna la velocidad de cambio de desplazamiento y profundidad al pulsar los botones de movimiento
- `Consigna_Servos_Manual`: Retorna la consigna servos a utilizar en desplazamiento manual.
- `Banda_Central`: Retorna la banda central usada para el cálculo de profundidad media
- `Num_Muestras_Prof_Media`: n° de muestras usado para el cálculo de la profundidad media.
- `Num_Muestras_Fin_Cordon`: número de muestras usado para el cálculo del final del cordón.
- `Offset_Seampilot`: Indica el offset de desplazamiento y de profundidad utilizado al convertir unidades del perfil leído por el SeamPilot a unidades del controlador.
- `Altura_Cordon`: Retorna la altura del cordón, usado para detectar el inicio del mismo
- `Posicion_Cordon`: posiciones de las zonas de los cordones izquierdo, central y derecho
- `Ancho_Cordon`: Retorna la anchura de la medida de la altura de cada cordón.
- `Offset_Sensor_Antorcha_Inicial`: Retorna el valor por omisión de la distancia del sensor láser a la antorcha de soldadura
- `Solapamiento_Cordon_Inicial`: Retorna el valor del solapamiento inicial del cordón.
- `Vel_Max_Virola`: Retorna la velocidad máxima de la virola (mm/s).
- `Vel_Min_Virola`: Retorna la velocidad mínima de la virola (mm/s).
- `Dist_Marcas_Virola`: Retorna la distancia entre dos marcas de la virola para medir la velocidad de la misma.
- `Periodo_Planificador`: Retorna el periodo de la tarea del planificador
- `Periodo_Reportero`: Retorna el periodo de la tarea del reportero
- `Periodo_Virola`: Retorna el periodo de la tarea de medida de la velocidad de la virola
- `Kp`: Retorna la constante de control proporcional.
- `Kd`: Retorna la constante de control derivativo.
- `Ki`: Retorna la constante de control integral.
- `max_integral`: Retorna el valor máximo de la componente integral.

Asimismo se dispone de las siguientes constantes:

- `Prioridad_Planificador`: Prioridad de la tarea del planificador
- `Prioridad_Reportero`: Prioridad de la tarea del reportero
- `Prioridad_Gestion_Luces`: Prioridad de la tarea de gestión de luces, en el módulo de luces
- `Prioridad_Virola`: Prioridad de la tarea de lectura de la velocidad de la virola
- `Prioridad_Gestion_SeamPilot`: Prioridad de la tarea de la tarea de gestión de los datos del seam-pilot
- `Prioridad_Receptor_SeamPilot`: Prioridad de la tarea `Receptor_Seampilot` que se encuentra en el módulo del protocolo de comunicación (`Envia_Recibe`).
- `Prioridad_Receptor_Puerto_Serie`: Prioridad de la tarea de recepción de caracteres del puerto serie (`Receptor_Puerto_Serie`), que se encuentra en el módulo del protocolo de comunicación (`Envia_Recibe`).
- `Prioridad_Gestion_Contadores`: Prioridad de la tarea de gestión de los contadores de los displays gráficos.
- `Techo_Seampilot_Datos_Perfil`: Techo de prioridad del objeto protegido que almacena los datos del perfil obtenido del Seam-pilot.
- `Techo_Envia_Recibe_Cola`: Techo de prioridad del objeto protegido que almacena los mensajes recibidos del Seam-pilot
- `Techo_Envia_Recibe_Estado`: Techo de prioridad del objeto protegido que almacena y gestiona el estado de las comunicaciones con el Seam-pilot.
- `Techo_Alarmas`: Techo de prioridad del objeto protegido que almacena las alarmas.
- `Techo_Virola`: Techo de prioridad del OP que almacena la velocidad de la virola.
- `Techo_Maquina`: Techo de prioridad del OP que almacena los datos relativos a los motores.
- `Techo_Luces`: Techo de prioridad del OP que almacena el estado de las luces.
- `Techo_Botones`: Techo de prioridad del OP que almacena el estado de los botones.
- `Techo_Contadores`: Techo de prioridad del OP que almacena el estado de los contadores.

3.7.1.1.3.- IO Digital

- `Valor_Inicial_Salida_Digital`: Devuelve el valor Inicial de la cada salida (0 o 1). Indica el valor al que se colocará la salida al arrancar el sistema.
- `Valor`: Función que transforma un estado digital (`asertado` o `no_asertado`) en un valor digital (0 o 1), para la salida digital indicada.
- `Estado`: Función que transforma un valor digital (0 o 1) en el correspondiente estado digital, para la entrada digital indicada.

3.7.1.1.4.- IO Analógica

- `Max_Salida_Analogica`: Devuelve el valor máximo Analógico, en unidades de convertidor.
- `Min_Salida_Analogica`: Devuelve el valor mínimo Analógico, en unidades de convertidor.
- `Valor_Inicial_Salida_Analogica`: Devuelve el valor Inicial de la cada salida, en unidades de convertidor. Indica el valor al que se colocará la salida al arrancar el sistema.
- `Limite_Sup_Salida_Analogica`: Devuelve el límite superior de cada salida analógica, en unidades de convertidor. Indica el subrango de los valores del convertidor que se permitirán en la práctica.
- `Limite_Inf_Salida_Analogica`: Devuelve el límite inferior de cada salida analógica, en unidades de convertidor. Indica el subrango de los valores del convertidor que se permitirán en la práctica.

3.7.2.- Mandos

3.7.2.1.- Descripción

Este módulo se encarga de la gestión a bajo nivel de la unidad de mandos. Está dividido en tres submódulos, que realizan las siguientes funciones:

- **Botones.** Contiene operaciones para leer el estado de los botones. Se puede leer el valor instantáneo, o el hecho de que un botón haya sufrido un cambio desde la última vez que se leyó, o haya sido pulsado desde la última vez que se leyó. Este módulo almacena como estado el valor del botón cuando se leyó la última vez.
- **Luces:** Contiene operaciones para encender o apagar las luces de forma absoluta, o para encender las luces en modo de parpadeo, o de pulso. El módulo contiene una tarea que se encarga de los aspectos temporales del parpadeo o del pulso. Esta tarea guarda como estado el modo en el que esta la luz, y los parámetros temporales necesarios para implementar las operaciones. Se incluye también el zumbador que, aunque no es una luz, tiene un tratamiento semejante
- **Contadores:** Contiene operaciones para dar valor a los displays/contadores de la unidad de mandos, y para poder modificar sus valores incrementándolos o decrementándolos.

3.7.2.2.- Operaciones que soporta

3.7.2.2.1.- Botones

- **Botones.Lee_Estado_Actual:** Permite obtener el estado actual (pulsado o no pulsado) del botón indicado.
- **Botones.Lee_Todos:** Permite obtener el estado actual de todos los botones del mando externo. Se ofrece para optimizar el tiempo de acceso en caso de necesitar leer el estado de muchos botones.
- **Botones.Detecta_Cambio:** Permite conocer si ha habido algún cambio en el estado del botón indicado (y su estado actual), desde la última vez que se leyó. A este efecto se considera que el botón se lee con cualquiera de las operaciones `Lee_Estado_Actual`, `Detecta_Cambio`, o `Detecta_Pulsacion`, pero no con `Lee_Todos`.
- **Botones.Detecta_Pulsacion:** Permite conocer si el botón indicado ha pasado del estado `no_pulsado` a `pulsado` desde la última vez que se leyó. A este efecto se considera que el botón se lee con cualquiera de las operaciones `Lee_Estado_Actual`, `Detecta_Cambio`, o `Detecta_Pulsacion`, pero no con `Lee_Todos`.

3.7.2.2.2.- Luces

- **Luces.Enciende:** La luz indicada permanece encendida a partir de esta llamada.
- **Luces.Apaga:** La luz indicada permanece apagada a partir de esta llamada.
- **Luces.Esta_Apagada:** Devuelve verdad si la luz indicada está apagada y en estado normal; devuelve falso en caso contrario.
- **Luces.Enciende_Temporizada:** A partir de esta llamada, la luz permanece encendida durante el tiempo indicado. Una vez transcurrido ese tiempo, la luz se apaga, a no ser que previamente se realice otra llamada para encender o apagar las luces.
- **Luces.Enciende_con_Parpadeo** (2 parámetros). A partir de esta llamada, la luz pasa al estado de parpadeante. En este estado, la luz se enciende y se apaga a intervalos periódicos, según el periodo indicado.

- `Luces.Enciende_con_Parpadeo` (4 parámetros): A partir de esta llamada, la luz pasa al estado de parpadeante durante el intervalo especificado por `Durante`. En este estado, la luz se enciende y se apaga a intervalos periódicos, según el periodo indicado. Al finalizar el intervalo, la luz se queda encendida o apagada (fija) según el valor indicado por `Valor_Final`.
- `Luces.Inicia_Prueba_Luces`. Esta operación enciende todas las luces del mando externo, independientemente de su estado actual. Las luces permanecen encendidas hasta que se invoca la operación `Luces.Finaliza_Prueba_Luces`. El estado de las luces no cambia con esta operación.
- `Luces.Finaliza_Prueba_Luces`. Esta operación devuelve todas las luces del mando externo a su estado habitual, si estaban todas encendidas por invocación de la operación `Luces.Inicia_Prueba_Luces`; si no, la operación no tiene ningún efecto.

3.7.2.2.3.- Contadores

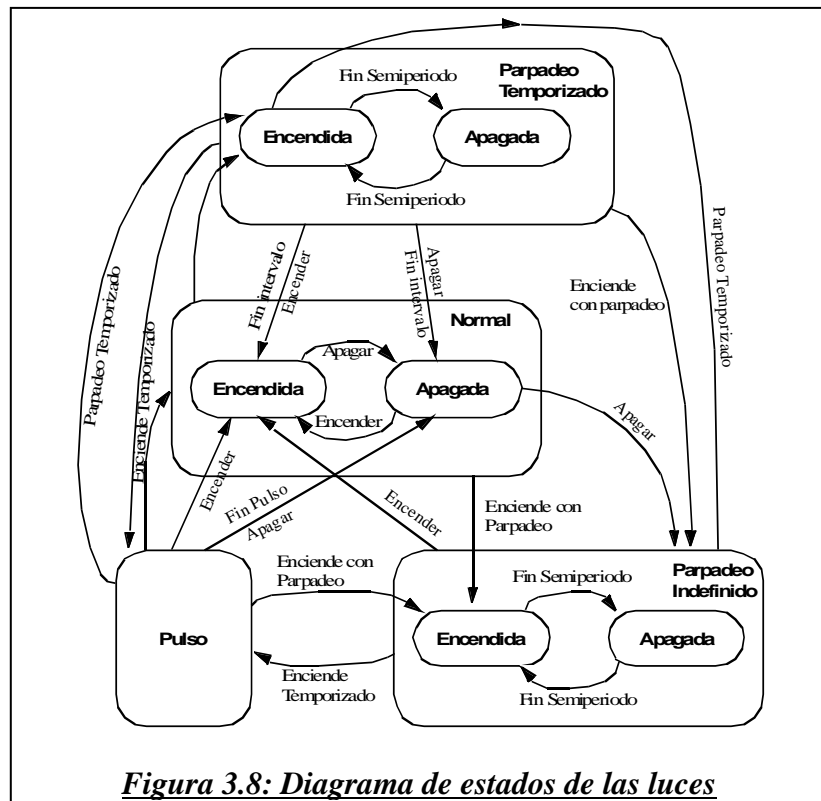
- `Contadores.Pon_valor`: Permite poner el contador elegido a un valor concreto.
- `Contadores.Incrementa`: Incrementa en una unidad el valor actual del contador elegido.
- `Contadores.Decrementa`: Decrementa en una unidad el valor actual del contador elegido.
- `Contadores.valor_Actual`: Retorna el valor actual del contador elegido

3.7.2.3.- Descripción Detallada

Estado de cada una de las luces (figura 3.8)

- *Normal*: La luz no está temporizada. Puede estar encendida, o apagada.
- *Pulso*: La luz está temporizada en forma de pulso. Se almacena el tiempo que resta hasta finalizar el pulso. Al finalizar el pulso, la luz pasa al estado normal, apagada.
- *Parpadeo Indefinido*: La luz está parpadeando. Puede estar encendida o apagada. Se almacena el semiperiodo, y el tiempo que resta hasta el siguiente cambio de estado.
- *Parpadeo Temporizado*: La luz esta parpadeando durante un intervalo de tiempo. Puede estar encendida o apagada. Se almacena el semiperiodo, el tiempo que resta hasta el siguiente cambio de estado, el tiempo que resta hasta el final del intervalo, y el estado final en el que se esta la luz.

Las operaciones de cambio de estado estarán protegidas para que el acceso sea mutuamente exclusivo, mediante un objeto protegido. Además de las operaciones básicas de la interfaz, existirá una operación invocada periódicamente, para implementar la temporización. Una tarea se encarga de realizar esta llamada periódica. Su periodo, llamado `Per_Control_Luces`, será inicialmente de 100 ms, y define el tiempo mínimo especificable para el semiperiodo o el pulso de una luz.



3.7.3.- Gestor de Mandos

3.7.3.1.- Descripción

Este objeto pasivo se encarga de leer los botones de la unidad de mandos y de calcular en función de los botones pulsados, así como de las alarmas y las señales de otros sensores leídos, el estado del sistema. Se encarga también de gestionar las luces de la unidad de mandos. Sus operaciones deben estar protegidas para que los accesos sean mutuamente exclusivos.

3.7.3.2.- Operaciones que soporta

- `Procesa_Estado`: Esta operación causa la lectura de la botonera y de las alarmas, el cálculo del nuevo estado del sistema, la gestión de las luces, y la ejecución de órdenes (prueba de lámparas). Almacena la última orden producida, para su lectura mediante `Lee_Orden`. También lee los errores del módulo Alarmas, y coloca las luces al valor especificado para los errores. Retorna en el parámetro de salida el nuevo estado.
- `Lee_Estado`: Esta operación retorna el estado actual de los mandos.
- `Lee_Orden`: Lee la última orden almacenada y la borra.
- `Cambia_Cordon`: Pone a `true` la variable de estado `Cambiando_Cordon`.

3.7.3.3.- Tratamiento de Alarmas

Activación y desactivación de alarmas cuando se llama a `Lee_Estado`:

- `Seta_Emergencia`: Si se activa la seta de emergencia, se activa esta alarma. Cuando la seta se desenchava, la alarma se desactiva.
- `Cancelado_Modo_Auto` y `Error_Software`: Cuando la alarma está activada, si se pulsa el botón de reconocimiento, la alarma se reconoce e inmediatamente después se desactiva.
- `Tope_Hardware`, `Profundidad_elevada`, y `Profundidad_Escasa`: Cuando la alarma está activada, si se pulsa el botón de reconocimiento, la alarma se reconoce. Tratamiento de alarmas, cada vez que se invoque `Lee_Estado`:
 - `Cancelado_Modo_Auto`, `Tope_Hardware`, `Profundidad_Escasa`, `Profundidad_elevada`, y `Error_Software`: Mientras la alarma está activada, parpadea la luz de alarma y la de reconocimiento. Mientras está reconocida, permanece encendida. Mientras está desactivada, se queda apagada.
 - `Error_Seampilot`: Mientras la alarma está activada, se ilumina la luz correspondiente. Cuando se desactiva, si ha transcurrido más de un segundo encendida se apaga, y si no se apaga al transcurrir un segundo desde que se activó.
- `Seta_de_Emergencia`: Mientras la alarma está presente el campo `Emergencia` del estado del sistema se hace igual a `True` y las luces del modo de operación se apagan. Cuando la alarma se desactiva se hace igual a `False`.

3.7.4.- Máquina

3.7.4.1.- Descripción

Este objeto presenta operaciones para la lectura de los sensores de la máquina de soldadura, así como para la actuación sobre los servos y otros actuadores. Para la realización de estas tareas interactúa directamente con los drivers de E/S. Está dividido en dos submódulos: Sensores, y Actuadores. Las operaciones deberán de estar protegidas para que los accesos sean mutuamente exclusivos.

3.7.4.1.- Operaciones que soporta

3.7.4.1.1.- Sensores

- `Sensores.Lee_Zona_Cordon`: Devuelve la zona del cordón indicada por las entradas de zona de cordón.
- `Sensores.Lee_Topes_Superiores`: devuelve el estado de los topes superiores.
- `Sensores.Lee_Topes_Inferiores`: devuelve el estado de los topes inferiores.
- `Sensores.Lee_Servos_OK`: devuelve el valor (encendido o apagado) que indica si los servos están habilitados o no.
- `Sensores.Lee_Soldando`: devuelve el valor (encendido o apagado) del relé que conecta la alimentación del brazo.

3.7.4.1.2.- Actuadores

- `Actuadores.Habilitacion_Servos`: Permite habilitar o inhibir los servos.
- `Actuadores.Controla_Servos`: Permite indicar la consigna que se suministrará a los servos durante el siguiente periodo de muestreo.
- `Actuadores.Pon_Zona_Cordon`: Permite actuar sobre las salidas de la zona del cordón, poniéndolas al valor indicado por `Zona_Cordon`.

3.7.5.- Alarmas

3.7.5.1.- Descripción

El objeto Alarmas se encarga de almacenar el estado de las alarmas que se pueden producir en el sistema. Es un objeto protegido frente a accesos simultáneos. Las alarmas pueden estar activadas, reconocidas (que implica que está activada, pero que ya ha sido reconocida por el usuario), desactivadas, y en el estado “desactivar al reconocer”. Algunas alarmas deben de ser reconocidas antes de ser desactivadas; esto lo indica la constante *Debe_Ser_Reconocida*. El diagrama de estados de las alarmas se muestra en la figura 3.9

3.7.5.2.- Operaciones que soporta

- *Lee*: Permite leer el estado de la alarma indicada.
- *Lee_Todas_Alarmas*: Permite leer de forma simultánea el estado de todas las alarmas.
- *Activa*: Permite activar la alarma indicada.
- *Reconoce*: Permite reconocer la alarma indicada. Si se reconoce una alarma desactivada o si la alarma no debe ser reconocida se eleva *operacion_incorrecta*.
- *Desactiva*: Permite desactivar la alarma indicada, o pasarla al estado *Desactivar_Al_Reconocer* si es una alarma de las que deben ser reconocidas.
- *Inserta_error*: Permite insertar un error, junto al mensaje correspondiente. Este error se almacena en una cola para su posterior extracción por parte del Reportero mediante *Extrae_Error*. Además, si el error es grave se activa la alarma *error_software*. Con objeto de evitar repetir muchas veces el mismo error, si en la cola ya hay mensajes de error y el nuevo error coincide con el último error insertado, no se almacenará. Si el error no cabe en la cola, se elimina el más antiguo.
- *Extrae_error*: Permite extraer un error, si lo hay, junto a su mensaje asociado, de la cola de errores insertados mediante *inserta_error*. Si no hay error almacenado, *Hay_Error* se devolverá con valor falso, y los otros dos parámetros no tendrán valores válidos.

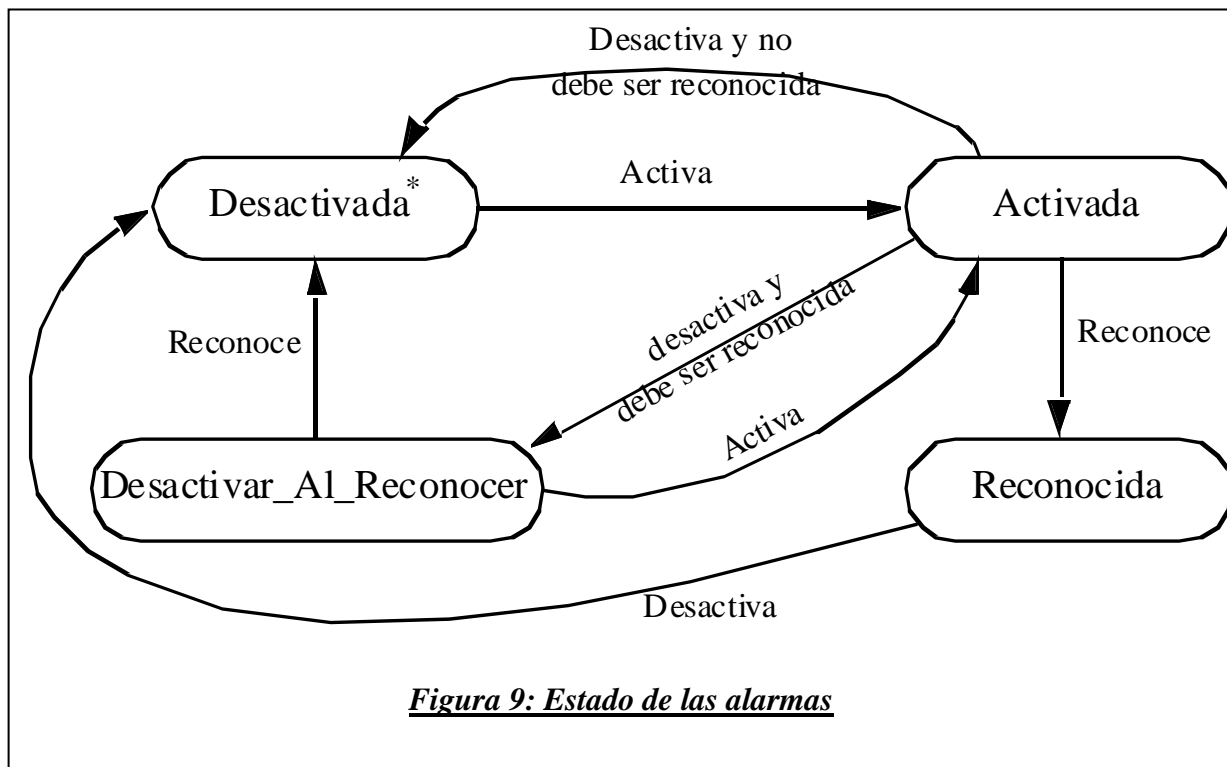


Figura 9: Estado de las alarmas

3.7.6.- Planificador

3.7.6.1.- Descripción

Este módulo se encarga de controlar las acciones del controlador de la máquina de soldadura. Para ello toma como entradas el modo de control (devuelto por el módulo `Gestor_Mandos`), la posición actual de la antorcha indicada por el `Seampilot`, la velocidad de la virola indicada por el módulo `Virola`, el estado de alarmas, y el estado de los sensores de la máquina. En función de estas entradas calcula las consignas de los servos para el próximo periodo de planificación (indicado por el parámetro configurable `Entorno.Periodo_Planificador` e inicialmente situado en 50 ms.). Estas consignas se escriben en los actuadores del módulo `Maquina`.

3.7.6.2.- Tratamiento de Alarmas

- Activa `Tope_Hardware` cuando detecta que se ha alcanzado un tope hardware, y la desactiva en caso contrario si ha sido reconocida.
- Activa `Cancelado_Modo_Auto` cuando esta en modo automático y detecta que los últimos perfiles son erróneos.

3.7.6.3.- Descripción detallada

El planificador de trayectorias contiene la tarea `Planificador`, que es una tarea periódica cuyo funcionamiento depende fuertemente del estado del sistema. La tarea del planificador en su parte inicial recaba información de todas las partes del sistema (`Gestor de mandos`, `Seampilot`, `Virola`, `Maquina` y `Alarmas`) para en función de los datos obtenidos tomar las decisiones adecuadas.

Los datos que inicialmente se obtienen son los siguientes:

- Del gestor de mandos lee el estado actual del sistema y las posibles órdenes que se le quieran dar a través de la botonera.
- Del `Seampilot` lee la posición en la que se encuentra el sistema.
- De la virola lee la velocidad a la que se esta moviendo.
- De la máquina lee si se ha alcanzado algún tope para evitar el desplazamiento en ese sentido y dirección.
- De las alarmas lee el estado de todas las alarmas del sistema.

La tarea una vez tomados los datos y en función del modo de la operación tomará decisiones para planificar las trayectorias:

Modo emergencia: Se tendrán apagados los servos.

Modo manual: Se habilitan los servos y en función de la orden que se le de a través de la botonera se pone una consigna u otra para que los servos respondan en consecuencia.

Modo automático: Si se entra por primera vez se aprende la posición que será la que salvo modificación (a través de la botonera con las ordenes de desplazamiento) sea la que se intente restablecer cuando ocurran variaciones. Si estando en el modo automático se superan el número máximo de perfiles erróneos, se pasa al modo manual. Si los perfiles son correctos y no se activa ninguna alarma comenzará el algoritmo del controlador de trayectorias que según en el submodo en el que nos encontremos controlará desplazamiento (en el submodo profundidad elevada), profundidad (en el submodo profundidad escasa) o ambas si está en modo normal. Si estando en modo automático se introducen órdenes de desplazamiento se modificará la posición aprendida.

Si se detecta un final de cordón es esta tarea la que en función de la distancia láser-antorcha retrasa el encendido de la luz que notifica que se ha llegado al final del cordón Para finalizar, la tarea pone la consigna adecuada a los servos.

3.7.7.- Virola

3.7.7.1.- Descripción

Este paquete define el objeto `Virola`, que encapsula las operaciones de medida de la velocidad .

3.7.7.2.- Operaciones que soporta

- `Lee_Velocidad`: Retorna `Valida` a `false` antes de la primera medida, y si ha transcurrido demasiado tiempo sin medir, y `true` en caso contrario. Si la medida es válida, retorna en `Medida` la velocidad actual de la virola.

3.7.7.3.- Descripción Detallada

A la velocidad máxima de la virola (7mm/seg) el tiempo de paso desde una marca del sensor de proximidad hasta la siguiente es de unos 28 segundos, por lo que para conseguir la precisión de la medida del 1% es posible usar una tarea de muestreo periódico, por ejemplo con periodo de 1 ms, que mida los tiempos. Por ello, el módulo `Virola` tendrá internamente una tarea periódica que irá detectando el tiempo de paso por las marcas y actualizando así la velocidad.

3.7.8.- Seampilot

3.7.8.1.- Descripción

Define el objeto `Seampilot`, que encapsula las operaciones de adquisición de perfiles láser.

3.7.8.2.- Operaciones que soporta

- `Lee_Estado`: Retorna el estado actual del `Seampilot`, un booleano que indica si se ha detectado un cambio de cordón desde la última vez que se llamó a `Lee_Posicion`, y un valor que indica si la profundidad de la poceta es correcta, escasa, o elevada.
- `Lee_Posicion`: Retorna el estado actual del `Seampilot`, un booleano que indica si se ha detectado un cambio de cordón desde la última vez que se llamó a `Lee_Posicion`, y un valor que indica si la profundidad de la poceta es correcta, escasa, o elevada. También retorna la posición de la poceta y la altura de los tres cordones, en caso de que el estado sea `Hay_Perfil`.
- `Lee_Posiciones`: Retorna los puntos que delimitan los segmentos horizontales detectados de los segmentos Izquierdo, Central y Derecho para poder representarlo con el Reportero.
- `Lee_Posicion_Aprendida`: función que retorna la posición que se toma como referencia para realizar el guiado, esta posición es la que se obtiene la primera vez que se entra en modo automático con un perfil correcto y se puede modificar desde este modo con los controles de Desplazamiento y Profundidad de la botonera.
- `Escribe_Posicion_Aprendida`: procedimiento que escribe la posición que se toma como referencia para el guiado.
- `Lee_Perfil`: Retorna todos los datos de `Lee_Posicion` y, además, el último perfil.

3.7.8.3.- Tratamiento de Alarmas

Activación/desactivación de alarmas:

- `Activa_Error_SeamPilot` cuando se entra en los estados `Error_Comunicacion` o `Error_SeamPilot`. Lo desactiva si no estamos en esos estados.
- `Activa_Profundidad_Escasa` cuando el estado de la profundidad es `Escasa`, y la desactiva cuando el estado es otro.
- `Activa_Profundidad_Elevada` cuando el estado de la profundidad es `Elevada`, y la desactiva cuando el estado es otro.

3.7.8.4.- Descripción Detallada

El módulo Seampilot se encarga de analizar la información de los perfiles obtenidos del sensor láser con objeto de obtener la información sobre la profundidad y desplazamiento de la poceta, la altura de cada cordón, y la detección del final de un cordón de soldadura.

La identificación de la poceta a partir de la información del perfil se hace en base a los patrones que se muestran en la tabla que aparece a continuación, diseñados de acuerdo a las diferentes posibilidades en que se puede encontrar la poceta y el controlador. La columna “H” indica el número de segmentos horizontales, y la “V” el número de segmentos verticales.

Patrón	Figura	H	V	Descripción
Peinado		3	0	Poceta llena: Calcula desplazamiento y profundidad
0		3	0	Poceta casi llena: Calcula desplazamiento y profundidad
1		3	2	Poceta medio llena: Calcula desplazamiento y profundidad
1a		2	2	Poceta medio llena: Calcula desplazamiento y profundidad
1b		2	2	Poceta medio llena: Calcula desplazamiento y profundidad
1x		3	1	Poceta medio llena: Calcula desplazamiento y profundidad
1x		3	1	Poceta medio llena: Calcula desplazamiento y profundidad

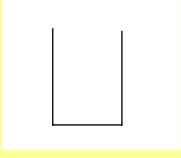

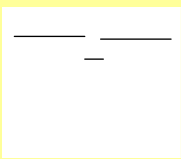
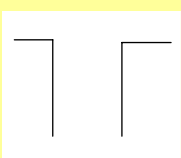
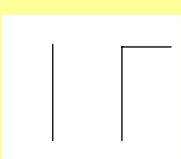
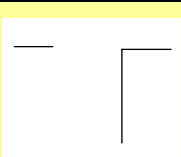
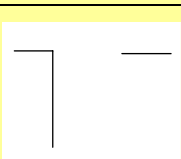
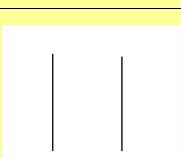
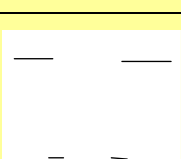
Patrón	Figura	H	V	Descripción
2		1	2	Poceta casi vacía: Calcula desplazamiento y profundidad
3		1	0	Profundidad escasa: No calcula desplazamiento
3b		2	0	Profundidad escasa: No calcula desplazamiento
4		2	2	Profundidad elevada: No calcula profundidad
4a		1	2	Profundidad elevada: No calcula profundidad
4x		2	1	Profundidad elevada: No calcula profundidad
4x		2	1	Profundidad elevada: No calcula profundidad
5		0	2	Profundidad elevada: No calcula profundidad
6		2	0	Profundidad elevada: No calcula profundidad

Tabla I: Patrones de identificación de la poceta

Una de las partes más importantes de esta aplicación es sin duda el reconocimiento de los perfiles obtenidos con el láser y el establecer con que patrón de los antes expuestos coincide.

Este proceso se hace del siguiente modo:

- Se van a intentar obtener segmentos verticales y horizontales del perfil obtenido con el láser, primero intentaremos obtener los verticales que aun no siendo críticos el encontrarlos si facilitara en mucho, el análisis del perfil, así como el determinar con que patrón coincide.
 - Para calcular los segmentos verticales lo primero que hacemos es buscar puntos que se encuentren en la mediana de la profundidad, ya que de haber segmentos verticales estos pasaran por ahí. Una vez encontrado estos puntos (si los hubiese) se van tomando los puntos inmediatamente anteriores y posteriores para intentar formar la línea vertical, para esto los puntos deben de cumplir el criterio de verticalidad, que se basa en la pendiente que tendría la línea que uniese dichos puntos. Una vez encontrados los segmentos verticales y en función de ellos calculamos los horizontales. Si encontramos:
 - 2 segmentos verticales: sabemos que a la izquierda del primero hay un segmento horizontal, a la derecha del segundo otro, y justo entre ambos se encuentra el segmento horizontal central, por lo que esto nos facilitara los cálculos, ya que podremos filtrar los puntos no significativos para el cálculo del Segmento_H_Izq, Segmento_H_Der, Segmento_H_Centro.
 - 1 segmento vertical: primero filtramos donde podría estar el segmento horizontal izquierdo, que serian aquellos que fuesen menores que la posición del segmento vertical menos la anchura de la poceta y el borde de la virola, en esos puntos se intentaría calcular el segmento horizontal izquierdo y posteriormente el central, ya que no sabemos si hemos encontrado el segmento izquierdo o derecho, después se filtrará de igual modo el segmento horizontal derecho para intentar calcularlo, y si no se encontró el central, sabemos que éste se encuentra si lo hubiese en los puntos que se encuentran desde el extremo final del segmento horizontal izquierdo al extremo inicial del derecho.
 - Si no se encuentra ni 1 ni 2 segmentos verticales la forma de analizarlo es simplemente filtrando los lugares donde pueden encontrarse los segmentos laterales que estarán fijados por la anchura de la poceta y la del borde de la virola, y en el centro estar posicionado el segmento central si lo hubiese.

- Una vez que tenemos la identificación de los segmentos obtenidos del perfil comparamos con la Tabla I para ver en que caso estamos, Aunque el hecho de que haya un numero determinado de segmentos horizontales y verticales no nos garantiza de forma univoca un patrón determinado y por ello hay que hacer una serie de comprobaciones adicionales, en función de lo que hayamos encontrado, además de unas comprobaciones para todos ellos como puede ser que el rizado del segmento este dentro de los parámetros establecidos, la anchura de la poceta encontrada...:
 - **H=0**
 - **V=2:** Hay que comprobar que los segmentos verticales tengan una profundidad mínima, y que estén a una distancia adecuada que coincida con la de la poceta.
 - **H=1**
 - **V=1:** Hay que comprobar donde se encuentra el segmento horizontal para saber a que patrón corresponde el perfil, ya que si por ejemplo esta entre los verticales seria un Patron_2, si no lo esta seria un Patron4a. En ambos casos además de esto debería verificarse tanto el rizado de los segmentos, como la altura mínima de los verticales o la distancia entre verticales que fuese coherente con el ancho de la poceta
 - **V=0:** Hay que comprobar que la longitud del segmento sea adecuada.
 - **H=2**
 - **V=2:** Además de comprobar las especificaciones de altura mínima de los segmentos verticales, distancia entre ellos y el rizado hay que comprobar donde se encuentra los segmentos horizontales para saber a que patrón corresponde el perfil, ya que, si por ejemplo, estan ambos por encima de los verticales seria un Patron_4, si solo lo está uno sería un Patron1a o Patron1b, en función de cual de los segmentos horizontales sea el que esta por encima.
 - **V=1:** Hay que comprobar además de lo típico en los segmentos verticales que ambos segmentos horizontales se encuentren por encima del vertical.
 - **V=0:** En este caso hay que comprobar si estamos al principio de la soldadura donde no se ve el fondo de la poceta (Profundidad elevada) Patron_6 o por el contrario estamos al final cuando ya se ha rellenado prácticamente toda la poceta Patron3b, y para ellos solo es necesario comprobar la separación entre los segmentos horizontales.
 - **H=3**
 - **V=2:** Además de comprobar las especificaciones de altura mínima de los segmentos verticales, distancia entre ellos y el rizado hay que comprobar donde se encuentra los segmentos horizontales, ya que los laterales deben de estar por encima de los verticales y el central entre ellos además de estar por debajo.
 - **V=1:** Hay que comprobar además de lo típico en los segmentos verticales que el segmento horizontal central se encuentren dentro de la poceta y los segmentos horizontales laterales por encima del vertical.
 - **V=0:** En este caso hay que comprobar si el segmento horizontal central esta por encima o por debajo de los laterales para saber si estamos peinando la soldadura o aun estamos en un modo normal de soldadura.

Por otro lado, el módulo `Seampilot` utiliza los siguientes módulos auxiliares:

- `Protocolo_SeamPilot`: implementa el protocolo de comunicaciones de alto nivel con el `SeamPilot`.
- `Envia_Recibe`: implementa el protocolo de transporte para el puerto serie, conforme a los requisitos del `Seampilot`

A continuación se detallan cada uno de estos módulos.

3.7.8.5.- Módulo `Protocolo_SeamPilot`

3.7.9.5.1.- Descripción

De las operaciones del módulo `Protocolo_SeamPilot`, las que vamos a usar para la lectura de perfiles son las siguientes:

- `Pon_Modo`: Pone el `Seampilot` en el modo de funcionamiento deseado. En nuestro caso seleccionaremos el modo `ON`, que hace que el `Seampilot` envíe periódicamente al controlador mensajes con los perfiles adquiridos.
- `Configura_Scan`: Permite configurar los parámetros de lectura de perfiles del `Seampilot`. En nuestro caso, la configuración elegida será (por confirmar):
- `Ancho`: Anchura del perfil en grados: `ANCHO_40` (40°)
- `Frec`: Frecuencia de toma de perfiles, en medidas por segundo: `FREC_10` (10 medidas/seg)
- `Unidad`: tipo de unidades: `MILIMETRO`
- `Modo_Fitted_Data`: Modo de funcionamiento de datos ajustados a patrón (*fitted data*): `NO_FITTED_DATA`
- `Frec_Fitted_Data`: Frecuencia de mensajes *fitted data*, en mensajes por seg: `FREC_FITT_DAT_1_M_S` (no se usa)
- `Modo_Filtered_Data`: Modo de funcionamiento de datos filtrados. Se usarán perfiles completos: `TODOS_PTOS`
- `Frec_Filtered_Data`: Frecuencia de envío de perfiles en mensajes por seg: Un mensaje por perfil (`FREC_FILT_DAT_1_M_S`)
- `Espera_Mensaje`: Se usará para leer los mensajes con los perfiles
- `Req_Mensaje`: Solicita al `Seampilot` que envíe un mensaje. Los mensajes que se solicitarán serán:
- `ESTADO_HARDWARE`: Indica el estado del hardware del `Seampilot`.
- `Envia_Mensaje`: Envía al `Seampilot` un mensaje. Los mensajes que se enviarán serán: • `RESET`: Reinicia el `Seampilot`

3.7.8.6.- Módulo Envía_Recibe

3.7.8.6.1.- Descripción

Las operaciones de este módulo son las siguientes:

- **Configura:** Configura el puerto serie con los parámetros necesarios para la comunicación con el Seampilot
- **Reconfigura:** Cierra el puerto serie y lo vuelve a configurar con los parámetros necesarios para la comunicación con el Seampilot
- **Envía_Mensaje:** Envía el mensaje indicado al Seampilot. La operación finaliza cuando el mensaje ha sido enviado.
- **Recibe_Mensaje:** Recibe un mensaje del Seampilot. Si no hay mensajes disponibles, suspende a la tarea que invoca la operación, hasta que haya un mensaje disponible.
- **Recibe_Mensaje_No_Bloqueante:** Recibe un mensaje del Seampilot si lo hay disponible. En caso contrario, retorna de inmediato indicando que no hay mensaje disponible. Internamente, el módulo dispone de un objeto protegido y una tarea. El objeto protegido almacena una cola donde la tarea deposita los bytes recibidos; además, contiene el estado de las comunicaciones, que responde al diagrama de estados que se define en la documentación del Seampilot.

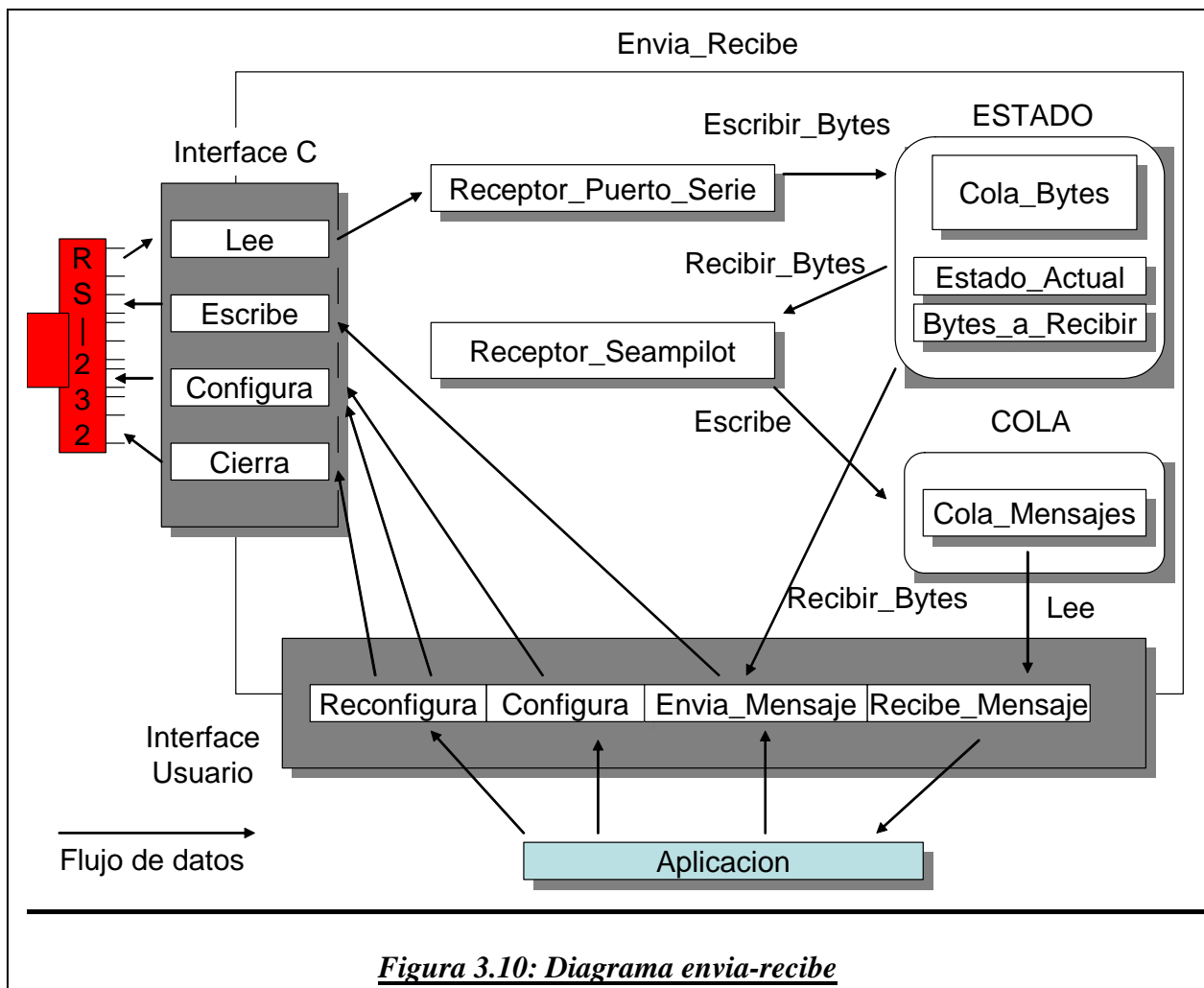


Figura 3.10: Diagrama envía-recibe

3.7.10.- Reportero

3.7.10.1.- Descripción

Este módulo se encarga de recopilar el estado del sistema consultando a los diferentes objetos que lo componen, para posteriormente mostrarlo en la pantalla. Los parámetros a mostrar están especificados en el documento de especificación, y son los siguientes:

- Posición (profundidad y desplazamiento) de la poceta
- Consigna de motores
- Modo de operación y zona del cordón
- Velocidad del virador
- Puntos significativos (profundidad y desplazamiento, profundidades de las tres zonas de cordón) del perfil obtenido por el seampilot.
- Punto Aprendido (punto de referencia para el guiado)
- Últimos mensajes de error
- Estado de alarmas. A: activa, R. reconocida, espacio en blanco: inactiva Cuando el sistema disponga de sistema de ficheros, los errores notificados se almacenarían en un fichero, que podría ser consultado para determinar las causas de un error.

3.7.10.2.- Descripción Detallada

El módulo reportero tiene una tarea denominada Reportero, que ejecuta de forma periódica, con el periodo indicado por el parámetro configurable Entorno.Periodo_Reportero. En cada periodo la tarea escribe el estado del sistema en la pantalla. La configuración de la pantalla será la indicada en la figura 3.11

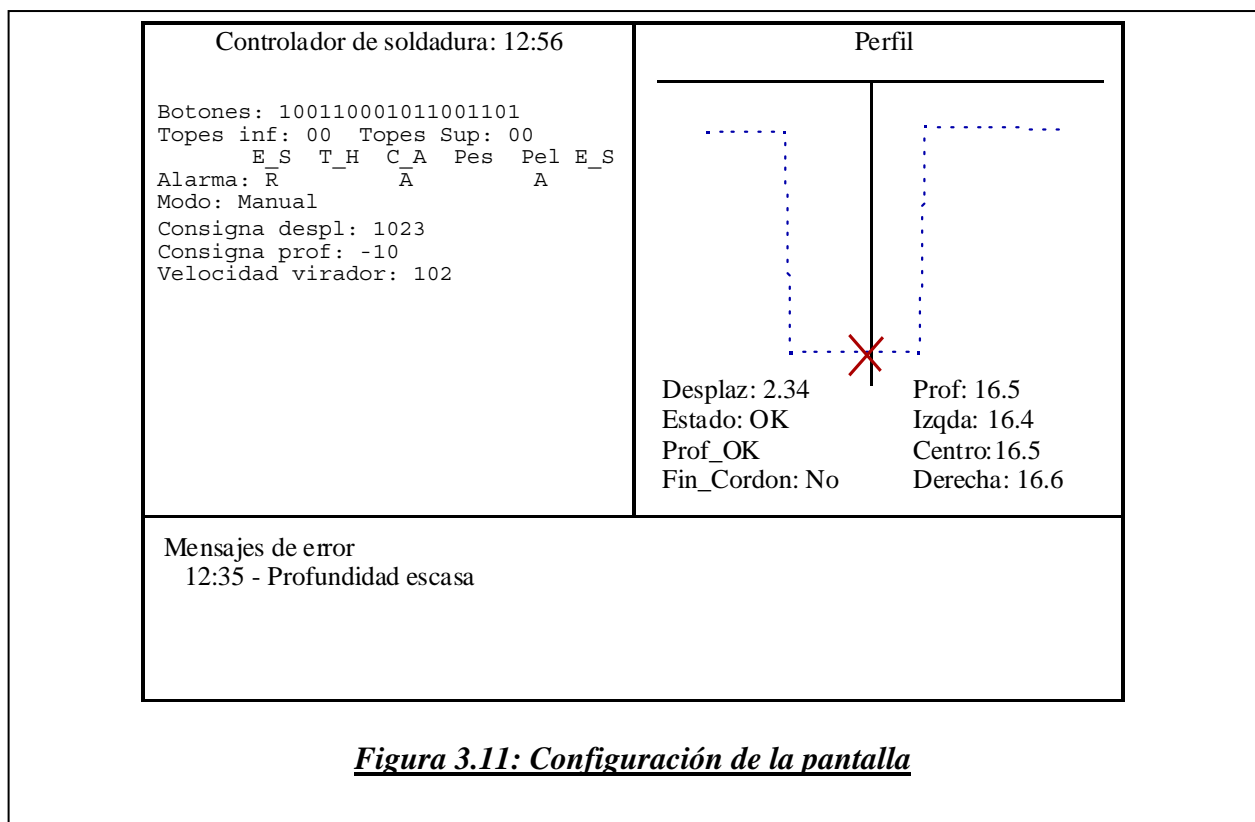


Figura 3.11: Configuración de la pantalla

3.7.11.- Drivers de Entrada/Salida

El sistema requiere drivers para realizar operaciones de entrada/salida sobre los siguientes dispositivos especiales:

- Convertidor digital/analógico
- Entradas y Salidas Digitales

Para facilitar el uso de los drivers de bajo nivel desde programas en Ada y para ofrecer una interfaz más “amigable”, se accederá a cada uno de estos drivers a través de módulos cuya interfaz y funciones se describen a continuación.

3.7.11.1.- Entradas y Salidas Digitales

3.7.11.1.1.- Descripción

Este módulo encapsula el manejo de las tarjetas de entradas digitales y salidas digitales.

3.7.11.1.2.- Operaciones que soporta

- `Pon_Canal`: Pone el canal de salida indicado por `Bit` al valor lógico indicado por `Estado`.
- `Lee_Canal`: Retorna el valor lógico leído en el canal de entrada indicado por `bit`.
- `Cierra`: Cierra el fichero de dispositivo. Debe usarse cuando se ha acabado de usar el paquete (el fichero se abre al instanciarse el paquete).

3.7.11.1.3.- Excepciones que eleva

- `Error_Drv_Digital`: Esta excepción se eleva como consecuencia de un fallo interno del driver.

3.7.11.2.- Salidas Analógicas

3.7.11.2.1.- Descripción

Este módulo encapsula el manejo de las salidas analógicas.

3.7.11.2.2.- Operaciones que soporta

- `Pon_Canal_Salida`: Pone el canal de salida indicado por `Canal` al valor indicado por `Valor` (medido en unidades del convertidor).
- `Lee_Canal_Salida`: Retorna el valor escrito en el registro correspondiente al canal de salida indicado por `Canal`.
- `Cierra`: Cierra el fichero de dispositivo. Debe usarse cuando se ha acabado de usar el paquete (el fichero se abre al instanciarse el paquete).

4.- Brazo Telemanipulado

4.1.- BTM

El brazo tele manipulado (BTM) es un brazo articulado antropomórfico que consta de 6 articulaciones independientes movidas por servomotores. Su base esta enclavada en la pieza donde se desea trabajar o a la superficie que la sostiene.

Desde el punto de vista mecánico el brazo se puede descomponer en tres partes:

Cadena básica: Consta de 3 articulaciones llamadas base, hombro y codo que se emplean para posicionar en el espacio la muñeca.

Muñeca: Posee 3 movimientos, giro, elevación y rotación, que se emplean para orientar el elemento terminal en el espacio

Herramienta: Es el elemento terminal del robot y en este caso se trata de una pinza.

4.1.1- Estructura global del controlador

La figura 4 muestra un esquema general de los objetos en que se ha dividido el software del controlador, así como las principales relaciones entre esos objetos.

A continuación se describen brevemente cada uno de los objetos del sistema.

- **Configuración:** En este módulo se describen todos los parámetros del sistema que son configurables. Aunque muchos de estos parámetros corresponden a otros módulos del sistema, tales como los mandos o el brazo, se ha preferido agruparlos todos en un mismo módulo con objeto de simplificar la labor de configurar el brazo. Casi todos los módulos del sistema leen los parámetros configurables del módulo configuración.
- **Mandos:** Este módulo engloba todos los elementos de la unidad de mandos del robot, en concreto la botonera, las luces de la botonera, la seta de emergencia y el joystick de control. El módulo proporciona operaciones para leer el estado de los mandos, así como para poder encender o apagar las diferentes luces del cuadro de mandos.
- **Brazo:** Este módulo engloba los elementos del brazo tele manipulado. Los elementos se agrupan en dos tipos: sensores, cuyos valores se pueden leer, y actuadores, sobre los que se puede actuar. Los sensores son los topes de recorrido de cada eje, los sensores de posición (resolvers) y los sensores de intensidad de los motores. Los actuadores son los que permiten especificar las consignas de los servos, y el rele o reles que controlan la alimentación del brazo.
- **Gestor de Mandos:** Este módulo gestiona las órdenes que el usuario introduce a través de la unidad de mandos. En concreto determina el modo de funcionamiento del robot, gestiona los cambios de modo de funcionamiento, y almacena órdenes que el planificador de trayectorias debe ejecutar. También inicializa el control de servos si se solicita. Por ultimo, gestiona las luces del cuadro, encendiéndolas y apagándolas en función del estado y de las alarmas existentes.
- **Planificador de Trayectorias:** Este módulo se encarga de planificar las trayectorias que el robot debe de seguir. Como entradas, recibe el estado actual y las órdenes —que provienen del gestor de mandos— y los valores del joystick, y la posición actual proviene del brazo. Como salida proporciona periódicamente puntos al controlador de servos.
- **Control de Servos:** Se encarga de ejecutar el algoritmo de control de servos. Como entradas recibe las posiciones actuales de los motores y el próximo punto a alcanzar, suministrado por el planificador de trayectorias. Genera como salidas las consignas de intensidad de los motores.
- **Reportero:** Este módulo se encarga de leer periódicamente el estado del sistema, y componer un mensaje que se envía a través de la línea serie. Este mensaje permite a un sistema externo conocer en cada momento el estado del sistema.
- **Alarmas:** Este módulo se encarga de almacenar el estado de las alarmas del sistema. Otros módulos pueden modificar este estado, o leerlo.

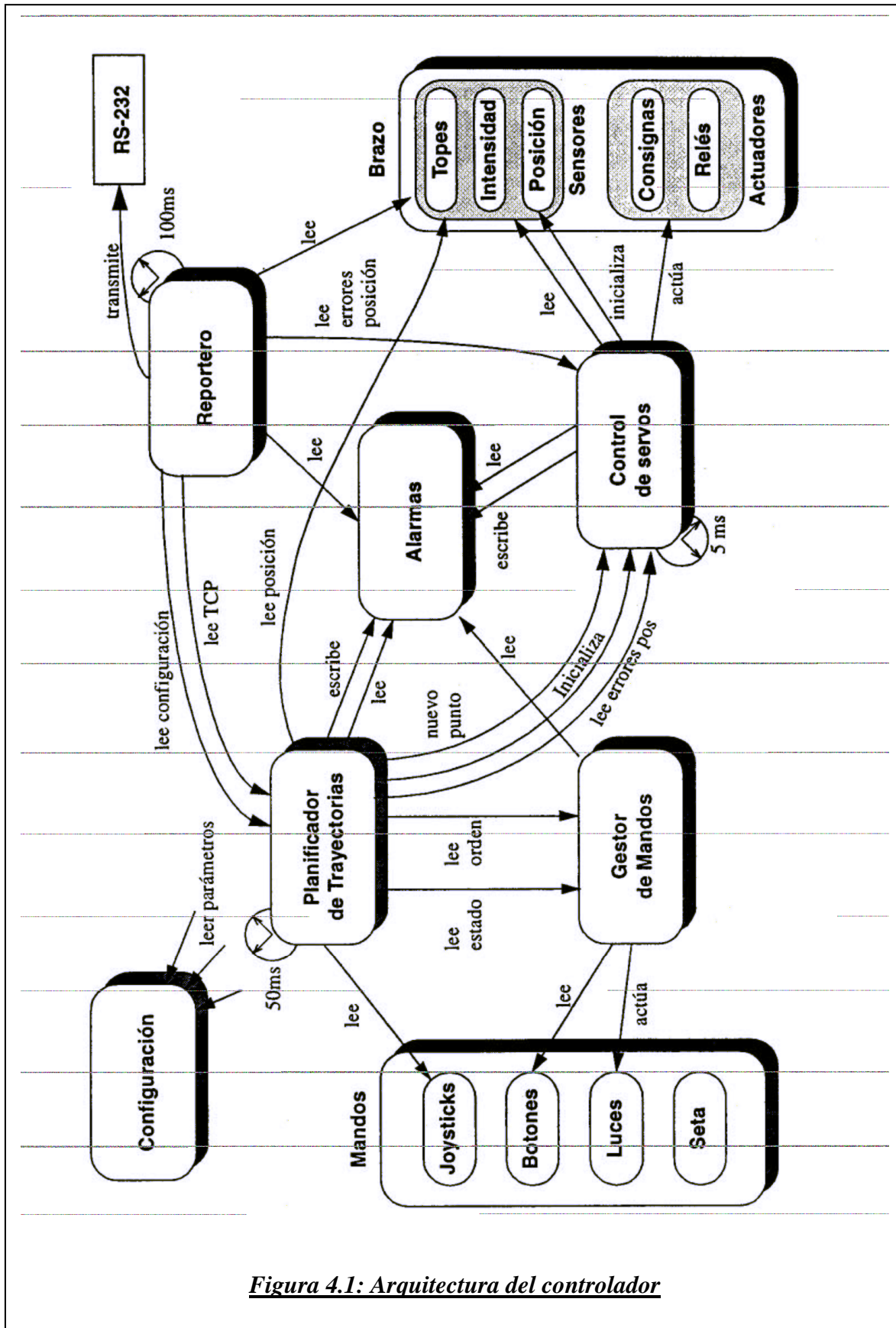
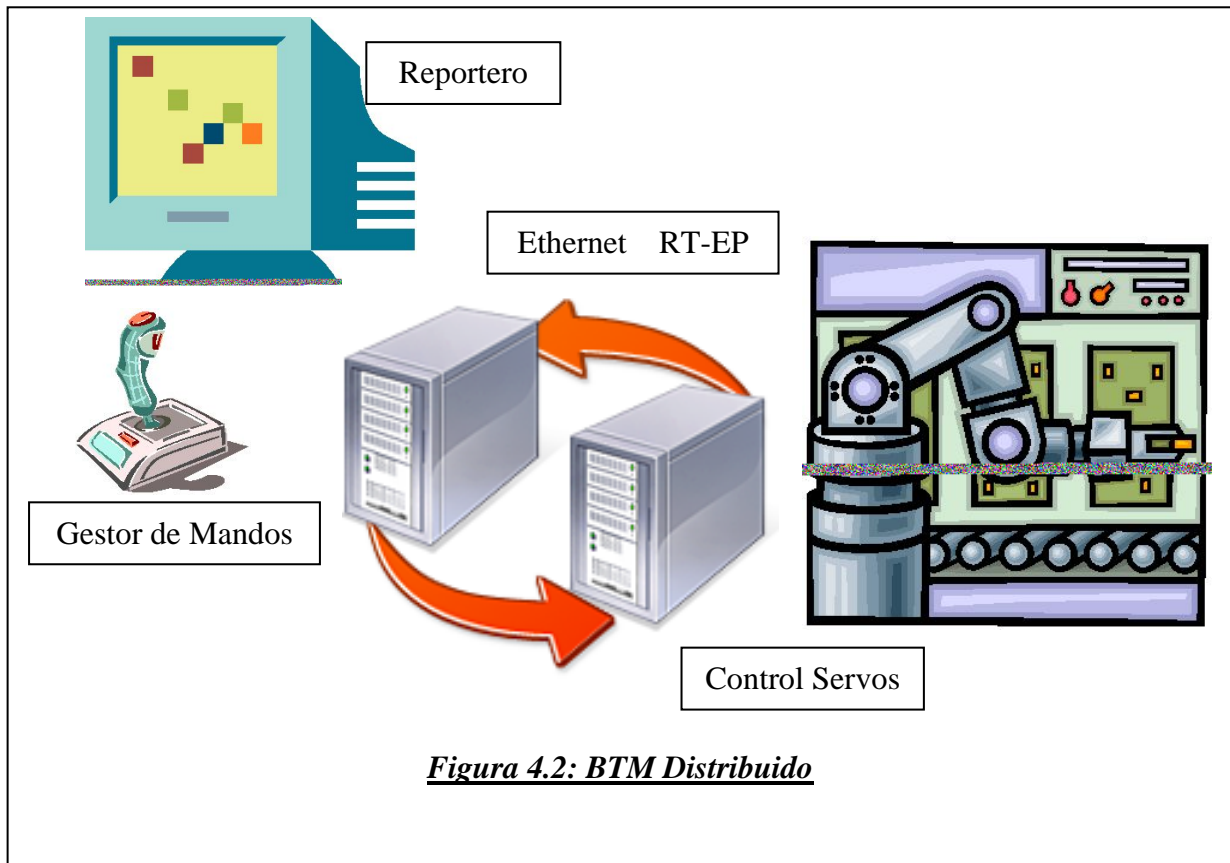


Figura 4.1: Arquitectura del controlador

Esta aplicación es distribuida, una parte se encuentra en el lugar de trabajo del operario, desde el cual éste está manejando el brazo robótica desde el Control de Mandos y viendo por pantalla lo que ocurre con la aplicación y una segunda que se encuentra en el lugar donde se sitúe el robot que es la encargada de manejar los servos del brazo robótico y realizar el control de las trayectorias, ambas particiones se comunican entre si a través de una red ethernet dedicada con un protocolo de tiempo real (RT-EP) que asegura las especificaciones temporales de comunicación de la red, y cuando el operario introduce ordenes estas se transmiten a de una particion de la aplicacion distribuida a la otra para que el robot las realice.



Conclusiones

Como se puede observar la estructura básica de la arquitectura del controlador en el caso del Láser y en el caso del BTM son muy similares, sin tener en cuenta las variaciones propias debido a los módulos específicos de cada aplicación para cumplir la funcionalidad de la aplicación.

La diferencia mas significativa entre ambas aplicaciones en lo que se refiere a la estructura básica de los módulos es que el Láser no tiene el control de los servos en un paquete específico sino que lo tiene integrado en el Planificador de la aplicación y el BTM lo tiene en un módulo específico. Esto es debido a que el control de servos en el caso del controlador Láser es muy sencillo, mientras que el BTM requiere un módulo específico (**Control de Servos**) para realizar este cometido ya que en este caso necesita una tarea dedicada para el calculo de las trayectorias y el calculo de las consignas debido a la complejidad de los cálculos.

También cabe destacar que esta estructura modular es independiente de la distribución de los módulos dentro de la aplicación, ya que en el caso de la aplicación de láser esta se encuentra empotrada en un único procesador, mientras que la del BTM es una aplicación distribuida.

Hemos visto que se puede aplicar este modelo de referencia, es decir, los distintos módulos con su funcionalidad y sus interrelaciones básicas para realizar diversos tipos de controladores industriales, como el caso del “Controlador de Posicionamiento por Perfil Láser en la Soldadura de Virolas”, en el que se ha visto de una forma mucho mas detallada la composición y funcionalidad de cada uno de los módulos, pero pudiéndose aplicar a otro tipo de sistemas industriales con funcionalidades y propósitos muy dispares como puede ser el caso del Brazo Telem manipulado (BTM).

Se ha realizado con éxito la aplicación “Controlador de Posicionamiento por Perfil Láser en la Soldadura de Virolas” basándose en este modelo de referencia, y habiendo sido satisfactoria la utilización del sistema operativo MaRTE OS, que nos a permitido garantizar que el sistema es de tiempo real, es decir, que es planificable, y tiene los tiempos de respuesta perfectamente acotados y el lenguaje de programación Ada.

Bibliografía y Referencias.

- Mario Aldea, “Planificación de Tareas en Sistemas Operativos de Tiempo Real Estricto para Aplicaciones Empotradas”, Tesis Doctoral, Dpto. Electrónica y Computadores, Grupo de Tiempo Real, Universidad de Cantabria, Mar 2003.
- Michael González Harbour. “Apuntes de Ada”. UC, 2001.
- José Luis Mantecón, “Librería Grafica de bajo nivel para el sistema operativo de tiempo real MaRTE OS”, Trabajo fin de carrera UC, 2003.
- Daniel Sangorrín, “Gestión de Dispositivos de Entrada/Salida Analógica, Digital y por el bus serie I2C”, Trabajo fin de carrera UC, 2006.
- J. Barnes. “Programming in Ada 95”. Firts Edition. Addison-Wesley, 1995.
- M Feldman, E. Koffman “Ada 95: Problem Solving and Problem Desing”.Addison-Wesley.
- Burns A. And Wwellings A.”Concurrencia in Ada”. Cambridge 1995.
- Manual Referencia Ada95. <http://www.adahome.com/rm95/>
- MaRTE OS. <http://martec.unican.es/>
- PCLD-782B. http://www.advantech.gr/products/Model_Detail-7957.asp.htm
- PCLD-785B. http://www.advantech.nl/products/24-ch-Relay-Output-Board/mod_1-2MLIFX.aspx
- PCM 3724. http://www.advantech.com/products/48-ch-Digital-I-O-PC-104-Module/mod_1-2MLIQC.aspx
- PCM 3712. http://www.advantech.com/products/2-ch-12-bit-Analog-Output-PC-104-Module/mod_1-2MLIOL.aspx
- Nova 7896. <http://www.jenlogix.co.nz/products/nova-7896.htm>
- PC 104 Bus. <http://pc104.org>
- Arranque por Compact Flash. <http://leaf.sourceforge.net/doc/guide/bugrub.html>
- Cargador NETBOOT. <http://netboot.sourceforge.net>

