Universidad de Valladolid

Escuela Técnica Superior de Ingenieros Industriales

PROYECTO FIN DE CARRERA

DESARROLLO DE UNA APLICACIÓN PARA LA GESTIÓN INTEGRAL Y EL CONTROL DE UNA CÉLULA DE FABRICACIÓN FLEXIBLE

Pedro Sanz Angulo

Septiembre, 2001

PROYECTO FIN DE CARRERA

DESARROLLO DE UNA APLICACIÓN PARA LA GESTIÓN INTEGRAL Y EL CONTROL DE UNA CÉLULA DE FABRICACIÓN FLEXIBLE

Presentado en la

Escuela Técnica Superior de Ingenieros Industriales

Por: D. Pedro Sanz Angulo

Dirigido por: D. Juan José de Benito Martín

Para la obtención del título de INGENIERO INDUSTRIAL

VALLADOLID, Septiembre de 2001

D. JUAN JOSÉ DE BENITO MARTÍN, Ingeniero Industrial, profesor del Departamento de Organización y Gestión de Empresas de la Escuela Técnica Superior de Ingenieros Industriales de la Universidad de Valladolid.

HACE CONSTAR que el proyecto Fin de Carrera titulado **"DESARROLLO DE UNA APLICACIÓN PARA LA GESTIÓN INTEGRAL Y EL CONTROL DE UNA CÉLULA DE FABRICACIÓN FLEXIBLE"**, que presenta D. Pedro Sanz Angulo, para optar al título de Ingeniero Industrial, ha sido realizado en el Departamento de Organización y Gestión de Empresas de la E.T.S.I.I. de Valladolid, bajo mi dirección, y constituye una valiosa aportación en el campo de la gestión de los sistemas productivos.

Para que así conste a efectos de su presentación ante el Tribunal correspondiente, firmo la presente en Valladolid, Septiembre de 2001.

Este Proyecto Fin de Carrera quiero dedicárselo a mis padres, a mis hermanos, a mi tía Lola y a todas las personas que me han apoyado a lo largo de mi carrera, y con especial cariño a mi novia Elena.

Finalmente, quiero agradecer la inestimable colaboración de mi director Juan José de Benito Martín, y hacer extensivo este agradecimiento al resto de profesores del Departamento.

ÍNDICE

INTRODUCCIÓN

El proyecto que se presenta a continuación pretende mejorar los proyectos anteriores encargados del control de la Célula de Fabricación Flexible (FMC) del Laboratorio de Organización Industrial y Producción (LOIP) de la Escuela Técnica Superior de Ingenieros Industriales de Valladolid.

El primero de dichos proyectos fue desarrollado en lenguaje C bajo el sistema operativo MS-DOS, lo que repercutió negativamente en su interfaz gráfica y en su accesibilidad. Además, no se implementaron todas las funciones posibles, por lo que el control remoto de cada una de las máquinas no era el adecuado. En cuanto al proceso de fabricación integral, este proyecto era poco flexible dado que tan sólo permitía una única configuración de las máquinas del LOIP. Por último, las comunicaciones de este proyecto se realizaban a través de una problemática tarjeta multiplexora/demultiplexora denominada NIU (Network Interface Unit).

Con la intención de solucionar algunos de estos problemas se planteó la realización de un segundo proyecto. Éste se desarrolló bajo el entorno Windows 95 empleando la herramienta de desarrollo Borland C++ Builder 1.0, y durante su creación se instaló la tarjeta de comunicaciones *National Instrument PCI-232/8*, que proporcionaba un total de 8 nuevos puertos serie. Sin embargo, estas fueron básicamente las únicas mejoras realizadas, y algunos de los problemas planteados en el párrafo anterior, en lugar de mejorar, empeoraron.

Dados estos antecedentes, se hacía imprescindible el desarrollo una nueva aplicación informática que, además de ser capaz de realizar la gestión integral y el control de todas las máquinas que constituyen la FMC del LOIP, solucionase la problemática generada en los proyectos anteriores.

Dicha aplicación se denominará FMC, que es el acrónimo de *Flexible Manufacturing Cell* (Célula de Fabricación Flexible), y permitirá mejorar aspectos significativos respecto a las versiones anteriores.

- Se acentuará la flexibilidad de la célula permitiendo que nuevos tipos de piezas se incorporen al sistema productivo.

- Permitiremos el uso de los puertos propios del ordenador, COM1 y COM2, junto a los de la nueva tarjeta, lo que aumentará la flexibilidad en la comunicación.

- Mejoraremos el entorno de trabajo del usuario principal gracias, sobre todo, al entorno de ventanas y botones de Windows y al empleo de la herramienta de desarrollo visual y rápido Borland C++ Builder 4.0.

- Incluiremos todas aquellas funciones que nos permitan un mejor control remoto de las máquinas.

- Además, intentaremos que todas las operaciones necesarias se puedan realizar sin tener que soltar la mano del ratón, lo que nos permitirá ganar en comodidad y rapidez.

- También perseguiremos optimizar el almacenamiento de la información utilizando bases de datos.

- Y, por último, intentaremos crear un sistema productivo independiente de la ubicación física de cada una de las máquinas de la FMC.

Tras definir el objetivo de este proyecto, se exponen a continuación los aspectos más importantes que se van a tratar en esta memoria. En concreto, el trabajo realizado durante este proyecto se ha resumido en siete capítulos. A lo largo de estos siete capítulos describiremos el funcionamiento de la aplicación FMC, así como los conceptos clave que nos han permitido conseguir la integración y el control de la célula.

En el capítulo primero se presentan algunas ideas que nos permitirán situarnos en el entorno de una FMC. Se analiza la filosofía CIM (*Computer Integrated Manufacturing*) y los Sistemas de Fabricación Flexible (FMS: *Flexible Manufacturing Systems*), como exponentes más representativos de los nuevos sistemas de fabricación, que es el ámbito en el que nos vamos a mover.

Las comunicaciones son cruciales en el funcionamiento de cualquier sistema integrado de producción, y lo han sido, en particular, para el desarrollo del presente proyecto. Por este motivo, en el capítulo segundo nos sumergiremos en el mundo de las comunicaciones bajo entorno Windows y, en concreto, analizaremos las funciones de comunicación de la API de Windows (Win32 Communications API).

El tercer capítulo tiene por objetivo mostrar los aspectos más relevantes de las diferentes máquinas que integran nuestra célula. Todas estas peculiaridades nos servirán para comprender su funcionamiento y la estructura de la aplicación desarrollada.

Los dos capítulos siguientes, cuarto y quinto, constituyen el *manual de utilización* de la aplicación FMC. En el primero de ellos se analizan los módulos encargados del control remoto de cada una de los elementos de la célula, cuyas características principales ya descubrimos en el capítulo anterior. Todos los elementos del sistema (torno, fresadora, robots, etc.), poseen dentro de FMC su pequeño módulo que nos permite realizar, a través del Ordenador Central (*Host*), la mayor parte de sus funciones.

En el capítulo quinto se presenta el resto del software FMC. Plantearemos un supuesto práctico que nos permitirá describir todos los pasos a seguir necesarios en la fabricación de piezas en la FMC: arranque y puesta en funcionamiento, elaboración del diseño de una célula, diseño de una tarea, secuenciación, ejecución,... También descubriremos las diferentes herramientas de que dispone el programa FMC destinadas a facilitar el trabajo del usuario que utilice el ordenador central.

El capítulo sexto proporciona una visión global del código desarrollado en este proyecto. Razonaremos la elección de la herramienta de desarrollo, analizaremos la estructura de directorios de nuestro programa y comentaremos los aspectos más relevantes del código (*manual de programación*).

El capítulo séptimo nos va a servir para ofrecer un estudio detallado del coste económico del software, realizando un desglose de las diferentes partidas de coste en las que se ha incurrido.

Finalizaremos con las conclusiones más relevantes de este proyecto indicando, además, posibles orientaciones destinadas a mejorar nuestro sistema que sirvan de guía a futuros proyectos.

CAPÍTULO PRIMERO:

ENTORNO GLOBAL DE UNA CÉLULA DE FABRICACIÓN FLEXIBLE.

1.1. CIM: CONSIDERACIONES BÁSICAS.

El sector de producción industrial está sufriendo una transformación rápida y profunda. Los mercados industriales internacionales se encuentran en continua evolución, lo que se manifiesta a través de unos ciclos de vida y suministro del producto especialmente cortos, un aumento de su diversidad y de los requisitos relativos a la calidad, etc.

Con el fin de poder subsistir en esta situación de competencia internacional cada vez más rigurosa, las empresas se ven obligadas a adoptar medidas encaminadas al incremento de su productividad y a imprimir flexibilidad a sus ciclos de producción, a fin de mejorar su rentabilidad y, por tanto, sus posibilidades en el mercado.

Es por ello que durante las últimas décadas se viene hablando de la filosofía CIM (Computer Integrated Manufacturing), que reúne todos aquellos aspectos que contribuyen a mejorar la rentabilidad. Este concepto se refiere también al tratamiento continuo de la información en una moderna empresa de producción. Las medidas que se resumen bajo el concepto CIM apuntan hacia la llamada *Fábrica del Futuro* o, mejor dicho, *Fábrica con Futuro* [B3].

1.1.1. Nacimiento y evolución del concepto CIM.

Antes era el fabricante quien definía el mercado a través de su programa de producción mientras que, actualmente, el mercado de los productores se ha convertido en el mercado de los compradores. Es el cliente quién determina el producto y sus características, y el fabricante debe aceptar sus deseos. Esta situación está dirigida por la tecnología, que es la causante de la reducción de los ciclos de vida y de la globalización, con el consiguiente aumento de la competencia internacional. Mantener la competitividad se ha convertido, por tanto, en una cuestión primordial para todas las empresas.

Todas estas ideas definen la situación en la que se encuentra el mercado, tal y como se aprecia en el esquema de la figura 1.1.

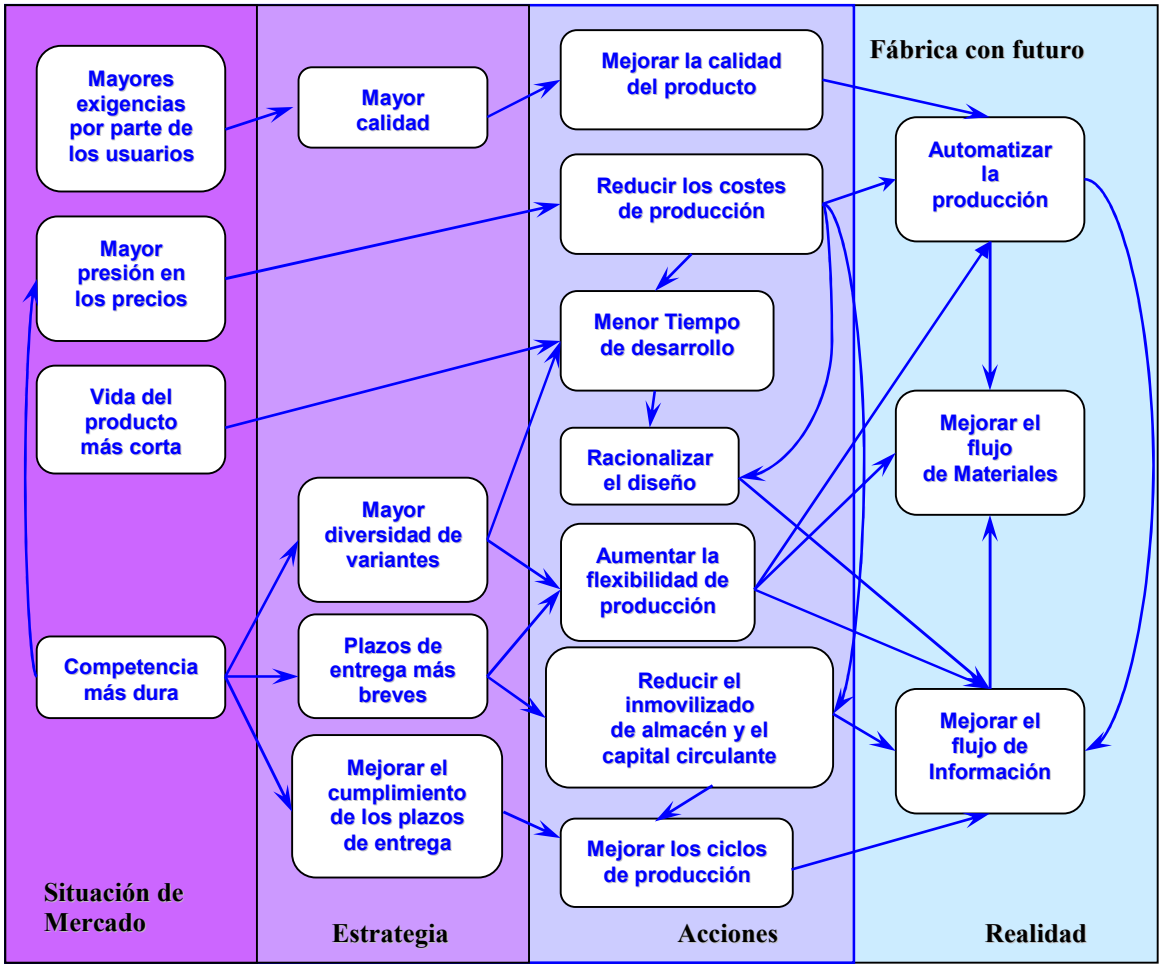


Figura 1.1. *Nacimiento del concepto CIM* [B3].

Las empresas pueden acercarse al objetivo de competitividad a través de las siguientes medidas estratégicas:

- mejora de la calidad de los productos,
- ampliación de la gama,
- reducción de los plazos de suministro,
- mejora en el cumplimiento de los plazos.

Actualmente se han introducido sistemas de fabricación y montaje para los fines más diversos. La utilización de sistemas de automatización, como por ejemplo ordenadores de gran capacidad para el control de la producción, sistemas de fabricación automatizados, máquinas-herramienta con control numérico, robots industriales, etc. permiten incrementar la productividad, incluso cuando se fabriquen lotes pequeños.

En el pasado, la mayor parte de las medidas encaminadas al incremento de la productividad se centraban casi exclusivamente en modernizar las técnicas de producción, mientras que la automatización se aplicaba en ámbitos parciales. Sin embargo, mediante estas soluciones aisladas los objetivos anteriores sólo podían alcanzarse hasta un determinado punto y, por tanto, tan sólo se lograban éxitos parciales.

Una automatización efectiva presupone la acción coordinada de tres funciones: mecanización, flujo de materiales y flujo de la información. En las modernas instalaciones, la *información* se convierte en un factor de producción decisivo. Para mejorar la flexibilidad de una empresa es necesario mejorar la calidad de la información disponible que, además, deberá ser procesada en cantidades mayores. Va a ser necesario el tratamiento integrado de los datos técnicos, lo que presupone la existencia de un flujo continuo de información para que el tratamiento electrónico de datos se convierta en un sistema de información global.

Por este motivo, después de haber desarrollado sistemas de automatización aislados, lo que se pretende es que los datos generados en cada uno de los sistemas sean también accesibles a otros ámbitos y sistemas. Por consiguiente, la técnica de la información orientada hacia el futuro no puede detenerse en los limites de las distintas islas o secciones automatizadas, sino que ha de proyectarse a un nivel superior. Al incluir todos los ámbitos de la empresa que participan en la producción, incluidos proveedores y clientes, la fábrica del futuro puede llegar a ser una realidad. Los objetivos fijados solo pueden alcanzarse si la fábrica se explota en su conjunto, y no en secciones parciales.

De la misma manera que el flujo de materiales y el flujo de energía se tratan desde el punto de vista logístico, actualmente se reconoce que el flujo de información debe tratarse también como un problema logístico. En lugar del factor de producción *Información* puede hablarse igualmente de una *Logística de la Información*, para la que son válidas las siguientes premisas:

- la información debe ser correcta,
- en la cantidad y calidad adecuadas a las necesidades,
- en el momento preciso,
- y en el lugar adecuado.

La solución logística de la información exige que se desenmarañen las estructuras tradicionales y se creen ámbitos funcionales con interfaces claras que garanticen la transparencia de las funciones de la empresa y faciliten el tratamiento informático. En este ámbito, en la resolución de este problema logístico de información, es donde surge el concepto CIM.

- El concepto CIM es un planteamiento dirigido hacia el futuro, a fin de poder crear y ampliar de forma sistemática los actuales sistemas de automatización de la producción.
- CIM define la futura estructura de automatización de la producción a partir de datos de producción comunes y homogéneos.

- CIM exige que se utilicen sistemas de automatización capaces de comunicarse entre sí, tales como controles de memoria programables, controles numéricos, ordenadores con sistemas de gestión de datos, redes de comunicación y sistemas de software, para poder asegurar un flujo continuo de información. CIM es, por lo tanto, un medio que permitirá convertir en una realidad los objetivos de la empresa. CIM permite, por lo tanto, asegurar el futuro de la empresa.

Sin embargo, el éxito y la utilidad de CIM dependen en gran parte del nivel de armonización que se consiga entre las posibilidades del tratamiento de información asistido por ordenador y las correspondientes estructuras organizativas. Las relaciones entre la organización, las técnicas de automatización y el tratamiento de la información deben considerarse en su conjunto y a nivel superior, sin perder de vista por ello las posibilidades y capacidades, ni los deseos de los empleados afectados.

Por último, para justificar las inversiones necesarias es preciso realizar cálculos de rentabilidad que no han de verse limitados por la exigencia de conseguir resultados a corto plazo. Al decidirnos por CIM, no debemos preguntar: "¿Cuánto vamos a ahorrar a corto plazo?", sino: "¿Hacia dónde se encaminará nuestra empresa si renunciamos a introducir el CIM?", o bien: "El desarrollo de la empresa sin CIM ¿es compatible con los objetivos que determinan su competitividad?". Estas preguntas demuestran que CIM es, ante todo, un concepto estratégico que asegura a largo plazo la competitividad de la empresa, en armonía con sus objetivos.

En definitiva, *"CIM es una estrategia que busca la integración de los procesos, la información, las estructuras y las tecnologías. Todos estos elementos están fuertemente interrelacionados entre sí y pueden abordarse mejor si los consideramos como partes de un único sistema global. La clave del CIM es, por tanto, la integración"* [B3].

1.1.2. Componentes del CIM.

La implantación de un sistema CIM precisa de un cambio planificado que tiene en cuenta los tres componentes fundamentales de la empresa: el sistema de fabricación, el sistema de información y el sistema humano. Estos tres sistemas interaccionan y se superponen entre sí.

La mayor parte de los problemas de la implantación de un sistema CIM se plantean en el área de interrelación que existe entre estos tres componentes de la empresa y en el área de interrelación entre cada dos de ellos. Esta interacción queda reflejada en la figura 1.2.

Figura 1.2. *Componentes básicos del CIM* [B2].

La intersección de estos tres componentes define el área donde el cambio es más pronunciado y, en consecuencia, donde aparece el mayor número de problemas y dificultades relacionadas con la implantación efectiva del CIM. Esta circunstancia se debe, principalmente, a que la mayoría de las empresas son capaces de gestionar cada sistema de manera independiente, pero encuentran serias dificultades cuando pretenden una gestión integrada de dichos sistemas, que es el objetivo fundamental del CIM.

- ***Sistema Humano***. Puede considerarse que está dividido en dos estructuras: una de dirección y otra organizativa. La estructura de dirección define la estrategia de negocio de la empresa, los planes de acción, los factores críticos de éxito, las medidas de rendimiento, la estructura organizativa y las diferentes políticas a adoptar. La estructura organizativa establece las relaciones entre todas las funciones de la empresa y está condicionada por la naturaleza del negocio y por sus clientes. La tendencia actual es disminuir los niveles jerárquicos en las organizaciones, unida a un fuerte incremento de las relaciones entre actividades.

- ***Sistema de Información***. Puede ser divido a su vez en dos estructuras: estructura de información y estructura de sistemas informáticos. La estructura de información representa las relaciones entre conocimiento, información y datos, mientras que la estructura de sistemas informáticos permite automatizar el almacenaje y procesado de la información.

- ***Sistema de Fabricación.*** Refleja cómo se realizan las actividades de la empresa. Su estructura depende de los procesos productivos y es la que representa el mayor desembolso financiero cuando queremos alcanzar el CIM.

1.1.3. Jerarquía CIM.

Dentro de una factoría dirigida mediante un sistema CIM, las actividades pueden ser ordenadas de acuerdo con un sistema jerárquico sobre el que discurre un flujo de intercambio de información. Existen varios niveles de control. La base de este sistema es la descentralización del control. El ordenador o controlador de cada nivel tiene asignadas determinadas tareas, pero para llevarlas acabo necesitará datos del controlador del nivel inmediatamente superior al que a su vez devolverá una cantidad específica de información. Existe, por tanto, un continuo flujo de información a través de los distintos niveles.

Podríamos describir los distintos niveles de control de un CIM basándonos en el *Modelo de la Pirámide* que se muestra en la figura 1.3.

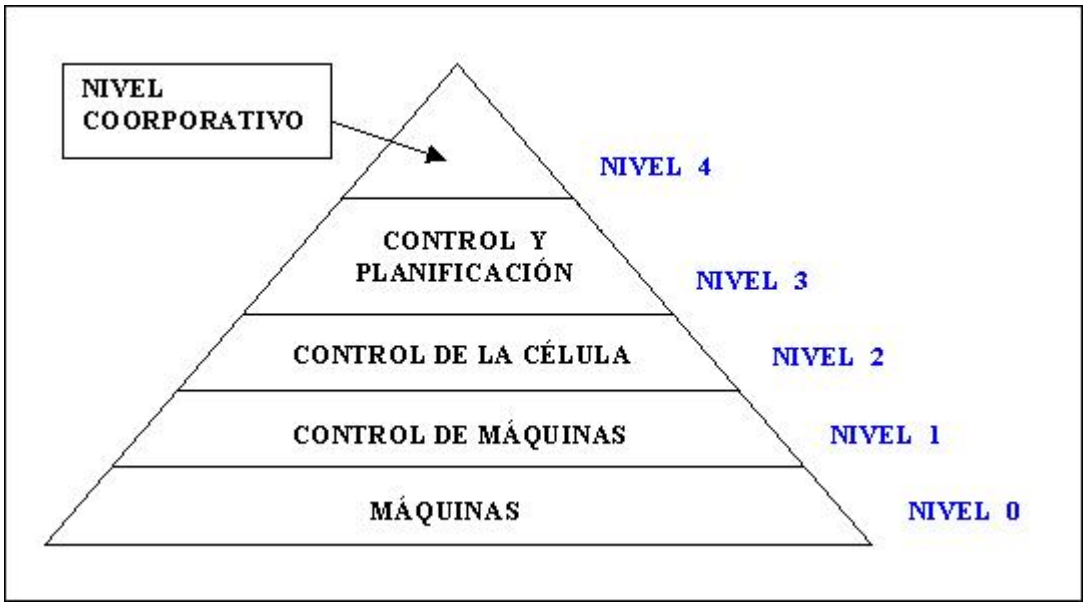


Figura 1.3. *Jerarquía CIM. Modelo de la Pirámide* [B2].

Nivel 0 ☞ Nivel de máquina.

Está compuesto por las distintas máquinas con la que trabaja la empresa (fresadoras, tornos, robots, hornos,...). En este nivel están situados los elementos que generan señales analógicas y/o digitales tales como sensores, terminales, cámaras de vídeo, etc. Es la información de más bajo nivel dentro de la pirámide.

Nivel 1 ☞ Nivel de control de máquinas.

Comprende los elementos de mando y control de la maquinaria del nivel 0. Incluye toda la lógica binaria, los controles numéricos, los PLC y los microprocesadores. Este nivel proporciona información al nivel 2 y se actualiza por la información que recibe del nivel 0. Se le suele llamar *Nivel de Automatización.*

Nos podemos encontrar con muchos sistemas de control, entre los que destacaríamos los siguientes:

- *Autómatas programables (PLC).* Es uno de los elementos básicos de cualquier fábrica automatizada. Estos elementos de control poseen un microprocesador, una memoria principal, un subsistema de entradas/salidas y un interface que posibilita la comunicación con el exterior de tal forma que podamos comunicarlo con un ordenador central. Sus ventajas son muchas, entre ellas podemos citar su bajo coste, la facilidad de utilización, su gran resistencia a los ambientes industriales, su gran fiabilidad y sobre todo que son capaces de trabajar en tiempo real. Uno de los principales inconvenientes es la programación de los mismos puesto que no existen estándares y cada fabricante usa su propio lenguaje. Sin embargo, existen similitudes entre todos ellos puesto que la mayoría se basan en diagramas de contactos o lógica cableada.

- *Controladores de robots*. Normalmente los robots sencillos pueden ser controlados directamente por un PLC, pero los robots más complejos necesitan de un ordenador y una unidad de entradas y salidas como controlador. Este controlador ha de ser capaz de ejecutar algoritmos complicados en función de la situación del robot y de su entorno para poder ejecutar uno u otro bloque de instrucciones según las señales que le lleguen.

- *Controladores para los sistemas automáticos de transporte de materiales*. Para estos sistemas suele utilizarse un autómata programable, aunque también podemos encontrarnos células de fabricación en donde estén controlados por un ordenador con su correspondiente sistema de entradas y salidas.

- *Control numérico (CNC).* Los controladores para máquinas herramienta, tales como fresadoras, tornos, centros de mecanizado, etc. se denominan controles numéricos. Este tipo de control ha sufrido muchas modificaciones desde sus comienzos en los años 60, consiguiendo un gran aumento de fiabilidad, un aumento en la memoria del sistema, y una programación en lenguajes de alto nivel

como el APT. La última generación de controles numéricos son los denominados DNC (Direct Numerical Control), que disponen de un puerto de comunicaciones que permite la conexión directa a un ordenador desde el cual recibe los bloques de instrucciones que forman el programa de mecanizado. La tendencia actual del control numérico es hacia los DNC muy simples que puedan ser usados en aquellas plantas que dispongan de sistemas de CAM avanzados y paquetes de distribución de programas desde un ordenador central, o bien hacia un CNC muy complejo, para aquellas plantas que no dispongan de un sistema CAM.

Nivel 2 ☞ Nivel de supervisión y control de la célula.

Tanto una célula como un sistema de fabricación flexible están formados por un conjunto de maquinaria automatizada, sensores y elementos de alimentación y transporte conectados mediante una red industrial de comunicaciones a un ordenador central que coordina todo el sistema. Aunque un FMS y un FMC se diferencian principalmente en el grado de flexibilidad y complejidad, siendo ambas mucho mayores en el FMS que en el FMC, un FMS puede estar formado por un conjunto de FMC, de donde se deduce que el elemento básico es la célula de fabricación flexible.

En este nivel se realizan tareas como: supervisión, recogida de datos, programación, control de calidad, control de las piezas en curso de fabricación, gestión de alarmas, sincronización de células de fabricación, coordinación del transporte de material, aprovisionamiento de líneas, seguimiento de lotes, mantenimiento correctivo, etc.

La célula de fabricación flexible actúa como filtro de la inmensa cantidad de información que intercambian los niveles 1 y 3. El nivel 2 (*FMS/FMC*) recibe y envía información al nivel 1, pero también recibe los programas de producción, calidad, mantenimiento, etc. del nivel 3 y, a su vez, envía información a dicho nivel.

Nivel 3 ☞ Nivel de planificación y control de planta.

Realiza, entre otras, las siguientes tareas: programación de la producción, gestión de materiales, compra de recursos de fabricación, gestión de la calidad, gestión del mantenimiento, análisis de costes de fabricación, control de inventarios, distribución, etc.

El nivel 3 envía programas al nivel 2 y recibe de este las incidencias que ocurren en la planta. Del nivel 4 recibe información sobre pedidos, previsiones de venta, información de ingeniería tanto del producto como del proceso, etc. A su vez, envía información al nivel 4 sobre cumplimiento de programas, costes de fabricación y de operación, etc.

Nivel 4 ☞ Nivel corporativo.

Es el nivel más alto de la empresa. Realiza las siguientes tareas:

- gestión comercial y de marketing,
- planificación estratégica,
- gestión financiera y administrativa,
- gestión de recursos humanos,
- ingeniería del producto,
- ingeniería del proceso, gestión de la tecnología y de los sistemas de información,
- investigación y desarrollo,
- planificación estratégica.

La gran diferencia entre unos niveles y otros viene dada por la vida media de la información, que es tanto mayor cuanto más alto sea el nivel en el que resida. Así, en el nivel 0 es del orden de segundos, en el nivel 1 de minutos u horas, en el nivel 2 de días, en el nivel 3 de semanas y en el nivel 4 de meses o años.

1.2. FMS: ASPECTOS FUNDAMENTALES.

La década de los 80 ha marcado un cambio de orientación en los sistemas de producción industrial. A partir de este momento, el mercado tiende a disminuir su tasa de crecimiento y a exigir una gama más amplia de productos. Este comportamiento del mercado es un reflejo de las necesidades específicas de unos clientes que presentan una menor fidelidad y demandan mayor calidad y fiabilidad del producto.

Por otro lado, la apertura gradual de los mercados mundiales hace que la competencia entre las empresas se sitúe no ya a un nivel nacional sino mundial. El mayor número de fabricantes exige a las empresas una continua adaptación a los gustos de los clientes para poder vender sus productos. Este aumento de la competencia se ha extendido no sólo al producto en sí, sino también a los componentes que lo forman; no basta con que los productos y sus componentes parezcan diferentes sino que deben ser realmente diferentes.

Estas condiciones del mercado exigen unas políticas de producción orientadas según los criterios siguientes:

- Mayor variedad de productos.
- Productos personalizados para cada cliente.
- Menor ciclo de vida del producto.
- Mayor calidad y fiabilidad de los productos.
- Reducción del tiempo de lanzamiento de nuevos productos.
- Frecuentes cambios en el diseño de productos.
- Tamaños de lote variables.
- Precios más competitivos.
- Reacción más rápida ante los cambios del mercado.
- Calidad total, *cero defectos.*

La respuesta ante estos desafíos del mercado son los *Sistemas de Fabricación Flexible,* que vienen a paliar las deficiencias que presentaban tanto los sistemas de producción en serie como los de producción por lotes.

Los *sistemas de fabricación en serie formados por líneas transfer*, presentan una alta productividad en términos de bajo coste por unidad, pero resultan ser muy poco flexibles. Para amortizar el coste de la inversión el producto debe tener un ciclo de vida muy largo.

Los *sistemas de fabricación por lote compuestos por conjuntos de máquinas convencionales* resultan ser, por el contrario, muy flexibles pero con baja productividad.

Ambos extremos resultan, por tanto, cada vez menos aplicables. El primero porque no es capaz de absorber cambios de diseño sin un elevado coste y acepta pocas variantes de una misma pieza. En cuanto al segundo, ningún cliente estará dispuesto a pagar los altos precios que conlleva su baja productividad. Ambos sistemas se presentan en inferioridad de condiciones frente a un competidor que haya adoptado un FMS. Estos sistemas aseguran una alta flexibilidad con el suficiente nivel de productividad, resultando ser una excelente solución de compromiso entre las necesidades de productividad y flexibilidad.

Un FMS proporciona una productividad elevada y, a la vez, permite una fácil adaptación a los cambios en el diseño del producto. Un ordenador principal integra los sistemas de máquinas automatizadas en un FMS para alcanzar estos objetivos. A su vez este FMS puede ser integrado en el ambiente global de CIM.

Las características fundamentales de un FMS son, por tanto:

- *Flexibilidad.* Permite fabricar muchos tipos de piezas diferentes y admite fácilmente cambios en los diseños de estas piezas.
- *Productividad*. El aumento de la productividad se debe principalmente a dos motivos:

1. Se reducen:
   - Tiempos de preparación de las máquinas.
   - Tiempos de espera.
   - Tiempos de comprobación.
   - Necesidades de almacenaje.
2. Y a la automatización de procesos como:
   - Mecanizado.
   - Cambio de piezas.
   - Cambio de herramientas.
   - Transporte de las piezas.
   - Identificación de las piezas.
   - Verificación de las piezas.

Las instalaciones de un FMS requieren de una elevada inversión, de una enorme infraestructura de soporte y de una gran experiencia en los campos de automatización y control. Pero estos inconvenientes pueden soslayarse gracias al concepto de modularidad. Podemos partir de la constitución de pequeñas FMC que más tarde podrán conectarse entre sí e integrarse en un FMS.

La diferencia entre FMS y FMC no viene dada tanto por el tamaño o número de máquinas que las componen, sino por el nivel de integración:

- La *Célula de Fabricación Flexible* está formada por unas cuantas máquinas (a veces sólo una) dotadas de control numérico, con dispositivos que permiten el cambio automático de herramientas y piezas, con una serie de almacenes que garantizan la autonomía de la célula durante, al menos, unas cuantas horas (se considera suficiente un turno de trabajo), y la incorporación de un ordenador central que controla y coordina todas las operaciones que tienen lugar en cada elemento de la FMC.

- Por contra, un *Sistema de Fabricación Flexible* puede estar constituido por varias células que se relacionan entre sí por un sistema de transporte e identificación de las piezas y están coordinadas por un ordenador central. El software de control es más sofisticado que el de una FMC y permite fabricar de forma desatendida y sin interrupción un elevado número de piezas.

1.2.1. Configuraciones de los sistemas flexibles.

La variedad de los sistemas de producción flexible es enorme debido, principalmente, a la gran diversidad de máquinas existentes y a la gran cantidad de piezas que se pueden fabricar.

El desarrollo de estos sistemas no hubiera podido existir sin la aplicación de la microelectrónica y la informática a los medios convencionales de fabricación. Con estos medios modernos se puede dotar de *flexibilidad* a las máquinas, permitiendo su adaptación a los cambios demandados por el entorno.

El *taller flexible*, o sistema de fabricación flexible, tiene por objeto dotar a las empresas de medios de producción automáticos que permitan una productividad elevada, como si se tratase de una gran serie, pero que a la vez sea fácilmente adaptable a los cambios en las características del producto y en los programas de fabricación.

Para poder conseguir el objetivo de un FMS se necesitan unos conocimientos y una experiencia elevada en los campos de control numérico, automatización de sistemas e informática, que generalmente sólo está al alcance de las grandes empresas (debido a la gran inversión que se debe hacer). Sin embargo, si bien dicha inversión es grande, las pequeñas empresas pueden optar por la introducción de una única FMC y gradualmente ir incorporando nuevas células hasta ampliar dicha automatización a todo el taller.

Un taller de fabricación flexible debe disponer de las funciones de recepción, inspección, almacenaje, transporte, mecanización, verificación, montaje, inspección y distribución, todas ellas coordinadas por un sistema informático. Cada célula dispondrá de un sistema de control, pero existirá un ordenador central que controle todos los procesos que se ejecuten en el taller.

1.2.2. Características generales de un FMS.

Un FMS debe responder rápidamente a las exigencias de los clientes manteniendo unos costes mínimos. Entre las principales características que podemos encontrar están:

i. **Flexibilidad.** Debe existir la posibilidad de mecanizar varios tipos de piezas de formas, dimensiones y materiales diferentes, lo que exige almacenar varios programas de trabajo en las máquinas. El software utilizado debe ser flexible permitiendo introducir todos los cambios necesarios y añadir o quitar los módulos de programación deseados.

ii. **Automatización.** Se tratará de automatizar el proceso de mecanizado (usando máquinas-herramienta de control numérico), el cambio de herramientas y piezas, el transporte de piezas entre los diferentes componentes de la célula, la identificación de las piezas y la comprobación de cotas para asegurar una calidad alta en el producto terminado.

iii. **Productividad.** Se persigue un funcionamiento sin operarios y un incremento de las tasas de utilización de las máquinas, una mayor rapidez en los cambios de herramientas y piezas, así como un menor tiempo de mecanizado. En definitiva, mejorar cuantitativamente el proceso de fabricación de las piezas.

iv. **Fiabilidad del Proceso.** Se refiere al control del desgaste de útiles y herramientas, a las condiciones de corte de las herramientas empleadas, a las desviaciones que puedan producirse como consecuencia de las vibraciones y de las altas temperaturas de funcionamiento, etc. Lo ideal

es que existieran mecanismos que pudiesen detectar y corregir toda esta serie de desviaciones y que dieran avisos de error cuando éstas se salen de la normalidad. En definitiva se trataría de obtener un mantenimiento preventivo eficaz.

v. **Rentabilidad.** En un taller de mecanizado tradicional los tiempos de permanencia no productivos de cada pieza en el taller son muy altos. Con el empleo de las técnicas de fabricación flexible se pretende reducir estos tiempos improductivos con el fin de minimizar los stocks de productos acabados e intentar que el volumen de fabricación coincida con el volumen de pedidos.

En conclusión, un sistema de fabricación flexible deberá ser capaz de hacer frente de una forma efectiva y rápida a los siguientes puntos:

- Modificación de los programas de fabricación (en lo referente a tipos de piezas y cantidad a fabricar de cada una de ellas).
- Permitir los cambios de diseño y especificación de las piezas.
- Posibilidad de funcionamiento sin operarios durante largos espacios de tiempo: ***mano de obra cero***.
- Capacidad de garantizar una calidad cien por cien: ***defectos cero***.
- Posibilidad de conseguir un almacenamiento nulo: ***stock cero***.
- Capacidad del uso del cien por cien de los equipos.
- Mantenimiento preventivo, con el objetivo de conseguir ***cero averías***.
- Capacidad de entrega inmediata de los pedidos: ***plazo de entrega cero***.

1.2.3. Componentes de un FMS.

Un FMS está dividido en los siguientes subsistemas:

Materiales.

Se trata de las piezas a fabricar, las herramientas y los utillajes de fijación a las máquinas. El diseño de las piezas debe revisarse para facilitar su mecanización, eliminar excesos de material y simplificar los montajes. Los sistemas de herramientas que deben utilizarse estarán normalizados y serán de acoplamiento rápido para así reducir los tiempos de cambio de herramienta.

Máquinas.

Dentro de este apartado se engloban las máquinas herramienta de mecanizado, los puestos de montaje y los de verificación. Estos puestos de trabajo estarán dotados de control numérico y de una gran robustez ya que estarán sometidos a unas condiciones de trabajo duras y a un alto porcentaje de utilización. Debido a las bajas tolerancias con las que se trabaja en estos procesos se exigirá también un alto grado de precisión en el guiado (eliminando juegos), y una gran estabilidad térmica mediante un sistema de refrigeración de la herramienta o mediante un proceso de autocorrección de las desviaciones.

La utilización en los puestos de inspección de máquinas de medición de coordenadas y procesos de inspección visual, permite automatizar la verificación final de las piezas y modificar las condiciones de mecanizado en función de los resultados obtenidos en estas pruebas de inspección.

Transporte de material.

Dentro de este apartado se incluyen los cambiadores automáticos de piezas y herramientas, los almacenes y el transporte de piezas entre las diferentes máquinas y el almacén central.

Para poder secuenciar adecuadamente las piezas será necesario un sistema de identificación que permita al ordenador central conocer, en todo momento, las piezas que se están mecanizando, de forma que se ejecute en cada máquina el programa adecuado para cada pieza.

Control del proceso.

Normalmente, estos sistemas son controlados a través de un ordenador central que tendrá a su cargo todas las funciones de alimentación, control y coordinación de los elementos del sistema. Para poder llevar a cabo todas estas funciones necesitaremos paquetes de software que garanticen el funcionamiento adecuado del sistema productivo.

Mano de obra.

La fabricación flexible es un sistema altamente automatizado, pero ello no significa que no necesite de unos recursos humanos para su perfecta operatividad. El factor humano se encargará de vigilar y mantener el sistema en perfecto funcionamiento.

Gestión.

En este apartado se incluyen múltiples aspectos:

i. *Gestión de herramientas*: conocer su desgaste, su vida útil y las posibilidades de reparación que pueden tener.
ii. *Gestión de materiales*: control del flujo de materiales.
iii. *Gestión de recursos, planificación del trabajo y distribución*.
iv. *Gestión del mantenimiento* preventivo de las máquinas.
v. *Estudio de simulación de planes de fabricación*: estudio de colas, saturación de máquinas, almacenes, utilización de máquinas, etc.
vi. *Generación de estadísticas e informes*.
vii. *Transmisión de todos los datos* generados a los ordenadores centrales de gestión.

Sin duda alguna, el subsistema más interesante desde el punto de vista de este proyecto es el **Sistema de Control de Proceso**, por lo que a continuación pasamos a analizarlo con mayor profundidad.

1.2.4. Sistema de control de la FMC.

Para que el conjunto de máquinas y elementos de alimentación y transporte trabajen automáticamente y de forma conjunta, hace falta un sistema que coordine y controle toda la instalación.

El *sistema de gestión* debe ser tal que permita tener en cuenta los siguientes aspectos:

- Enviar los programas de mecanización adecuados para cada proceso elemental.
- Centralizar toda la información disponible del sistema de fabricación para conocer su estado en todo momento.
- Dar las órdenes de ejecución de cada uno de los procesos que deberán realizarse en la fabricación flexible para conseguir el objetivo de producción.
- Ser capaz de reaccionar a posibles imprevistos que puedan surgir en cada proceso: rotura de herramientas, falta de aprovisionamiento, defectos detectados en el control de calidad, etc.
- Elaborar un plan de producción, de forma que se secuencie adecuadamente el paso de cada tipo de pieza por las diferentes máquinas.

El sistema de control está formado por un conjunto de dispositivos electrónicos (actuadores y sensores) conectados a equipos de control numérico y a autómatas programables que a su vez se conectan a un ordenador central a través de una red de comunicaciones. Todos estos elementos de control disponen de unos programas donde se encuentra el código necesario para garantizar un perfecto funcionamiento de cada máquina.

En el ordenador central debe encontrarse el software necesario para la ejecución de todas las tareas y coordinación de toda la célula. Los programas de control contienen la secuencia de operaciones elementales que deben realizarse en cada uno de los equipos que forman la FMC. Además de enviar las órdenes a cada máquina, los programas de control deben recoger y almacenar la información devuelta por las máquinas de la célula. Dicha información va a servir para conocer el estado del sistema de producción en todo momento.

El *sistema de control* es uno de los puntos más críticos y complejos de un FMS, y dicha complejidad crece con la de la propia instalación. La gran multitud de variables que condicionan el diseño de una célula de fabricación flexible, la amplia gama de productos, la variedad en el tamaño de los lotes, el grado de flexibilidad y automatismo que se quiera obtener, etc., hacen que cada sistema de control se deba diseñar de acuerdo con las necesidades propias de cada instalación.

Una FMC puede considerarse dividida en dos partes fundamentales:

i. ***Parte operativa***: realiza las funciones de mecanizado, control de los procesos individuales, la alimentación y el transporte.

ii. ***Parte de control***: encargada de las funciones del control del proceso global y de su coordinación.

Los procesos elementales que conforman la parte operativa están controlados por medio de controles numéricos, autómatas programables, controladores de robots, etc. Para que estos procesos elementales puedan comunicarse entre sí es necesario introducir un sistema de comunicaciones que nos permita un intercambio de información entre los diferentes puestos de la FMC. Además, para poder coordinar la transferencia de información entre todos los procesos elementales será necesario que exista un coordinador (generalmente un ordenador central) que se ocupe de la transmisión de programas, ficheros de datos, órdenes, etc.

Podemos desglosar la estructura de un sistema de control en los niveles que se recogen a continuación:

a. **Nivel 0.** Actuadores y sensores.
b. **Nivel 1.** Controladores de cada máquina.
c. **Nivel 2.** Gestión de automatización de la FMC.
d. **Nivel 3.** Gestión del control de producción de cada FMS.
e. **Nive1 4.** Gestión de la empresa.

En nuestro caso sólo nos ocupamos de una *célula de fabricación flexible*, por lo que dispondremos únicamente de los tres primeros niveles. El *nivel 0*, formado por actuadores y sensores, está gobernado por una serie de controladores (autómatas programables, máquinas de control numérico, etc.). Estos últimos forman el llamado *nivel 1* (controladores de máquinas). A su vez, los elementos del *nivel 1* están interconexionados entre sí y gobernados por un ordenador central que se ocupa de toda la coordinación de la célula (se corresponde con el *nivel 2*). A esta configuración, así definida, se la conoce con el nombre de *Configuración Jerárquica.*

1.2.5. Componentes del sistema de control de la FMC.

Cuando la célula está formada por una única máquina de mecanizado, como por ejemplo un sólo centro de torneado, es la propia unidad de control numérico de esta máquina la que se ocupa del control del sistema. Sin embargo, lo más habitual es que la FMC no sea tan simple e incluya más máquinas. Así, una configuración bastante extendida de lo que puede ser una FMC es la siguiente:

- Dos o tres máquinas de control numérico (ejemplo: un torno y una fresadora).
- Una serie de robots para la alimentación de piezas.
- Un almacén con un sistema de transporte para que circulen las piezas entre las diferentes máquinas.

Al equipo anterior debemos añadir la unidad de coordinación de todas las operaciones, que es la que posibilita las comunicaciones entre todos los controladores. Este sistema de control será, por lo general, un ordenador.

El conjunto de elementos anteriores, unidos a través de un sistema de comunicaciones y gobernados por el ordenador central, constituye una estructura capaz de darnos las siguientes características:

1. Elementos descentralizados.
2. Modularidad.
3. Compatibilidad e integrabilidad.
4. Facilidad de comunicaciones.
5. Rapidez de procesamiento.
6. Facilidad de programación.

Las unidades de control de algunas máquinas-herramienta, además de ofrecer capacidad de almacenamiento de programas de mecanizado y gobernar el funcionamiento de la máquina, pueden también funcionar mediante un sistema de comunicaciones DNC. En este caso es el ordenador central de la FMC el auténtico controlador de la máquina, pues iría enviando la parte de programa de mecanizado que se necesitase en cada momento, no siendo necesario almacenar estos programas en la propia máquina.

En los robots se usan controladores programables cuyo funcionamiento puede ser verdaderamente autónomo (es el propio controlador del robot el que se ocupa de todos los movimientos, giros y desplazamientos del mismo), o bien, a través del sistema de comunicaciones, podemos dejar que el ordenador central se ocupe de su control.

Los sistemas de transporte entre los diferentes elementos, suelen estar gobernados por un autómata programable. Este autómata, además, interacciona con los controladores de las diferentes máquinas para conseguir el sincronismo necesario en la realización de las tareas.

Estos autómatas reciben señales de los controladores de las máquinas-herramienta como pueden ser: señales de *pieza fabricada, máquina libre/máquina ocupada, alarma en el proceso de fabricación,* etc. Todas estas señales se almacenan en posiciones de memoria del autómata, que pueden ser leídas e interpretadas por el ordenador central. La lectura de los estados internos del autómata es posible gracias al sistema de comunicaciones instalado entre todas las máquinas. El ordenador central sólo tiene que enviar ciertas órdenes de acceso a la memoria del autómata a través de su puerto serie para acceder a esta información.

Un autómata industrial es fácil de instalar y de programar (mediante diagrama de contactos, GRAFCET,...), y por lo tanto no necesita de unos grandes conocimientos en informática. Tienen la ventaja de poder leer todo tipo de señales: de entrada, tanto analógicas como digitales y actuar en consecuencia generando una serie de salidas (que también pueden ser tanto analógicas como digitales).

La unidad de control central de la FMC es un ordenador personal (microordenador) que está especialmente programado para el tratamiento de las señales que le llegan a través de su puerto serie de todas las máquinas que forman dicha célula. La lógica del programa actúa de acuerdo a las señales que le llegan, generando otras de salida que irán destinadas a cada una de las máquinas.

Este ordenador central deberá disponer de la velocidad de proceso suficiente para el funcionamiento en tiempo real de la célula, así como de una memoria adecuada y programas de aplicación capaces de facilitar la programación de todas las máquinas-herramienta, robots, manipuladores y autómatas programables existentes en el FMS.

También deberá disponer de una memoria auxiliar (disco duro), para el almacenamiento de los programas de aplicación, de los programas de control numérico que deberán transmitirse a las máquinas y de los ficheros y tablas con los datos relativos al estado de la célula

Además, es deseable que disponga de una pantalla gráfica, a través de la que se pueda informar del estado de la célula de una forma cómoda y visual. De esta forma, de un simple vistazo podemos conocer el estado del sistema de fabricación en cada momento.

1.2.6. Sistema de comunicaciones.

Es, tal vez, una de las partes de la FMC que más relacionada está con el objetivo de este proyecto. Todo el sistema de comunicaciones entre los diferentes niveles de la FMC está a cargo de una red local o de una tarjeta de comunicaciones, tal y como ocurre en la FMC del LOIP. Este problema de interconexión de elementos, ya complejo de por sí, se complica aun más debido a la gran variedad de equipos que poseen características heterogéneas.

En principio, se puede optar por uno de los dos tipos de comunicación básicos, *comunicación en serie* o *comunicación en paralelo*.

- La ventaja de la transmisión *paralela* es la velocidad, pues al transmitirse cada bit que forma el dato por un hilo diferente y todos ellos a la vez, esta puede ser bastante elevada. El problema de este tipo de transmisión es su coste de instalación y que las distancias a las que se puede llegar, sin problemas de pérdidas de información y errores en la línea, son muy pequeñas.

- La otra modalidad, la comunicación *serie*, tiene el problema de la velocidad, pues todos los bits de una palabra se transmiten, uno detrás de otro, por un único hilo, con lo cual la velocidad de transmisión va a ser mucho menor. La ventaja de este tipo de comunicación es que pueden alcanzarse distancias bastante mayores que con la comunicación en paralelo, asegurando un alto grado de fiabilidad en el proceso de transmisión (puede garantizarse una transmisión sin errores a una velocidad de 9600 baudios a una distancia de 15 metros). Es uno de los protocolos más utilizados en ambientes industriales.

Cualquiera que sea la forma de transmisión (serie o paralelo), será necesario que el emisor y el receptor dispongan de una base común de tiempos, o sea, deberá existir una sincronización entre ambos terminales. Atendiendo a la forma en que se realice dicha sincronización podremos distinguir:

i. ***Asíncrona.*** El carácter que se transmite viene precedido por un bit de START o bit de Inicio (a nivel bajo), a continuación se envían todos los bits que forman el carácter (generalmente 7 u 8 bits), y por último se envía un bit de STOP o bit de Parada (a nivel alto), que indica el fin de la transmisión de ese carácter. Por lo tanto, en este tipo de transmisión el sincronismo sólo existe cuando la línea está transmitiendo, o lo que es lo mismo, los elementos emisor y receptor se sincronizan cuando el receptor recibe el bit de START y dejan de estar sincronizados cuando recibe el bit de STOP. Este tipo de sincronización se usa cuando las velocidades de transmisión son bajas (hasta 9600 baudios).

ii. ***Síncrona.*** El emisor y el receptor están continuamente sincronizados debido a que emplean una base de tiempos común. Esta base de tiempo se consigue mediante una inicialización que consiste en el envío de una serie de bits que marcan el comienzo de la sincronización. Este tipo de transmisión admite mayores velocidades que la transmisión asíncrona, pero tiene el problema de necesitar más elementos: son necesarios los modems.

La transmisión también puede ser clasificada según el tipo de línea, o sea, según el tipo de circuito de datos:

i. **Simplex**: se transmite en un sólo sentido (siendo imposible la transmisión en el otro sentido).
ii. **Semiduplex o Halfduplex**: permite la transmisión en ambos sentidos, pero realizándola de forma alternativa, o sea, no permite la transmisión en ambos sentidos simultáneamente.
iii. **Full Duplex**: permite la transmisión simultánea en ambos sentidos.

Para la conexión entre los controles de cada máquina y el controlador de la célula (ordenador central) se requiere una transmisión tipo serie, asíncrona y semiduplex o duplex (según casos). Esta conexión generalmente se hace en estrella, o sea, cada máquina está conectada directamente al ordenador central de la FMC.

Si habláramos de la conexión entre los distintos ordenadores centrales de cada FMC de un taller de fabricación flexible y un ordenador más general que se ocupase del control de toda la producción en el ámbito de una fábrica, las conexiones entre estos equipos serían síncronas y duplex (posibilitando de esa forma una velocidad de transmisión muy elevada).

1.2.7. Software en una FMC.

Es una de las partes más críticas de toda la instalación. Se puede disponer de las mejores máquinas y de los mejores elementos de alimentación y transporte, pero todo ello puede resultar inútil si la parte encargada de coordinar todo el conjunto de elementos (es decir, el software de control) es mediocre o no funciona correctamente.

El desarrollo de los programas de control de la FMC es un proceso largo y costoso que puede suponer hasta el 15% del coste total de la implantación de un sistema de fabricación.

Para poder desarrollar el software debe tenerse un conocimiento muy elevado acerca del funcionamiento de las máquinas, de los elementos de control de cada una, de los autómatas programables y en general de toda la instalación. Muchas veces esta parte de la instalación de la célula se subestima, con lo que el control de la célula no es el adecuado y los resultados obtenidos serán deficientes.

El software de control debe encargarse de las siguientes tareas:

- Coordinar todos los elementos de la célula.
- Confeccionar los programas de mecanizado para las diferentes piezas que queremos fabricar.
- Reaccionar ante imprevistos en el cambio de diseño del producto, en la rotura de herramientas o por averías.
- Debe llevar el control del material en bruto y del que se encuentra en curso de fabricación.
- Debe tener en cuenta la gestión del mantenimiento preventivo de toda la instalación.

Resulta evidente, por lo tanto, que los programas de control de la célula son una pieza clave para su correcto funcionamiento: *Un sistema de fabricación flexible sólo puede hacer lo que los programas de control le permitan. Por este motivo, estas aplicaciones deben ser lo suficientemente flexibles para poder adaptarse a los cambios necesarios y prever nuevas funciones e integraciones con otros sistemas.*

CAPÍTULO SEGUNDO:

COMUNICACIÓN EN SERIE.

2.1. INTRODUCCIÓN.

El correcto funcionamiento de cualquier sistema integrado de producción, y en particular el de la FMC del LOIP, depende en gran medida de la perfecta definición de las comunicaciones. Por otro lado, la transferencia de datos ha sido, sin duda alguna, la gran dificultad a la que nos hemos tenido que enfrentar durante el desarrollo de este proyecto. Por ello, consideramos imprescindible abordar el tema de las comunicaciones con cierta profundidad.

En este capítulo se muestran los aspectos básicos de la comunicación en serie. El conocimiento de estas características nos servirá para estudiar eficazmente la transferencia de datos mediante las funciones de comunicación de la API de Windows (WIN32 Communications API).

También es importante hacer notar el estrecho vínculo que históricamente ha existido entre el puerto serie y el modem. Por ese motivo, cuando utilicemos la palabra **modem** a lo largo del proyecto, nos estaremos refiriendo al controlador remoto de las diferentes máquinas de la célula conectadas a través de una tarjeta *National Instrument PCI-232/8.*

2.1.1. Aspectos básicos de la comunicación en serie.

Los ordenadores personales y los grandes sistemas informáticos necesitan un medio de comunicación con el exterior a través del que poder intercambiar información. En la actualidad existen diversos medios de comunicación, pero todos ellos necesitan de dispositivos controladores de entrada/salida que realicen las funciones de comunicación. De esta forma no será necesario que el microprocesador se ocupe de este trabajo, quedando libre para poder realizar otras tareas.

La comunicación serie, que es la que nosotros vamos a tratar a lo largo de este capítulo, se utiliza cuando no se requiere una velocidad de transmisión de datos elevada. Además, este tipo de comunicación se realiza de una forma sencilla, es poco costosa y permite comunicar ordenadores a través de largas distancias.

En un primer intento por asegurar que un dispositivo serie se entendiera con cualquier otro, la EIA (*Electronics Industries Association*) creó un estándar para definir la señalización eléctrica y las características de conexión de los cables de un puerto serie.

Esta organización coopera con *CCITT (International Telegraph and Telephone Consultive Committee*), para establecer los estándares de la interconexión de los diferentes equipos. En 1969 se publicó la recomendación número 232 en su versión C (*Recommended Standard 232-C*), también conocida como RS-232-C. Posteriormente se crearon otra serie de recomendaciones más robustas y para distancias más largas, tales como RS449, RS422 y RS423, que son especiales para ambientes industriales en los que existan condiciones ambientales duras, tales como grandes vibraciones provocadas por máquinas, campos electromagnéticos importantes, etc.

Los ordenadores personales siguen usando la recomendación básica RS-232-C, que define la función de cada señal en el interface serie y su conexión física. Este estándar de comunicaciones define dos clases de conexiones serie: una para *terminales* (OTE: Data Terminal Equipment) y otra para *equipos de comunicaciones* (DCE: Data Communications Equipment).

Una conexión RS-232-C emplea normalmente un conector de 25 pines con un macho en un extremo y una hembra en el otro, pero también podemos encontrar otros tipos de conexiones bien sea usando un DB-25 (25 pines) o un DB-9 (9 pines). Los ordenadores actuales suelen incorporar dos puertos serie, uno con cada tipo de conector para asegurar compatibilidades.

Pero, ¿por qué tantos cables, si sólo son necesarios dos para formar un circuito?. En la realidad sólo 9 de los cables (de los 25 posibles) son utilizados por la mayoría de los ordenadores para la interconexión con el mundo exterior (de ahí la existencia de un puerto serie DB-9). Dos de estas señales se ocupan de la transmisión y recepción de datos, otra es la referencia eléctrica (*Tierra Lógica*: esta señal es obligatoria para establecer un punto de referencia de señal igual para los dos terminales que estén conectados). También puede usarse otra señal de *Tierra Física* para evitar sobrecargas que puedan producirse en algún punto de la red. El resto de señales se usan para el control de la transmisión. En definitiva, una conexión serie consiste en varios circuitos independientes entre sí, existiendo dos circuitos de datos (envío y recepción) y varios circuitos de control.

1. ***Circuitos de datos.*** Por estos circuitos se van a transmitir los datos de forma bidireccional. Los estándares RS-232-C describen minuciosamente cómo se codifica la información en las líneas de datos. Los componentes eléctricos generan y captan cambios en los niveles de tensión de los cables. Para las señales digitales (que tienen una forma de onda cuadrada) se imponen tensiones en los rangos de +3 a +25 y de -3 a -25. Los ordenadores emplean normalmente tensiones de ±12 a ±15 voltios para las cotas de las ondas cuadradas. El intervalo de tensiones que va entre +3 y -3 voltios es un área de transición, cada

paso por esta área de transición señala un bit. Las señales digitales pierden potencia y también su forma cuadrada ante la presencia de campos eléctricos como los generados por maquinaria, por lo que existe un límite para la distancia que dichas señales pueden alcanzar. Se define una distancia máxima para cada velocidad usada que dependerá de la capacidad que tenga el cable de transmisión (para un cable apantallado que tenga una capacidad de 150 pF/m podremos alcanzar una longitud de 60 metros para una velocidad de hasta 9600 baudios[1] sin tener pérdidas en la línea).

2. ***Circuitos de Control.*** Hay cinco circuitos de control en el interface RS-232-C. Estos circuitos permiten al ordenador conocer en todo momento el estado de la línea y el estado del equipo al que está conectada. Todos estos circuitos de control pueden ser ignorados engañando a los equipos que están conectados, por lo que pueden darse diferentes situaciones de interconexión.

Para poder establecer la comunicación serie va a ser necesario realizar una transformación que convierta los datos en paralelo (que es como los manejan interiormente los ordenadores) a datos en formato serie (para poder enviar consecutivamente por el puerto serie los bits que forman cada carácter). Para realizar esta operación el ordenador dispone de un chip especial denominado UART (*Universal Asynchronous Receiver Transmitter*). Este chip es el corazón de la comunicación serie RS-232 y sirve para conectar los buses internos del ordenador con el puerto serie.

[1] Los **baudios** miden la velocidad de señalización o de modulación de un modem. Los modems envían datos como una serie de tonos a través de la línea telefónica. Los tonos se 'encienden' y 'apagan' para indicar un 1 ó un 0 digital. El baudio es el número de veces por segundo que esos tonos se 'encienden' o 'apagan'. Los modems antiguos enviaban un bit de datos por cada cambio en el tono, o sea, un bit por baudio. De ahí que se confundan los términos baudio y bit por segundo (BPS). Estos términos no son equivalentes, dado que existen modems más modernos que pueden enviar varios bits por baudio.

La definición de la función que realiza cada señal se recoge en la normativa RS-232-C de comunicación en serie. Un resumen de esta se muestra en la tabla 2.1.

DB-25	DB-9	Función	Nombre	Entrada/Salida
1	1	Tierra de Protección (tierra física)		
2	3	Datos Transmitidos	TD	Salida
3	2	Datos Recibidos	TR	Entrada
4	7	Petición para enviar datos	RTS	Salida
5	8	Listo para enviar datos	CTS	Entrada
6	6	Equipo de datos preparado	DSR	Entrada
7	5	Señal de referencia (tierra lógica)		
8	1	Detector de portadora	DCD	Entrada
20	4	Terminal de datos preparado	DTR	Salida
22	9	Indicador de llamada	RI	Entrada

Tabla 2.1. *Función de cada pin del conector serie.*

Cada una de estas señales forma un circuito totalmente independiente de los demás.

A continuación vemos el uso y el funcionamiento de aquellos circuitos destinados al control de la transmisión (circuitos de control). Todos los pines que se indican están referidos al conector DB-25, que es el que se utiliza en la FMC del LOIP.

Circuito Request To Send (RTS) y Clear To Send (CTS): Pin 4 y 5.

Estos dos pines controlan el flujo de datos entre los equipos interconectados. El ordenador coloca el pin RTS en ON cuando está preparado para recibir datos y lo apaga cuando no está listo para aceptar datos (lo pone en OFF). Por el contrario el modem coloca en ON la señal CTS cuando está listo para recibir datos y la apaga cuando no puede aceptar más datos.

Con estas dos líneas conseguimos que el modem y el ordenador se comuniquen a muy alta velocidad y a la vez sirven para pedir la transferencia de datos cuando sea necesario. Estas señales generalmente están unidas entre sí, o sea: el pin 4 (RTS) del ordenador irá conectado al pin 5 (CTS) del modem y viceversa.

Circuito Data Terminal Ready (DTR) y Data Set Ready (DSR): Pin 20 y 6.

DTR (pin 20) es una salida del ordenador: indica al otro equipo que éste está encendido y listo para establecer una comunicación. Normalmente el otro equipo (bien sea un modem o un autómata u otro ordenador) responde a esta señal poniendo en ON la línea DSR (pin 6), para que el primer ordenador sepa que el otro está listo para recibir los datos. Lo más frecuente es que el pin 20 (DTR) de un ordenador esté conectado al pin 6 (DSR) del otro ordenador y viceversa.

Circuito Carrier Detect (DCD) y Ring Indicator (RI): Pin 8 y 22.

La señal DCD (Detector de portadora) indica que el modem ha conectado y reconocido los tonos de otro modem. La señal RI (pin 22), permite saber al ordenador que la línea telefónica del modem está indicando una llamada (que está sonando). En nuestra célula de fabricación flexible no se utiliza esta señal, pues no estamos usando ningún modem.

2.1.2. Transferencia de datos en serie.

Los datos viajan por los circuitos como una cadena de ceros y unos. Normalmente cada dato serie contiene 7 u 8 bits de datos por carácter. Si se usan 7 bits de datos podremos codificar un total de 128 caracteres diferentes (código ASCII), si usamos 8 bits podremos llegar a codificar un total de 256 caracteres (código ASCII extendido).

Los datos serie viajan de bit en bit, es decir, un bit detrás de otro. Cada bit es un dígito binario (un uno o un cero). La transferencia serie de datos puede realizarse de dos formas diferentes: asíncrona y síncrona.

1. ***Comunicación asíncrona.*** La mayoría de las comunicaciones con ordenador son asíncronas, es decir, no hay un período de tiempo definido entre los caracteres transmitidos. Se envía primero el bit de orden más bajo (el de menor peso), seguido por los bits de orden superior. Antes de comenzar el envío de los bits que forman el carácter debemos enviar una señal de comienzo (llamado bit *Start* o bit de *Inicio*), que consiste en poner a nivel bajo la línea durante un tiempo determinado. A continuación enviaremos el bit de menor peso significativo del carácter y, siguiendo a éste, el resto de bits que forman dicho carácter. Una vez transmitidos estos bits se puede, opcionalmente, enviar un bit que indique la paridad (par o impar) para enviar posteriormente los bits de parada (bits *Stop*).

2. ***Comunicación síncrona.*** La otra forma es hacer la transferencia de forma síncrona. Ésta se emplea sólo para tareas especiales, como las conexiones a alta velocidad entre LAN y LAN *(Redes de área local: Local Area Network*). En un sistema síncrono los bits de cada carácter se envían en una cadena sin ningún tipo de espacio o pausa. Todo el flujo de información está sincronizado con un pulso de reloj enviado en una línea diferente. Para conseguir el sincronismo se incluyen bits especiales al principio y al final de la transmisión. Y para conseguir la misma frecuencia de temporización será necesario también transmitir el pulso de reloj.

2.2. ASPECTOS BÁSICOS SOBRE LA TRANSFERENCIA DE DATOS MEDIANTE LAS FUNCIONES DE COMUNICACIÓN DE LA API DE WINDOWS.

Para realizar la transferencia de datos a través del puerto serie de Windows debemos usar las funciones de comunicación establecidas en *Win32 Communications API*, y aunque existen diversas formas de hacerlo, nosotros sólo veremos aquellas que se han utilizado en este proyecto.

2.2.1. Abrir un puerto serie.

En Windows 95, los puertos serie y otros dispositivos de comunicación se tratan igual que ficheros. Los puertos serie se abren, se cierran, se lee de ellos y se escribe en ellos usando las mismas funciones que se usan para manejar ficheros.

Una sesión de comunicaciones comienza con una llamada a la función *CreateFile*. Dicha función "abre" el puerto serie para lectura, escritura o para ambas. Como es normal en Windows, *CreateFile* retorna una dirección de memoria para usarla en posteriores operaciones sobre el puerto ya abierto.

La función *CreateFile* es compleja. Una razón para su complejidad es su propósito general. Se puede usar *CreateFile* para abrir ficheros ya existentes, crear nuevos ficheros y abrir dispositivos que no son ficheros, como puertos serie, puertos paralelos y modems.

```
CreateFile ( szDevice, fdwAccess, fdwShareMode, lpsa, fdwCreate, fdwAttrsAndFlags, hTemplateFile);
```

El primer parámetro, *szDevice,* es el nombre lógico del puerto serie a abrir, tal como COM1 o COM2. El segundo parámetro, *fdwAccess,* especifica el tipo de acceso al puerto. Al igual que los archivos, los puertos serie se pueden abrir para leer, escribir o para ambas cosas. Si damos a *fdwAccess* el valor GENERIC_READ abrimos el puerto para lectura mientras que con el valor GENERIC_WRITE abrimos el puerto para escritura. Uniendo los dos valores

anteriores con el operador OR conseguimos abrir el puerto para leer y escribir. La mayor parte de los puertos de comunicación son bidireccionales, por lo que lo más común es encontrarse esta última opción:

```
fdwAccess = GENERIC_READ | GENERIC_WRITE;
```

El tercer parámetro, *fdwShareMode*, especifica los atributos de acceso compartido al puerto. Este parámetro es usado por los ficheros que pueden ser compartidos por múltiples aplicaciones. Los puertos serie, al no poder usarse en modo compartido, deben tener un valor 0. Esta es una de las principales diferencias entre los archivos y los dispositivos de comunicación. Si una aplicación ya ha abierto el puerto, cuando la aplicación actual llama a la función *CreateFile,* ésta devuelve un error ya que dos aplicaciones no pueden compartir el mismo puerto. Sin embargo, múltiples funciones de la misma aplicación pueden compartir la dirección devuelta por la función *CreateFile*, y dependiendo de los atributos de seguridad establecidos, esta dirección puede ser heredada por las aplicaciones hijas de la aplicación que abrió el puerto.

El cuarto parámetro, *lpsa*, hace referencia a la estructura de atributos de seguridad, que define tanto dichos atributos como la forma en que la dirección del puerto puede ser heredada por los hijos de la aplicación que abrió el puerto. Estableciendo este parámetro a NULL asignamos al puerto los atributos de seguridad establecidos por defecto.

El quinto parámetro, *fdwCreate*, especifica que hacer si en *CreateFile* se hace referencia a un archivo que ya existe. Dado que los puertos serie siempre existen, *fdwCreate* valdrá OPEN_EXISTING. Esto hace que Windows no intente crear un puerto nuevo, sino abrir un puerto que ya existe.

El sexto parámetro, *fdwAttrsAndFlags,* describe varios atributos del puerto. Para los puertos serie el único valor de interés es FILE_FLAG_OVERLAPPED. Cuando establecemos este valor, el puerto de entrada/salida puede trabajar de forma asíncrona. El último parámetro, *hTemplateFile,* no se usa cuando se abre un puerto, por lo que debe valer NULL.

2.2.2. Cerrar un puerto serie.

Cerrar un puerto serie es mucho más sencillo que abrirlo. Simplemente se llama a la función *CloseHandle* con la dirección devuelta por *CreateFile* como único parámetro.

```
CloseHandle(hComm);
```

Al terminar de usar un puerto serie tendremos que cerrarlo. Si olvidáramos cerrar el puerto, éste permanecería abierto y el resto de aplicaciones no podrían abrirlo para usarlo.

2.3. CONFIGURACIÓN DE UN PUERTO SERIE.

Configurar un puerto en Windows 95 es más complicado que en anteriores versiones de Windows. Sin embargo, con esta mayor complejidad ha llegado una mayor capacidad. Windows 95 mejora los drivers de los puertos serie al mismo tiempo que les proporciona una mayor integración.

La configuración comienza con la determinación de las capacidades del puerto. Esto permite a una aplicación evitar valores incorrectos que el puerto no puede soportar. La estructura COMMPROP es la encargada de devolver las capacidades del puerto.

2.3.1. La estructura COMMPROP.

COMMPROP quizá sea el elemento más interesante y útil que se ha añadido a la familia de estructuras de datos para las comunicaciones bajo Windows 95. La estructura COMMPROP contiene los valores permitidos para el puerto, por ejemplo, los valores permitidos para la tasa de baudios, el número de bits de datos, el número de bits de parada y el método de paridad.

Recuperar las propiedades del puerto.

La función *GetCommProperties* recupera la información sobre el puerto serie contenida en la estructura COMMPROP. Los contenidos de dicha estructura son estáticos (ninguna aplicación tiene permiso para modificar su información), por lo que no hay ninguna función para escribir los valores del puerto en la estructura COMMPROP. Los parámetros de *GetCommProperties* son la dirección del puerto y un puntero a la estructura COMMPROP.

```
GetCommProperties(hComm , lpCommProp);
```

La estructura COMMPROP es la mostrada a continuación:

```
typedef struct _COMMPROP{                  // cmmp.
        WORD wPacketLength;                // tamaño de la estructura, en bytes.
        WORD wPacketVersion                // versión de la estructura.
        DWORD dwServiceMask;               // funcionalidad del puerto.
        DWORD dwReserved1;                 // no se usa.
        DWORD dwMaxTxQueue;                // tamaño máximo permitido del buffer de entrada (en bytes).
        DWORD dwMaxRxQueue;                // tamaño máximo permitido del buffer de salida (en bytes).
        DWORD dwMaxBaud;                   // tasa máxima de baudios permitida para el puerto.
        DWORD dwProvSubType;               // especifica el tipo de dispositivo de comunicación.
        DWORD dwProvCapabilities;          // capacidades soportadas por el puerto.
        DWORD dwSettableParams;            // especifica que valores del puerto son configurables.
        DWORD dwSettableBaud;              // valores permitidos de tasa de baudios.
        WORD wSettableData;                // valores permitidos de número de bits de datos.
        WORD wSettableStopParity;          // número de bits de parada y modo de paridad permitidos.
        DWORD dwCurrentTxQueue;            // tamaño actual del buffer de transmisión.
        DWORD dwCurrentRxQueue;            // tamaño actual del buffer de recepción.
        DWORD dwProvSpec1;                 // no se usa en puertos serie RS-232.
        DWORD dwProvSpec2;                 // no se usa en puertos serie RS-232.
        WCHAR wcProvChar[1];               // no se usa en puertos serie RS-232.
} COMMPROP, *LPCOMMPROP;
```

Seguidamente comentaremos algunos de los miembros de la estructura.

Definición de las propiedades del puerto.

El miembro *wPacketVersion* especifica la versión de la estructura COMMPROP. Las diferentes versiones de Windows así como otros sistemas operativos de Microsoft usan diferentes versiones de COMMPROP con una interpretación diferente de sus miembros, por lo que es importante analizar primero *wPacketVersion* para estar seguros de que interpretamos correctamente la estructura. En Windows 95, la versión está fijada en 0xffff.

El miembro *dwServiceMask* generalmente define la funcionalidad del puerto. Este miembro toma el valor SP_SERIALCOMM para los puertos serie.

dwMaxTxQueue y *dwMaxRxQueue* contienen el tamaño máximo permitido (en bytes) del buffer de entrada y del buffer de salida del puerto, respectivamente. Si este miembro devuelve 0, significa que el puerto no establece ninguna restricción para el tamaño del buffer.

dMaxBaud es la máxima tasa de datos que el puerto soporta. Este número puede ser fijado o programado. Si soporta la programación *dwMaxBaud* se pone a BAUD_USCR. En caso contrario, *mxMaxBaud* debe tomar uno de los siguientes valores:

BAUD_075	75 bps
BAUD_110	110 bps
BAUD_134_5	134.5 bps
BAUD_150	150 bps
BAUD_300	300 bps
BAUD_600	600 bps
BAUD_1200	1200 bps
BAUD_1800	1800 bps
BAUD_2400	2400 bps
BAUD_4800	4800 bps
BAUD_7200	7200 bps
BAUD_9600	9600 bps
BAUD_14400	14400 bps
BAUD_19200	19200 bps
BAUD_38400	38400 bps

BAUD_56K	56 Kbps
BAUD_57600	57600 bps
BAUD_115200	115200bps
BAUD_128K	128 Kbps

dwProvSubType especifica el tipo de dispositivo de comunicación con mayor detalle que *dwServiceMask*. Para los puertos serie, este campo toma el valor PST_RS232. A continuación se listan otros dispositivos soportados:

PST_FAX	Fax
PST_LAT	Protocolo LAT
PST_MODEM	Modem
PST_NETWORK_BRIDGE	Puente de trabajo en red sin especificar
PST_PARALLELPORT	Puerto paralelo
PST_RS422	Puerto RS-422
PST_RS423	Puerto RS-423
PST_R5449	Puerto RS449
PST_SCANNER	Escaner
PST_TCPI_TELNET	Protocolo Telnet TCP/IP
PST_UNSPECIFIED	Sin especificar
PST_X25	Estándar X25

A continuación hablaremos del miembro más útil, del miembro que hace que la estructura COMMPROP sea realmente interesante, el miembro *dwProvCapabilities*. Este miembro describe las capacidades soportadas por el puerto. Cada valor especifica un simple bit en el miembro *dwProvCapabilities*. Los valores y sus correspondientes capacidades se muestran a continuación:

PCF_16BITMODE	El puerto soporta el modo especial de 16-bits.
PCF_DTRDSR	El puerto soporta las señales DTR y DSR.
PCE_INTTIMEOUTS	El puerto soporta el intervalo timeout.
PCF_PARITY_CHECK	El puerto soporta el chequeo de paridad.
PCF_RLSD	El puerto soporta la señal RLDS.
PCF_RTSCTS	El puerto soporta las señales RTS y CTS.
PCF_SETXCHAR	El puerto soporta establecer los valores XON/XOFF .
PCF_SPECIALCHARS	El puerto soporta los caracteres especiales.
PCF_TOTALTIMEOUTS	El puerto soporta el timeouts total.
PCF_XONXOFF	El puerto soporta el flujo de control XON/XOFF .

Los puertos serie RS-232 estándar y los drivers de los puertos serie de Windows soportan la mayor parte de las capacidades mostradas arriba, con la excepción de PCF_16BITMODE y PCF_SPECIALCHARS.

El miembro *dwSettableParams* especifica que valores del puerto son configurables. Los valores y su interpretación se muestran a continuación:

SP_BAUD	La tasa de baudios se puede configurar.
SP_DATABITS	El número de bits de datos es configurable.
SP_HANDSHAKING	El coloquio (flujo de control) es configurable.
SP_PARITY	Podemos configurar el modo de paridad.
SP_PARIY_CHECK	Podemos habilitar/deshabilitar el chequeo de paridad.
SP_RLSD	La señal RLSD puede ser configurada
SP_STOPBITS	Podemos establecer el número de bits de parada.

Los puertos serie RS-232 estándar y los drivers de los puerto serie de Windows soportan la configuración de todos los valores mostrados anteriormente.

Los miembros *dwSettableBaud*, *wSettableData* y *wSettableStopParity* contienen los valores permitidos para la tasa de baudios, el número de bits de datos, el número de bits de parada y el modo de paridad. Los valores permitidos de tasa de baudios son los mostrados en la página 44. Windows permite un valor máximo de 38 Kbps para los puertos serie RS-232 estándar. Los valores permitidos para el número de bits de datos son los que se muestran a continuación:

DATABITS_5	5 bits de datos
DATABITS_6	6 bits de datos
DATABITS_7	7 bits de datos
DATABITS_8	8 bits de datos
DATABITS_16	16 bits de datos
DATABITS_16X	Tamaño especial

Los miembros *dwCurrentTxQueue* y *dwCurrentRxQueue* contienen, respectivamente, el tamaño actual de los buffers de transmisión y recepción. Los miembros *dwProvSpec1* y *dwProvSpec2* y *wcProvChar* no se usan en los puertos serie RS-232.

2.3.2. La estructura DCB.

La estructura COMMPROP nos cuenta qué valores son soportados por el puerto. COMMPROP es una estructura puramente informativa y no puede usarse para modificar los valores del puerto. Aquí es donde entra en juego la estructura DCB (*Device Control Block*). La estructura DCB cumple el mismo propósito en Windows 95 que en otras versiones anteriores de Windows: es un contenedor para los valores actuales del puerto que pueden ser usados para configurar un puerto mediante programas. La estructura DCB para Windows 95 es la siguiente:

```
typedef struct _DCB {
        DWORD DCBlength;
        DWORD BaudRate;
        DWORD fBinary: 1;
        DWORD fParity: 1;
        DWORD fOutxCtsFlow: 1;
        DWORD fOutxDsrFlow: 1;
        DWORD fDtrControl: 2;
        DWORD dDsrSensivity: 1;
        DWORD fTXContinueOnXoff: 1;
        DWORD fOutX: 1;
        DWORD flnX: 1;
        DWORD fErrorChar: 1;
        DWORD fNull: 1;
        DWORD fRtsControl: 2;
        DWORD fAbortOnError: 1;
        OWORD fDummy2: 17;
        WORD wReserved;
        WORO XonLim;
        WORD XoffLim;
        BYTE ByteSize;
        BYTE Parity;
        BYTE StopBits;
        char XonChar;
        char XoffChar;
        char ErrorChar;
        char EofChar;
```

```
    char EvtChar;
    WORD wReserved1;
} DCB;
```

La definición de los miembros de esta estructura es la siguiente:

- *DCBLength.* Establece el tamaño, en bytes, de la estructura DCB.
- *BaudRate.* Define la tasa de baudios actual.
- *FBinary.* Especifica si está permitido el modo binario. Este miembro debe ser siempre TRUE en Windows 95.
- *FParity.* Precisa si se permite el chequeo de paridad.
- *FOutxCtsFlow.* Distingue si la señal CTS es gestionada por el control de flujo de transmisión. Cuando este miembro es TRUE y CTS está en OFF la transmisión se suspende hasta que CTS vuelve al estado ON.
- *FOutxDsrFlow.* Especifica si la señal DSR se gobierna mediante el control de flujo de transmisión. Cuando este miembro es TRUE y DSR está en OFF la transmisión se suspende hasta que DSR vuelve al estado ON.
- *FDtrControl.* Un valor DTR_CONTROL_DISABLE establece a la señal DTR en OFF. Un valor DTR_CONTROL_ENABLE pone a DTR en ON, y un valor DTR_CONTROL_HANDSHAKE permite un coloquio DTR.
- *FDsrSensitivity.* Cuando este miembro está en TRUE, los bytes recibidos mientras DSR está en OFF son ignorados.
- *FTxContinueOnXoff.* Establece si se detiene la transmisión cuando el buffer de recepción está lleno y el driver ha transmitido el carácter XoffChar.
- *FOutX.* Cuando este miembro vale TRUE, la transmisión se interrumpe cuando el carácter XoffChar es recibido y comienza cuando se recibe el carácter XonChar.

- *FInX.* Si este miembro vale TRUE, el carácter XoffChar es enviado cuando el número de bytes en el buffer de recepción está entre el valor XoffLim y el número máximo (completamente lleno), y el carácter XonChar es enviado cuando el número de bytes del buffer de recepción se mueve entre el valor mínimo (completamente vacio) y XonLim.
- *FErrorChar.* Si este miembro y fParity valen TRUE, los bytes recibidos con un error de paridad son reemplazados con el carácter especificado por el miembro ErrorChar.
- *FNull.* Cuando este miembro vale TRUE, no se descarta ningún byte recibido.
- *FRtsControl.* La señal RTS pasa a valor OFF cuando este valor se establece a RTS_CONTROL_DISABLE. Por el contrario, la señal RTS pasa a valer ON cuando este miembro vale RTS_CONTROL_DISABLE. Un valor RTS_CONTROL_ HANDSHAKE indica al driver del puerto que cambie la señal RTS a ON cuando el buffer de recepción esté por debajo de la mitad y que lo cambie a OFF cuando esté lleno hasta las tres cuartas partes. Un valor RTS_CONTROL_TOGGLE cambia RTS a ON cuando hay bytes esperando en el buffer de recepción y lo cambia a OFF en caso contrario. Un valor 0 equivale a RTS_CONTROL_HANDSHAKE.
- *FAbortOnError.* Cuando este miembro vale TRUE, las operaciones de lectura y escritura terminan cuando se produce un error.
- *fDummy2.* No se usa.
- *WReserved.* No se usa. Debe valer 0.
- *XonLim.* Precisa el número mínimo de bytes que se permiten en el buffer de recepción antes de que se envíe el carácter XON.
- *XoffLim.* Define el número mínimo de bytes disponibles que se permiten en el buffer de recepción antes de que se envíe el carácter XOFF.

- *ByteSize.* Fija el número de bits de datos actualmente usados por el puerto.
- *Parity.* Determina el método de paridad actualmente usado por el puerto. Los valores posibles son EVENPARITY, MARKPARITY, NOPARITY y ODDPARITY.
- *StopBits.* Especifica el número de bits de parada que el puerto usa en ese momento. Los valores posibles son ONESTOPBIT, ONE5STOPBITS, y TWOSTOPBITS.
- *XonChar, XoffChar.* Precisan el valor de los caracteres XON y XOFF para ser usados tanto en la transmisión como en la recepción.
- *ErrorChar.* Este carácter reemplaza el carácter recibido con un error de paridad.
- *EofChar.* Este carácter señala el final del dato cuando no se usa el modo binario.
- *EvtChar.* Se genera un evento cada vez que este carácter sea recibido.
- *wReserved1.* No se usa.

Como cambiar los valores del puerto.

Para leer los valores DCB relativos al puerto, debemos llamar a la función *GetCommState*. Esta función recibe como parámetros la dirección del puerto abierto y un puntero a la estructura DCB, a la cual retornará la información. La función complementaria a *GetCommState* es *SetCommState* que también requiere la dirección del puerto y el puntero a la estructura DCB. *SetCommState* escribe el contenido de la estructura DCB en los valores del puerto. Las funciones comentadas se muestran a continuación:

```
BOOL GetCommState (hComm , &dcb);

BOOL SetCommState (hComm , &dcb);
```

Cambiar los valores del puerto no es tan simple como leer con *GetCommState*, cambiar los valores del puerto y después escribirlos con *SetCommState*. Se requiere además otro paso adicional: comparar los nuevos valores con los valores permitidos para el puerto. Hay que asegurarse de que los nuevos valores están en la estructura COMMPROP. Si el puerto soporta los nuevos valores, estos serán copiados en la estructura DCB y escritos con *SetCommState*. El siguiente ejemplo cambia la tasa de baudios a 19,2 Kbps.

```
GetCommProperties (hComm , &commprop);
GetCommState(hComm , &dcb);
// si el puerto soporta los 19,2 Kbps realizamos el cambio, en caso contrario no cambiamos
if (commprop.dwSettableBaud & BAUD_19200)
        dcb.dwBaudRate=19200;
SetCommState(hComm , &dcb);
```

Flujo de control y coloquio.

La mayoría de los miembros de la estructura DCB son fáciles de comprender, pero los miembros que precisan de un coloquio y un flujo de control pueden dar lugar a confusión.

Empecemos con el miembro *fDtrControl*, que se usa para poner la señal DTR en ON o en OFF, o para especificar un *coloquio DTR*. Cuando el puerto se usa con un modem o un puerto remoto, el cambio de la señal DTR a ON le indica al modem que el puerto está preparado para recibir datos.La señal DTR actúa como una señal de habilitación, por lo que cuando la señal DTR cambia a OFF el modem deja de enviar bytes por el puerto. El driver del puerto cambia la señal DTR a OFF cuando el buffer de recepción se llena, suspendiéndose la recepción hasta que el buffer de recepción vuelve a ser utilizable.

Por ejemplo, si el puerto 1 está conectado al puerto 2 usando un cable de modem tipo NULL, normalmente conectaremos la señal DTR del puerto 1 con la señal DSR del puerto 2, y viceversa. Cuando el puerto 1 cambia DTR a ON, DSR

cambia a ON en el puerto 2, lo que indica al puerto 2 que el puerto 1 está preparado para recibir bytes. El puerto 2 puede entonces transmitir bytes al puerto 1 hasta que este puerto cambie DTR a OFF.

fDtrControl puede ser usado junto a *fOutxDsrFlow* para controlar la transmisión y recepción de bytes hacia y desde un puerto. La figura 2.1. ilustra el flujo de transmisión del proceso descrito anteriormente que puede ser el resultado del siguiente código:

```
fOutxDsrFlow=TRUE;
fDtrControl=DTR_CONTROL_HANDSHAKE;
```

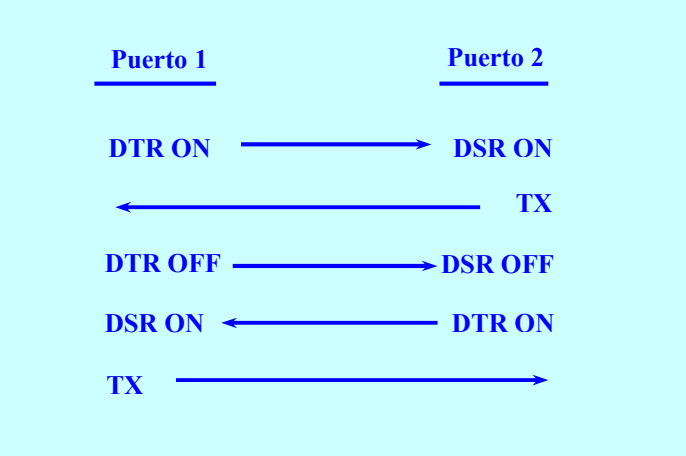


Figura 2.1. *Flujo de transmisión entre dos puertos*[B13].

Si se usa el coloquio DTR, *fDsrSensitivity* debe establecerse en TRUE. De esta forma, los bytes recibidos mientras la señal DSR está en OFF son ignorados.

Muchos de los miembros de la estructura DCB están relacionados con el *flujo de control XON/XOFF*. Este flujo de control usa unas señales fijas especiales para la transmisión y recepción de bytes entre dos puertos. Estas señales son caracteres especiales usados *dentro de la corriente de datos* para indicarle al puerto cuando transmitir o recibir.

Por ejemplo, cuando el buffer de recepción está lleno, podemos enviar el carácter ASCII 19 para indicar al puerto remoto que deje de enviar datos. Un carácter especial con este propósito es el llamado XOFF. Cuando vuelve a haber espacio en el buffer de recepción, podemos enviar el carácter ASCII 17 para indicar al puerto remoto que reanude el envío de datos. Un carácter especial con este propósito es el llamado XON. Los caracteres XON y XOFF son especificados por *XonChar* y *XoffChar*.

Podemos establecer el miembro *fOutX* a TRUE para indicar al driver del puerto que suspenda la transmisión de bytes cuando el puerto remoto envíe XOFF. La transmisión se reanudará cuando el puerto remoto envíe el carácter XON.

También podemos fijar el miembro *fInX* a TRUE para indicar al driver del puerto que envíe el carácter XOFF al puerto remoto cuando el número de bytes en el buffer de recepción exceda un cierto límite. Este límite viene especificado por *XoffLim*.

XoffLim no precisa el número máximo de bytes permitidos en el buffer de recepción antes de enviar el carácter XOFF, más bien, este valor especifica el mínimo número permitido de bytes disponibles que quedan libres en el buffer de recepción antes de que se envíe el carácter XOFF. El número máximo de bytes permitidos en el buffer de recepción se obtiene, por lo tanto, de restar *XoffLim* al tamaño total del buffer (buffsize). *XoffLim*, también conocido como *Marca de Marea Alta*, es el límite que origina la interrupción de la transferencia de datos a través del puerto.

Fijando *fInX* en TRUE también le indicamos al driver del puerto que envíe el carácter XON para reanudar la transmisión una vez que el número de bytes en el buffer de recepción ha descendido por debajo de un cierto límite, especificado por *XonLim*. A *XonLim* también se le conoce como *Marca de Marea Baja*.

Cuando *fInX* vale TRUE, XOFF es enviado cuando el número de bytes en el buffer de recepción excede la señal de marea alta definida por *XoffLim*, tal y como se muestra en la figura 2.2.

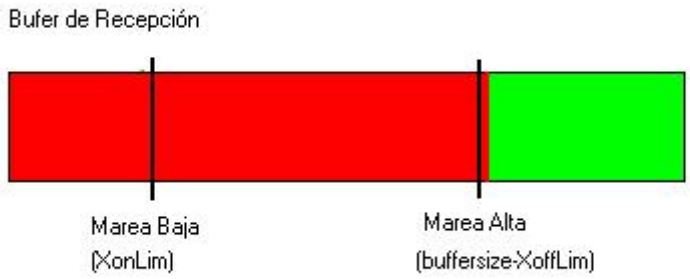


Figura 2.2. *El buffer de recepción se ha llenado por encima de la Señal de Marea Alta* [B13].

Se envía el carácter XON cuando el número de bytes que hay en el buffer de recepción desciende por debajo de un valor límite dado por *XonLim*, tal y como se muestra en la figura 2.3.

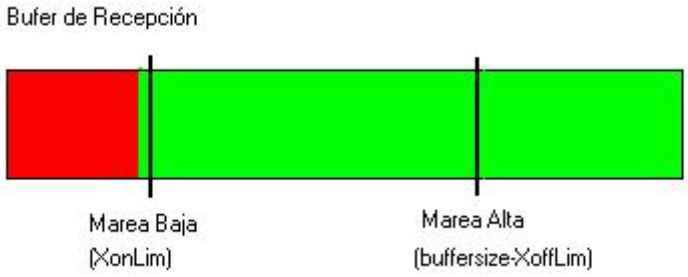


Figura 2.3. *El buffer de recepción vaciado por debajo de la Marca de Marea Baja* [B13].

Cuando el miembro *fTXContinueOnOff* toma el valor TRUE, la transmisión no se ve afectada por las operaciones comentadas anteriormente. En otras palabras, la transmisión continúa incluso después de recibir el carácter XOFF. Cuando *fTXContinueOnOff* toma el valor FALSE, la transmisión se suspende y se reanuda en sincronía con la recepción.

El tipo final de coloquio que puede especificarse en la estructura DCB es el *coloquio RTS/CTS*. Este es el ***clásico*** flujo de control de hardware de las comunicaciones serie RS-232. Usando el miembro *fRtsControl*, la señal RTS puede ser cambiada a ON (RTS_CONTROL_DISABLE) o a OFF (RTS_CONTROL_DISABLE) o podemos establecer el coloquio RTS (RTS_CONTROL_ HANDSHAKE).

Cuando permitimos el coloquio RTS, un puerto cambia la señal RTS a ON para indicar que el puerto remoto está listo para recibir bytes. El puerto cambia RTS a OFF cuando el buffer de recepción está lleno, para indicar al puerto remoto que detenga el envío de datos. El puerto remoto cambia la señal CTS a ON para indicar que está preparado para recibir bytes. Cuando CTS está a OFF, el puerto remoto no está preparado para recibir datos, por lo que el puerto no transmitirá hasta que la señal CTS cambie a ON. Con el coloquio RTS/CTS, el miembro *fOutxCtsFlow* se pone en TRUE para indicar al puerto que no transmita ningún byte mientras la señal CTS esté en OFF.

Cambiar los valores más frecuentes.

Los valores que más frecuentemente se cambian de la estructura DCB son la tasa de baudios, el método de paridad, el número de bits de datos y el número de bits de parada. Una función que conviene utilizar para cambiar estos valores es *BuildCommDCB*:

```
BuildCommDCB(szSettings,&DCB);
```

Los parámetros de esta nueva función son una variable de tipo string que contiene todos los nuevos valores y un puntero a la estructura DCB dentro de la cual serán aplicados los nuevos valores. Pero, ¿por qué se usa un string para pasar los valores?. La respuesta es que una variable tipo string es práctica, cómoda y familiar. Consideremos el siguiente ejemplo:

```
“baud=12 parity=N data=8 stop=1”
```

Esta variable string cambia la tasa de baudios a 1200bps, elimina el chequeo de paridad y establece el número de bits de datos a 8 y el número de bits de parada a 1. Podemos omitir los parámetros identificativos pasando sólo los valores en el mismo orden que en el string. El ejemplo que vimos anteriormente puede reducirse a "12,N,8,1".

La tabla siguiente muestra los valores válidos para cada parámetro pasado a la función *BuildCommDCB.*

Baudios	Paridad	Bits de datos	Bits de parada
11 ó 110 =110 bps	n = none	5	1
15 ó 150 =150 bps	e = even	6	1,5
30 ó 300 =300 bps	o = odd	7	2
60 ó 600 =600 bps	m = mark	8	
12 ó 1200 =1200 bps	s = space		
24 ó 2400 =2400 bps			
48 ó 4800 =4800 bps			
96 ó 9600 =9600 bps			
19 ó 19200 =19200 bps			

Tabla 2.2. *Valores válidos para cada parámetro del control pasado a BuildCommDCB* [B13]**.**

Deberemos comprobar la estructura COMMPROP para estar seguros de que el puerto serie soporta los valores antes de intentar cambiarlos con *BuildCommDCB y SetCommState.*

2.3.3. La estructura COMMCONFIG.

Otra estructura de configuración que merece la pena comentar es COMMCONFIG. COMMCONFIG contiene una estructura DCB que es idéntica en todo con respecto a la devuelta por *GetCommState*. La estructura es la que se muestra a continuación:

```
typedef struct _COMMCONFIG {
        DWORD dwSize;                   // tamaño de la estructura entera.
        WORD  wVersion;                 // versión de la estructura.
        WORD  wReserved;                // alineación.
        DCB dcb;                        // DCB (Device Control Block).
        DWORD dwProviderSubType;        // valor ordinario para identificar  el formato de la estructura .
                                        // de datos  definida por el proveedor.
        DWORD dwproviderOffset;         // especifica los offset dados por el proveedor.
        DWORD dwProviderSize;           //tamaño del campo de datos especificado por el proveedor.
        WCHAR wcProviderData[1];        // datos especificados por el proveedor.
} COMMCONFIG, *LPCOMMCONFIG;
```

COMMCONFIG viene con una función muy útil para configurar el puerto, llamada *CommConfigDialog*. Esta función muestra una ventana modal para cambiar la tasa de baudios, los bits de datos, el método de paridad, los bits de parada y el método de flujo de control para el puerto. También contiene un botón para restablecer los valores por defecto.

La función devuelve los valores seleccionados en el miembro *dcb* del COMMCONFIG. Una llamada típica a *CommConfigDialog* es la mostrada a continuación.

```
CommConfigDialog(lpszComName, hWnd, &cc);
```

El primer parámetro es el nombre del puerto a configurar, el segundo es la dirección de la ventana del cuadro de diálogo, y el tercero apunta a la estructura COMMCONFIG.

CommConfigDialog no cambia normalmente los valores del puerto (no se pasa la dirección del puerto como parámetro). El código siguiente muestra como cambiar los valores del puerto con la subsiguiente llamada a *SetCommState*. El ejemplo no muestra la parte donde los valores devueltos en COMMCONFIG son chequeados contra los valores soportados en COMMPROP aunque siempre deberemos incluir esto en nuestro código.

```
COMMCONFIG cc;
#define WINDOWS95_COMMCONFIG_VERSION          0x100
...
cc.dwSize = sizeof(COMMCONFIG);
cc.wVersion = WINDOWS95_COMMCONFIG_VERSION;
// la función devuelve false si  hay un error
If (!CommConfigDialog ("COM2"', hWnd, &cc))break;
...
// No se muestra: debemos comparar los valores devueltos con los valores de  COMMPROP
// para estar seguros de que el puerto los soporta
dcb.BaudRate = cc.dcb.BaudRate;
dcb.BaudRate = cc.dcb.ByteSize;
dcb.BaudRate = cc.dcb.StopBits;
dcb.BaudRate = cc.dcb.Parity;
SetCommState (hComm. &dcb);
```

2.3.4. Los valores timeout.

¿Qué ocurre si la conexión se interrumpe en medio de la recepción de datos, o si por otra parte, la transmisión simplemente se suspende? ¿Qué ocurre cuando los bytes son transmitidos y se espera una conversación que nunca se produce?. A menos que tengamos cuidado, estas circunstancias pueden bloquear las comunicaciones.

Afortunadamente, Windows 95 dispone de una protección contra este tipo de problemas: podemos especificar el tiempo máximo que debe esperar una función de lectura o escritura antes de darse por vencida. Esto puede hacerse gracias a los valores *Timeout*, usando la estructura COMMTIMEOUTS tal y como se muestra a continuación.

```
typedef struct  _COMMTIMEOUTS {
        DWORD ReadIntervalTimeout;
        DWORD ReadTotalTimeoutMultiplier;
        DWORD ReadTotalTimeoutConstant;
        DWORD WriteTotalTimeoutMultiplier;
        DWORD WriteTotalTimeoutConstant;
} COMMTIMEOUTS, *LPCOMMTIMEOUTS;
```

Hay dos tipos de timeouts. El primer tipo, denominado *intervalo timeout*, se aplica solamente a la lectura desde un puerto. Este especifica cuanto tiempo ha de transcurrir entre la lectura de dos caracteres. Windows inicia un reloj interno cada vez que se recibe un carácter. Si el tiempo excede el intervalo timeout antes de que llegue el siguiente carácter, la función de lectura se da por vencida.

El segundo tipo de timeout, el *timeout total*, se aplica tanto a la lectura como a la escritura a través del puerto. Este timeout se pone en funcionamiento cuando el tiempo total para leer excede un umbral. Por ejemplo, si el timeout total está establecido en 10 ms (milisegundos), y una aplicación intenta leer del puerto 10 bytes, el tiempo timeout total se activa si se tarda más de 10 ms en leer los 10 bytes. No importa si 9 de los 10 bytes han sido leídos en un tiempo inferior a 10 ms; el timeout se activará y la función de lectura sólo devolverá 9 bytes.

La ecuación del timeout.

Los valores del timeout se especifican en milisegundos. El intervalo timeout solamente se aplica a la lectura desde un puerto, y se especifica en el miembro *ReadIntervalTimeout*. El tiempo total timeout se especifica independientemente para leer y escribir, y está dividido en dos partes: una constante y un multiplicador. Windows calcula el timeout total para leer y escribir a partir de dicha constante y del multiplicador utilizando las fórmulas siguientes:

```
ReadTotalTimeout = (ReadTotalTimeoutMultiplier * bytes_to_read) + ReadTotalTimeoutConstant;
WriteTotalTimeout = (WriteTotalTimeoutMultiplier * bytes_to_read) +  WriteTotalTimeoutConstant;
```

Esta ecuación hace que el timeout total sea una herramienta flexible. En lugar de un valor fijo, el timeout total varía de acuerdo al número de bytes a leer o escribir. Como punto de partida, consideremos sólo la escritura de bytes. Cuando *WriteTotalTimeoutConstant* vale 0, la fórmula para el timeout total de escritura se reduce a:

```
WriteTotalTimeout = (WriteTotalTimeoutMultiplier * bytes_to_read);
```

El resultado es que si el valor medio de tiempo que se tarda en escribir cada byte excede *WriteTotalTimeoutMultiplier*/*bytes_to_write*, el timeout se activa. Ésta es una forma muy adecuada para indicar el requisito medio de funcionamiento cuando se transmiten bytes.

Ahora consideremos lo que ocurre cuando se añade a la fórmula la constante *WriteTotalTimeoutConstant*. El timeout total todavía está directamente relacionado con el funcionamiento medio de la función de escritura, pero ahora el funcionamiento promedio tiene un umbral superior definido por la constante.

Finalmente, *WriteTotaltimeoutMultiplier* puede valer 0. En este caso, el timeout total se concreta en una constante. Esto puede ser útil en aquellas situaciones en las que la escritura deba completarse dentro de un tiempo fijo, independientemente del número de bytes involucrados.

Si no queremos usar los valores timeout debemos poner todos los miembros de la estructura COMMTIMEOUTS a cero. Esto es peligroso, ya que una función de lectura o escritura no terminará hasta que los bytes requeridos hayan sido escritos o leídos.

También podemos usar nuestra función para conocer cuántos bytes esperan en la cola de recepción, o para forzar que la función de lectura retorne inmediatamente cuando no haya bytes esperando en dicha cola. El truco consiste en establecer *ReadTotalTimeout* a MAXWORD, y poner *ReadTotalTimeout Multiplier* y *ReadTotalTimeoutConstant* a cero.

Para comprender un poco mejor todo lo que hemos comentado, nada mejor que unos ejemplos. En el primero, nosotros queremos establecer el timeout para la escritura de bytes, siendo 5 ms el tiempo medio aproximado permitido para la escritura de un byte. También deseamos definir un umbral de forma que el timeout total para la escritura nunca sea inferior a 50 ms. Los siguientes valores consiguen este propósito.

```
ReadTotalTimeoutMultiplier=5;
ReadTotalTimeoutConstant=50;
```

Supongamos ahora que queremos leer bytes en un bucle. Si tarda más de 5 ms en leer bytes consecutivos nosotros queremos que la función de lectura regrese inmediatamente a la parte de programa que la llamó para de este modo poder seguir con la ejecución.

```
ReadIntervalTimeout=5;
```

Para forzar que la función de lectura retorne inmediatamente cuando no haya bytes esperando en la cola de recepción, deberemos usar la sentencia siguiente.

```
ReadIntervalTimeout=MAXWORD;
```

En ningún caso usaremos *ReadIntervalTimeout=0*; ya que provoca que la función espere respuesta indefinidamente.

Implementación de los valores timeout.

Una vez que hemos decidido los valores timeout, llamamos a la función *SetCommTimeouts* para implementarlos. *SetCommTimeouts* toma como parámetros la dirección de memoria del puerto abierto y un puntero a la estructura COMMTIMEOUTS que contiene los valores.

Para cambiar sólo alguno de los valores y dejar los otros sin tocar, primero llamamos a la función *GetCommTimeouts* para llenar la estructura con los valores actuales. Entonces, cambiamos los valores deseados y finalmente llamamos a *SetCommTimeouts* para implementarlos, tal y como se muestra a continuación:

```
GetCommTimeouts ( hComm, &timeouts);
Timeouts.ReadIntervalTimout=5;
SetCommTimeouts ( hComm, &timeouts);
```

Debemos hacer constar, nuevamente, que como acto previo a cualquier modificación debemos comprobar el miembro *dwProvCapabilities* de la estructura COMMPROP para estar seguros de que el puerto soporta los cambios en los tiempos de descanso.

2.4. LEER Y ESCRIBIR DATOS.

En Windows 95, para leer y escribir en un puerto usamos las mismas funciones que se utilizan para leer y escribir datos de un archivo. En otras palabras, hay tantas técnicas para leer y escribir en un puerto como para leer y escribir en un archivo: polling, E/S síncrona, E/S asíncrona, E/S mediante eventos, etc. A continuación vamos a ver algunas de estas técnicas de lectura.

2.4.1. Polling.

Sin duda alguna, se trata de la técnica más directa para leer de un puerto. Una función intenta continuamente leer del puerto (Poll). Si hay bytes presentes en el buffer de recepción del puerto, la función lee. En caso contrario, la función continua. Por el contrario, esta técnica consume mucho tiempo de la CPU dado que la función está continuamente ejecutándose buscando bytes (incluso cuando no los hay). La ventaja de esta técnica es que es directa y fácil de entender. El siguiente código nos muestra cómo recibir bytes del puerto.

```
bool FuncionLectura (HANDLE puerto)
{
DWORD nBytesRead;
char buffer_entrada[100];
COMMTIMEOUTS to;
// las siguientes dos líneas comprueban que el puerto soporta el intervalo timout
if (!(cp.dwProvCapabilities & PCF_INTTIMEOUTS))
        return false;  // error: no soporta el intervalo timeout
 //las siguientes líneas permiten que la función de lectura retorne inmediatamente
//si no hay bytes esperando en la cola de recepción del puerto
memset (&to, 0, sizeof(to));
to.ReadIntervalTimeout = MAXDWORD:
SetCommTimeouts (puerto, &to);
// intentamos leer del puerto si es posible
if (!ReadFile(puerto, buffer_entrada, 100, &nBytesRead, NULL)){
        // handle error
        locProcesscommError(GetLastError, puerto);
        return false;
```

```
}else {
        // si hay datos esperando proceso la información
        if (nBytesRead) FuncionProceso(buffer_entrada, nBytesRead);
}
// limpio los datos pendientes en el buffer de recepción
PurgeComm(puerto, PURGE_RXCLEAR);
return true;
}
```

Como se podrá apreciar fácilmente, el código anterior tiene una parte que nos resulta familiar y otra que no lo es. Empecemos con la parte familiar. El código anterior configura el timeout del puerto para que la función de lectura regrese inmediatamente si no hay datos en el buffer. Esto lo conseguimos estableciendo el intervalo de lectura timeout al MAXDWORD. Se puede observar cómo el código comprueba la estructura COMMPROP con la línea *if(!(cp.dwProvCapabilities & PCF_INTTIMEOUTS))* para estar seguros de que el puerto soporta el intervalo timeout antes de intentar modificarlo.

Ahora tratemos la parte que no nos resulta familiar. Para leer los datos del puerto se utiliza la función *ReadFile*, cuya sintaxis es la siguiente:

```
ReadFile (hComm , inBuff , nBytes , &nBytesRead , &overlapped );
```

El primer parámetro que se pasa a esta función es la dirección del puerto previamente abierto. El segundo parámetro es el buffer donde se depositarán los bytes leídos por el puerto (hay que estar seguros de reservar el suficiente espacio para todos los datos que queremos leer).

El tercer parámetro, *nBytes*, es el número de bytes que se espera leer, mientras que el cuarto parámetro es el número de bytes leídos hasta el momento. Hay varias razones por las que el número de bytes leídos puede ser menor que el número de bytes que se quieren leer. Por ejemplo, puede darse el caso de que se produzca un error antes de que todos los bytes sean leídos, o bien que el intervalo timeout expire antes de que todos los bytes sean leídos. Nunca deberemos asumir que todos los bytes han sido leídos, por lo que siempre tendremos que comparar *nBytesRead* con el valor que deseamos leer.

El último parámetro es un puntero a una estructura *overlapped*. La función del ejemplo lee el número de bytes disponibles en la cola de recepción hasta un máximo de 100.

Si ocurre algún error durante la lectura, éste es procesado por la función *locProcessCommError* (las funciones con el prefijo *loc* son locales a la aplicación). Cuando la función *ReadFile* ha finalizado con éxito, si *nBytesRead* no es cero los bytes son procesados mediante la *FuncionProceso*.

Al final de la función *FuncionLectura* se llama a *PurgeComm*. *PurgeComm* es una función de limpieza: termina cualquier lectura o escritura que esté en espera y también limpia los buffers de E/S. El código usa *PurgeComm* para limpiar el buffer de entrada de aquellos posibles valores que no han sido leídos especificando la acción PURGE_RXCLEAR.

```
PurgeComm(puerto, PURGE_RXCLEAR);
```

Otros posibles valores para la acción son PURGE_TXABORT, que impide la escritura de aquellos bytes que se encuentren en espera; PURGE_RXABORT, que termina toda lectura en espera, y PURGE_TXCLEAR que limpia el buffer de transmisión.

Una desventaja de esta técnica polling es que siempre intenta leer exactamente *nBytes* bytes (100 en nuestro ejemplo) cada vez que llamamos a la función *ReadFile*. Esto provoca una ineficiencia dado que en ocasiones hay más de 100 datos esperando ser leídos, en cuyo caso deberemos llamar a la función *ReadFile* varias veces.

Una técnica mejor consiste en comprobar el número de bytes disponibles en el buffer de lectura y leer exactamente ese número. Pero, ¿cómo podemos conocer el número de bytes que hay en el buffer de recepción?. La función *ClearCommError* determina ese número de bytes.

ClearCommError posee una doble función. Su primer propósito es, como su nombre sugiere, el de limpiar cualquier condición de error. Su segundo propósito, que es el que nos interesa ahora, es el de determinar el estado del puerto.

```
ClearCommError (hComm, &dwErrorMask, &Comstat);
```

El primer parámetro de *ClearCommError* es la dirección del puerto abierto. El segundo parámetro es un puntero a la máscara de la posible condición de error que es devuelta por la función (esta condición de error la trataremos con mayor profundidad más adelante). El último parámetro es el que ahora nos concierne; se trata de un puntero a la estructura COMSTAT que contiene información útil sobre el estado del puerto. La estructura se muestra a continuación.

```
typedef struct  _COMSTAT {
        DWORD fCtsHold : 1;        // Transmisión esperando a la señal CTS.
        DWORD fDsrHold : 1;        // Transmisión esperando a la señal DSR.
        DWORD fRlsdHold : 1;       // Transmisión esperando a la señal RLSD.
        DWORD fXoffHold : 1;       // Transmisión esperando a que el carácter XOFF sea recibido .
        DWORD fXoffSent   1;       // Transmisión esperando a que el carácter XON sea enviado.
        DWORD fEof : 1;            // el carácter EOF fue enviado.
        DWORD fTxim : 1;           // hay bytes esperando a la transmisión.
        DWORD fReserved: 25;       // reservado.
        DWORD cblnQue;             // número de bytes en el buffer de entrada.
        DWORD cbOutQue;            // número de bytes en el buffer de salida.
} COMSTAT, *LPCOMSTAT;
```

Los primeros cinco miembros de COMSTAT proporcionan información acerca de la causa del fin de la transmisión. Pueden ocurrirle cosas anómalas a nuestra aplicación, y estos miembros son los encargados de contarnos porqué los bytes no se mueven del buffer.

Dependiendo del tipo de coloquio involucrado, la transmisión puede cesar cuando la señal CTS esté en OFF, la señal DSR esté en OFF, o cuando la señal RLDS esté en OFF. Los miembros que indican estos condiciones son respectivamente *fCtsHold*, *fDsrHold* y *fRlsdHold*. Si utilizamos el flujo de control XON/XOFF, *fXoffHold* y *fXonHold* nos indicarán si la transmisión está esperando a que el carácter XOFF sea recibido o a que el carácter XON sea enviado. El último caso sólo ocurre cuando *fTxContinueOnXoff* es falso en la estructura DCB.

Como ejemplo, supongamos que el puerto está configurado para un control de flujo a través de las señales CTS y RTS. No se transmitirán datos hasta que la señal CTS se ponga en ON. Supongamos que nos surge un problema y que los datos del buffer de transmisión están esperando, sin salir del buffer como se supone que deberían actuar, ¿cómo podemos darnos cuenta de que esto es lo que está ocurriendo?. Llamando a la función *ClearCommError* y comprobando el miembro *cbOutQueue*, ya que él contiene el número de datos que esperan en el buffer de transmisión. Si este número no cambia durante un largo periodo de tiempo (una fracción de segundo o un segundo), la transmisión ha cesado. La aplicación deberá entonces comprobar *fCtsHold* para analizar si el puerto está esperando a que la señal CTS cambie a ON. Si *fCtsHold* tiene el valor TRUE, el puerto remoto no ha cambiado la señal CTS a ON.

Otros miembros de COMSTAT son *fEof*, que se pone en TRUE cuando recibimos el carácter *fin-de-archivo (Eof: end-of-file)*, y *fTxim*, que especifica si los datos están esperando en el buffer de transmisión. Finalmente, están *cbInQue* y *cbOutQue*, que contienen el número de bytes que esperan ser leídos y escritos.

En los párrafos anteriores hemos descubierto cómo recibir información mediante la técnica polling usando la función *ReadFile*. Para escribir a través de un puerto emplearemos la función *WriteFile* que se muestra a continuación:

```
WriteFile ( hComm, outBuff, nToWrite, &nActualWrite, &overlapped);
```

El primer parámetro es la dirección del puerto anteriormente abierto. El segundo parámetro, *outBuff*, es un puntero al buffer donde se almacenan los datos que van a ser escritos. El tercer parámetro, *nToWrite*, es el número de bytes a

escribir, el cuarto parámetro apunta al número actualmente escrito y el quinto es un puntero a la estructura OVERLAPPED (no nos hace falta conocer nada sobre esta estructura ya que sólo se utiliza con la transmisión síncrona, que no hemos visto ni veremos).

Cuando transmitimos gran cantidad de datos, es útil tener una herramienta que suspenda la transmisión sin perturbar el contenido del buffer. La función *SetCommBreak* sirve a este propósito: suspende inmediatamente la transmisión incluso si el buffer está lleno de bytes. El contenido de este buffer no se ve alterado por la llamada a *SetCommBreak*.

2.4.2. Control E/S mediante eventos.

En ocasiones es útil conocer si una condición ocurre sin tener que comprobar el estado del puerto. Por ejemplo, estaría bien que Windows nos pudiese avisar si una cierta señal cambia de estado para así poder llevar a cabo una determinada acción. Existe una forma de hacer esto, es decir, Windows 95 puede avisar a nuestra aplicación cuando se cumpla una cierta condición conocida como evento.

Los eventos que Windows 95 puede presentarnos son devueltos por la función *GetCommMask*. Para añadir o para modificar la lista de eventos que Windows da a conocer, se usa *SetCommMask*. La llamada a estas funciones es como sigue a continuación:

```
GetCommMask ( hComm, &dwMask);

SetCommMask ( hComm, dwMask);
```

El primer parámetro es la dirección del puerto abierto. El segundo parámetro es la máscara de uno o más eventos que se deben esperar. Estos eventos se describen a continuación:

EV_BREAK	Se ha detectado un Break en la entrada.
EV_CTS	La señal CTS cambió de estado.
EV_DSR	La señal DSR cambió de estado.
EV_ERR	Se ha producido un Error en la línea de estado.
EV_RING	Se ha detectado una señal Ring
EV_RLSD	La señal RLSD cambió de estado.
EV_RXCHAR	Se ha recibido un carácter y se ha posicionado en el buffer de entrada.
EV_RXFLAG	El carácter de evento ha sido recibido y posicionado en el buffer de entrada.
EV_TXEMPTY	Ya se ha enviado el último carácter del buffer de salida.

Por ejemplo, si queremos que se genere un evento cada vez que la señal CTS cambie deberemos escribir:

```
SetCommMask ( hComm, EV_CTS);
```

Después de especificar el evento de interés con *SetCommMask*, la aplicación esperará a que ocurra el evento llamando a la función *WaitCommEvent*, tal y como se muestra a continuación.

```
WaitCommEvent ( hComm, &dwEvent, &overlapped);
```

WaitCommEvent, como *ReadFile* y *WriteFile*, puede operar síncrona o asíncronamente dependiendo de si especificamos la estructura OVERLAPPED como tercer parámetro. Cuando ponemos este parámetro a NULL, la función se comporta síncronamente y no retorna hasta que el evento especificado en *SetCommMask* ocurre. *WaitCommEvent* devuelve el evento en *dwEvent*.

Usando eventos, una aplicación no necesita controlar continuamente el puerto en busca de los bytes recibidos. En lugar de esto, la aplicación entra en un estado de espera, en el que se consumen muy pocos ciclos de la CPU, hasta que la función *WaitCommEvent* retorna.

2.5. ERRORES.

Las comunicaciones son un tema bastante complejo, y los errores pueden ocurrir por una gran variedad de razones. Cuando aparece un error, debemos llamar siempre a la función *ClearCommError* para determinar precisamente el tipo de error que ha ocurrido. Podemos usar *ClearCommError* para determinar la información sobre el estado del puerto o para obtener detalles sobre lo que puede ir mal cuando enviamos o recibimos información. La llamada a la función es como sigue:

```
ClearCommError ( hComm, &dwErrorMask, &comstat);
```

El segundo parámetro contiene la máscara del error que ha ocurrido. Esta máscara puede incluir uno o más de los valores que se muestran a continuación:

CE_BREAK	Condición de Break.
CE_FRAME	Error de Estructura.
CE_IOE	Error general de E/S.
CE_MODE	El modo requerido no es soportado, o la dirección de comunicación es inválida.
CE_OVERRUN	Se ha producido una sobrecarga.
CE_RXOVER	Se ha producido una sobrecarga en el buffer de recepción.
CE_RXPARITY	Error de paridad.
CE_TXFULL	El buffer de transmisión está lleno.
CE_DNS	El dispositivo paralelo no ha sido seleccionado.
CE_PTO	Se ha producido una interrupción en el dispositivo paralelo
CE_OOP	El dispositivo paralelo está sin papel.

Si el miembro *fAbortOnError* de la estructura DCB está activo, debemos llamar a la función *ClearCommError* cada vez que la función de comunicación devuelva FALSE (indica que se ha producido un error). Cuando *fAbortOnError* está activo y se produce un error de comunicación, cualquier lectura o escritura pendiente se cancelan y no se permiten nuevos procesos de lectura o escritura hasta que no se llame a la función *ClearCommError*.

CAPÍTULO TERCERO:

LAS MÁQUINAS DE LA FMC DEL LOIP.

3.1. INTRODUCCIÓN.

El presente capítulo pretende describir brevemente cada grupo de máquinas que componen la FMC del LOIP. En concreto, analizaremos los siguientes puntos:

a) Las máquinas-herramienta.

b) El almacén.

c) Los robots.

d) El autómata.

e) El sistema de transporte.

f) Los sistemas auxiliares.

Primero estudiaremos las características generales más relevantes de cada grupo de máquinas y, posteriormente, describiremos los aspectos más interesantes de los componentes de la célula del LOIP. Estas últimas particularidades nos servirán para poder comprender mejor algunas características de la aplicación FMC desarrollada en este proyecto.

3.2. MÁQUINAS-HERRAMIENTA DE CONTROL NUMÉRICO.

En los últimos años, se ha producido en las máquinas-herramienta un gran avance cualitativo con el fin de integrarse dentro de los FMS. Su incorporación en una célula de fabricación implica que gran parte de las tareas que realizaba el operario se lleven a cabo de forma automática. En este sentido, un instrumento que ha contribuido a su flexibilización ha sido el *Control Numérico*.

El control numérico podría definirse como un instrumento flexible de automatización de una máquina, que controla su funcionamiento mediante un lenguaje especial de programación basado en números, letras y símbolos. Estos símbolos constituyen un programa de instrucciones preparado para poder realizar una determinada tarea.

El control numérico es, por lo tanto, una de las bases de la automatización de los talleres dentro de la nueva filosofía de fabricación flexible y de un entorno CIM.

Sin embargo, esta incorporación no hubiese sido posible sin la evolución de determinados elementos de las máquinas-herramienta. En los últimos años se ha conseguido dotar de gran flexibilidad el cambio de herramientas, se han mejorado sustancialmente los sistemas de sujeción de las piezas, su manipulación, etc.

Otra de las grandes evoluciones en este tipo de máquinas reside en los dispositivos de accionamiento y control, y en las funciones de mecanizado. Todas estas mejoras nos han proporcionado máquinas más robustas, rápidas, fiables y capaces de desarrollar un mayor número de operaciones.

3.2.1. Evolución de las máquinas-herramienta.

Las máquinas-herramienta convencionales requerían de una atención continua por parte del operario que tenía que ocuparse de tareas tales como:

- Transportar, cargar, fijar y descargar las piezas.
- Buscar y cambiar las herramientas.
- Fijar los avances y las velocidades.
- Determinar la secuencia correcta de mecanizado.
- Limpieza de la máquina, evacuación de virutas, etc.

La incorporación del CN (Control Numérico) supuso la eliminación de buena parte de estas tareas por parte del operario. Sin embargo, estas máquinas seguían sin gozar de flexibilidad, pues toda la lógica de funcionamiento se basaba en válvulas electrónicas, relés y transistores con lo que era imposible almacenar un programa en memoria para una posterior ejecución, pues estos elementos no disponían de tal memoria ni de una unidad de cálculo. En definitiva, seguíamos teniendo una estructura rígida, lenta y poco flexible.

Fue a finales de los años 80 cuando nació el CNC (Control Numérico por Ordenador). Gracias al control numérico por ordenador se pudo empezar a desarrollar funciones de control que residieran en la memoria del ordenador y que pudieran ser fácilmente modificables. Se podía adaptar el programa contenido en la memoria de la máquina según las necesidades, lo que supuso dar un paso adelante en el diseño de procesos mucho más flexibles (fáciles de modificar).

Posteriormente surgió el DNC (Control Numérico Distribuido o Control Numérico Directo) como solución a los problemas derivados de la escasa capacidad de almacenamiento de los dispositivos de la época. En el DNC el microprocesador de la máquina se conecta a un ordenador central mediante un sistema de comunicaciones en tiempo real, de forma que se pueden enviar a la máquina las instrucciones del programa de mecanizado eliminándose el transporte mediante disquetes.

Sin embargo, el DNC está actualmente en desuso debido, principalmente, al empleo de redes locales de comunicación y al aumento de la capacidad de los dispositivos de almacenamiento de datos (consecuencia del vertiginoso avance informático de los últimos años).

3.2.2. Generalidades sobre las máquinas-herramienta.

Las máquinas-herramienta son máquinas especiales, diseñadas y construidas para poder realizar los distintos tipos de mecanizado que se pueden presentar en la práctica.

Una máquina-herramienta consta de dos partes principalmente:

- *Máquina*. Es un conjunto de órganos y mecanismos capaz de realizar dos funciones:
    1. Sujetar adecuadamente la pieza que va a ser mecanizada.
    2. Transmitir los movimientos necesarios para que la herramienta pueda realizar su trabajo.
- *Herramienta*. Es el útil que al entrar en contacto con la pieza realizará el mecanizado.

Los movimientos que pueden darse en una máquina-herramienta son de corte y de avance; la correcta combinación de ambos permitirá la formación de virutas en condiciones adecuadas para obtener un rendimiento elevado.

Las condiciones en las que se desarrolla el trabajo en las máquinas dependen de los valores asignados a los factores de corte que son tres: velocidad de corte, avance y profundidad de pasada.

3.2.3. Prestaciones de las máquinas-herramienta.

Las características requeridas de precisión y repetibilidad durante muchas horas seguidas de funcionamiento, unido a unas velocidades y esfuerzos de corte elevados, exigen de estas máquinas unas grandes prestaciones de rigidez, precisión de guiado y estabilidades térmica y dinámica.

Rigidez. La incorporación de herramientas fabricadas en materiales carburos o cerámicos ha permitido el aumento de las velocidades de corte y avance de la herramienta y la profundidad de pasada. Pero esto ha supuesto un aumento paralelo en las solicitaciones mecánicas a las que se ven sometidos todos los elementos de la máquina. Ha sido necesario, por tanto, incrementar la rigidez para disminuir al máximo las deformaciones estáticas y aumentar la capacidad de absorción de las vibraciones. Para ello se recurre al empleo de estructuras, formadas por la mesa, los montantes y los ejes portaherramienta, muy compactas que permitan desplazamientos cortos sin condiciones extremas de voladizo. Se emplean bastidores de acero en lugar de fundición. Incluso se llegan a emplear bastidores de hormigón armado para absorber mejor las vibraciones.

Estabilidad Térmica. La diferencia de temperatura entre la pieza, la herramienta y las demás partes de la máquina puede provocar perdidas de precisión. Para evitarlo se diseñan circuitos que expulsan liquido de refrigeración sobre la zona de mecanizado reduciendo la temperatura y ayudando a la evacuación de virutas. En algunas máquinas se dirigen chorros de líquido por toda la pieza y no sólo en el punto que se está mecanizando. Se utilizan grandes caudales de incluso 200 l/min para refrigerar pieza, herramienta y máquina.

Precisión en el Guiado de la Herramienta. Es esencial para conseguir precisión y repetibilidad en el mecanizado. Para alcanzar precisiones altas se intentan eliminar los juegos mediante montajes pretensados en las guías y se utilizan materiales especiales para mitigar el rozamiento.

3.2.4. Clasificación del control numérico.

La clasificación de los *sistemas de control numérico* más empleada actualmente se basa en el tipo de contorno que podamos realizar y más concretamente, en su complejidad.

- *Sistema punto a punto*. Sólo tiene en cuenta el punto final a donde queremos llevar la herramienta, y no la trayectoria que ésta deba seguir para alcanzar dicho punto. Sólo interesa que la herramienta alcance con rapidez y precisión el punto deseado.

- *Control numérico paraxial*. Permite el mecanizado continuo en direcciones paralelas a los ejes de las máquinas no pudiendo mecanizar en otras direcciones diferentes. Este tipo de control también puede realizar el control punto a punto anterior. Su uso es mínimo en la actualidad.

- *Control numérico de contorno*. Puede mecanizar contornos de cualquier forma. El elemento terminal de la máquina-herramienta va a poder seguir cualquier tipo de trayectoria continua por complicada que sea. Para poder conseguirlo será necesario que el controlador de los ejes de la máquina sea capaz de regular el movimiento de todos los ejes de forma simultánea.

Al *control numérico de contorno* también se le denomina *control numérico continuo* y es el más complejo de los tres. El método de cálculo de los puntos a los cuales deberá ir la herramienta se realiza mediante una interpolación. Este tipo de control puede funcionar como los dos anteriores.

3.2.5. Componentes básicos de una máquina de control numérico.

Podríamos decir que los elementos más importantes de una máquina de control numérico son los siguientes:

- Programa almacenado en memoria.
- Unidad de control (es la que se encarga de ejecutar el programa y controlar la máquina).
- Máquina-herramienta.

El programa de la máquina-herramienta.

Esta formado por una serie de números, letras y símbolos que describen, para cada pieza a mecanizar, las operaciones que debe realizar la máquina-herramienta. Contiene información geométrica relacionada con la pieza y con las herramientas (coordenadas y dimensiones), así como información acerca de velocidades de avance, rotación y características de refrigeración. Las acciones que puede realizar una máquina-herramienta pueden resumirse en las siguientes:

- Posicionamiento de la pieza y de la herramienta.
- Fijar la velocidad de rotación y avance de los útiles.
- Controlar los cambios de piezas y herramientas.

El programa de mecanizado contiene todas las acciones que deben ser realizadas en una secuencia dada, con el fin de obtener la pieza deseada. Para una programación eficiente será necesario tener conocimientos profundos sobre las características de la máquina (potencia, velocidades, esfuerzos,...), características de las piezas a fabricar (dimensiones, forma, acabado superficial,...), características de las herramientas disponibles (formas, dimensiones, acciones que realizan,...) y, por supuesto, las características del propio lenguaje de programación de control numérico.

Para la realización de este programa de control suele utilizarse un lenguaje de alto nivel pues es más fácil de modificar y corregir que un lenguaje en código máquina (lo que realmente entiende la máquina). Estos lenguajes consisten en un paquete de software cuyo objeto es facilitar la tarea del programador a la hora de definir y especificar la geometría de la pieza a mecanizar, así como las instrucciones necesarias para generar el programa en código máquina a partir del fichero fuente desarrollado en este lenguaje de alto nivel.

El controlador o unidad de control de la máquina-herramienta.

Este componente, generalmente un *microprocesador*, lee e interpreta todas las instrucciones contenidas en el programa de mecanizado. Una vez interpretadas las convierte en señales que envía a las diferentes partes de la máquina involucradas en cada instrucción. Estas señales generan desplazamientos de ejes, apertura y cierre de elementos como la puerta y las pinzas de sujeción, selección de una herramienta, etc. Se generan señales capaces de hacer que la máquina herramienta se ponga a trabajar para realizar el mecanizado de una pieza.

Esta unidad controladora debe incluir la lista de elementos siguientes.

- Unidad de entrada / salida.
- Memoria de almacenamiento de los programas.
- Unidad de cálculo (para poder realizar operaciones en el cálculo de trayectorias).
- Intérprete del código del programa de control numérico.
- Unidad de conexión con el resto de componentes de la máquina (para poder enviarles las señales necesarias para su puesta en funcionamiento).

La máquina-herramienta.

Puede decirse que una máquina-herramienta convencional no se parece en nada a una máquina de control numérico moderna. Se han perfeccionado las cadenas cinemáticas de potencia y posicionamiento, la estructura de las mismas, la incorporación de almacenes de herramientas con su controlador automático de carga y descarga, la concentración de muchas operaciones elementales en una única máquina y, por último, la incorporación de un ordenador como unidad de control.

Mejoras en la flexibilidad de las herramientas.

Como hemos comentado anteriormente, para obtener un mayor grado de flexibilidad en nuestra máquina-herramienta debemos ser capaces de poder almacenar varias herramientas dentro de la máquina. Las dos formas principales para el almacenamiento y acceso a las herramientas son:

- *Las máquinas de cadena*. En este tipo de máquinas las herramientas se guardan en porta-útiles. Este tipo de solución presenta la ventaja de tener una alta capacidad de almacenamiento pero, en contra, presentan el inconveniente de un acceso más lento.
- *Las máquinas de tipo revolver.* En este caso, las herramientas se sitúan sobre cada una de las posiciones de un revolver. Este revolver puede almacenar de 6 a 12 herramientas. En el caso de las máquinas del LOIP, tanto el torno como la fresadora poseen un revolver con 6 herramientas, aunque en el caso del torno, una de esas herramientas es un dispositivo que permite fijar correctamente la pieza entre las garras del cabezal. La ventaja de esta segunda opción consiste en que el cambio de herramienta es rápido. La desventaja se basa en que podemos almacenar menos herramientas que en el primer sistema.

Mejoras relacionadas con la flexibilidad en la sujeción y manipulación de las piezas.

La sujeción y la manipulación de las piezas son dos de los grandes problemas de la fabricación flexible, debido a que limitan en muchos casos el tipo de piezas a mecanizar. Así, en nuestro caso, la FMC del LOIP está preparada para manejar materiales cilíndricos, de un diámetro próximo a los 40 mm, que llevan incorporados en su parte inferior un suplemento cilíndrico desde el que pueden ser agarradas.

- **Sujeción.** La resolución del problema de sujeción de las piezas presenta varias opciones dependiendo del tipo de máquina, pieza a mecanizar, etc. En la fresadora del LOIP disponemos de una mordaza activada por un pistón neumático. La mordaza recoge la pieza mediante dos suplementos de forma semicilíndrica que se adaptan a la forma de la pieza. En el torno del LOIP, la pieza cilíndrica se sujeta gracias a un cabezal; este cabezal posee unos pernos que se encargan de fijar el suplemento de la pieza, de forma que la herramienta del torno puede acceder a todas las partes del material.

- **Manipulación**. En el LOIP disponemos de un robot encargado de recoger las piezas y colocarlas en el centro de trabajo para su posterior mecanización, el robot Mitsubishi. La flexibilidad de este robot se ha visto incrementada con la incorporación de un sistema de desplazamiento lateral (slide) que permite un acceso rápido a los dos centros de mecanizado.

3.2.6. Descripción de la fresadora TRIAC-VMC.

Es un centro de mecanizado vertical de la casa Denford, modelo TRIAC-VMC y nº de serie EO-4290. Sus movimientos básicos se muestran en la figura 3.1. Puede programarse directamente con un lenguaje ISO. La fresadora tiene un equipamiento estándar dotado de una pantalla gráfica a color, un sistema automático de lubricación, 7 estaciones automáticas de cambio de herramientas, y una unidad de disco para cargar programas.

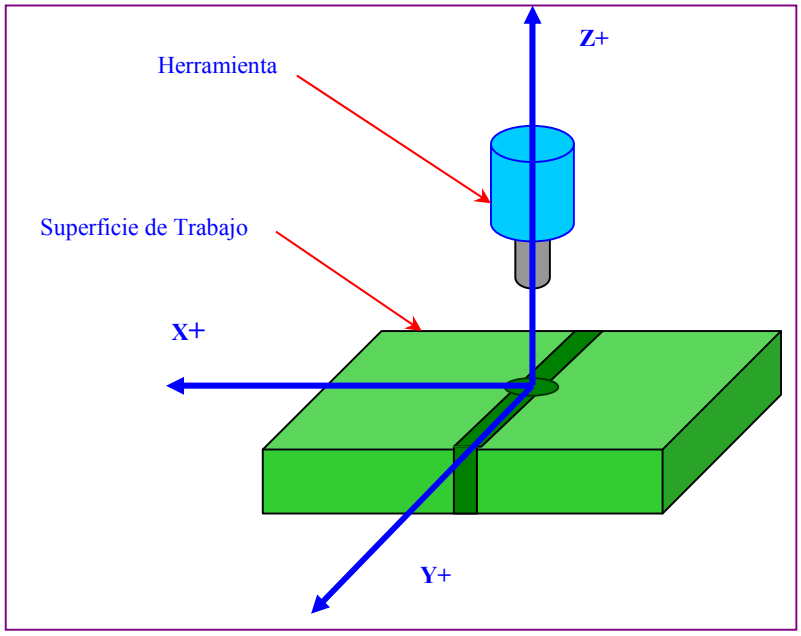


Figura 3.1. *Movimientos básicos de la fresadora TRIAC-VMC.*

Sus características técnicas más relevantes son:

- Máximo desplazamiento longitudinal: 290 mm.
- Máximo desplazamiento transversal: 170 mm.
- Velocidad del eje: 0÷4000 rpm.
- Superficie de trabajo: 500 × 160 mm.
- Resolución: 0.01 mm.
- Peso: 380 Kg.
- Longitud: 1805 mm.
- Anchura: 800 mm.

- Altura: 925 mm.
- Sus características eléctricas son: potencia del motor: 1CV; tensión eléctrica: 220/240 V; y suministro de la red: 50-60 Hz.

En la figura 3.2. se puede apreciar el aspecto de la fresadora Triac VMC:

Figura 3.2. *Imagen de la fresadora TRIAC* [W8].

3.2.7. Descripción del TORNO MIRAC.

El torno ha sido fabricado por la empresa Denford, modelo MIRAC, y posee el número de serie EO-4290. Se programa usando el formato ISO y viene equipado con teclado, con una pantalla gráfica a color y con una unidad de disco para cargar programas.

Sus características técnicas son:

- Máximo diámetro de mecanizado: 160 mm.
- Máxima longitud de mecanización: 180 mm.
- Velocidad del eje: 0÷5000 r.p.m.
- Desplazamiento en eje X: 85 mm.
- Desplazamiento en eje Z: 200 mm.
- Resolución: 0.01 mm.
- Peso: 320 Kg.

- Longitud: 1100 mm.
- Anchura: 1050 mm.
- Altura: 620 mm.
- Sus características eléctricas son: potencia del motor: 1CV; tensión eléctrica: 240 V; y suministro de la red: 50-60 Hz.

La principal diferencia entre el torno MIRAC y la fresadora TRIAC, es que aquél sólo tiene dos ejes, a diferencia de la fresadora que tiene tres. Al ser máquinas de la misma empresa, su manejo es similar. Los movimientos básicos del torno se muestran en la figura 3.3.

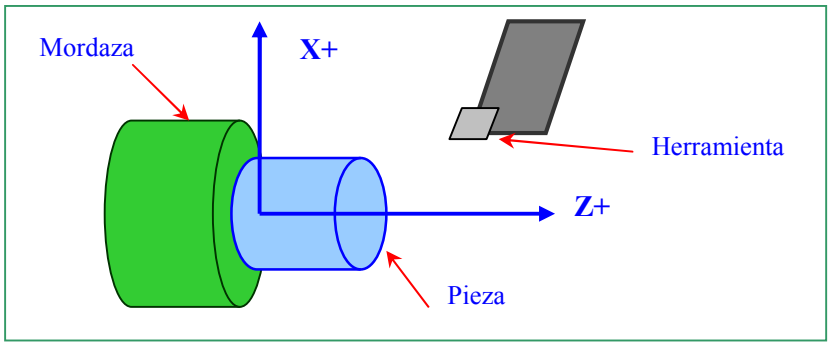


Figura 3.3. *Movimientos básicos del torno MIRAC.*

- **EJE Z**. El eje Z está definido entre el rotor del torno y el extremo del almacén de las herramientas. La dirección se extiende a lo largo del eje de rotación del rotor. El movimiento (-) desplaza la torreta hacia la izquierda de su posición, mientras el movimiento (+) desplaza la torreta hacia la derecha de la posición en la que se encuentre.

- **EJE X**. El eje X está definido por el eje perpendicular al eje Z. El movimiento (-) desplaza la torreta hacia la línea central de rotación del rotor, mientras que el movimiento (+) desplaza la torreta alejándola del eje de rotación del rotor.

La figura 3.4. nos proporciona una vista frontal del torno Mirac.

Figura 3.4. *Imagen del torno MIRAC* [W8].

3.3. SISTEMAS DE ALMACENAMIENTO.

Tal y como vimos en el primer capítulo, uno de los objetivos de la fabricación flexible es la eliminación de stocks, tanto en lo referente a productos terminados como a materiales en curso de fabricación o materiales en bruto, dado que los stocks de productos lo único que conllevan es un incremento en el coste del proceso productivo. Por este motivo, la tendencia es eliminar los grandes almacenamientos de materia prima y de productos acabados para conseguir pequeños almacenamientos de piezas a pie de máquina que garanticen una fabricación automática y desatendida durante largos periodos de tiempo.

3.3.1. Almacén ASRS.

Para conseguir ese objetivo de eliminación de stocks, disponemos de los *Sistemas Automáticos de Almacenamiento y Recogida de Piezas*, también conocidos como *ASRS* (Automated Storage and Retrieval Systems), que son los encargados de llevar el inventario de todas las piezas que se han fabricado y de todas las que están en curso de fabricación.

Esta técnica de manipulación de materiales surgió en los años 50 en Estados Unidos con el fin de reemplazar el manipulado manual de piezas por otro que estuviera controlado por un ordenador.

Normalmente, y como ocurre en el almacén del LOIP, están formados por una matriz de celdas dispuestas verticalmente que se definen por medio de coordenadas cartesianas. Estos valores cartesianos, 'x' e 'y', definen la posición de cada celda respecto a un origen de referencia que puede estar en cualquier punto dentro del campo de acción del manipulador incorporado en el almacén.

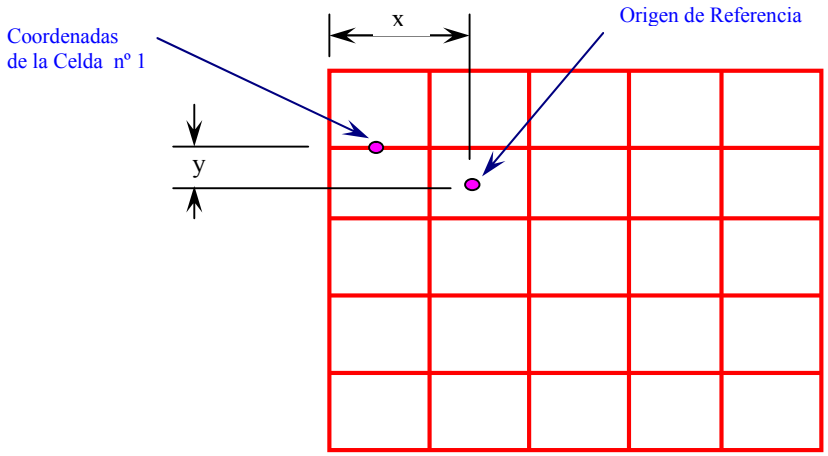


Figura 3.5. *Definición de las coordenadas de las celdas en un almacén automático.*

Las piezas se disponen sobre palés situados uno en cada estantería o celda del almacén. Su sistema de control conoce en todo momento las coordenadas de cada celda, así como la ubicación de cada tipo de pieza y las celdas vacías, por lo que se lleva un control estricto de cada entrada y salida de palés (*gestión de stocks*).

Para manipular los palés, los sistemas ASRS disponen de un brazo robotizado, tal y como se muestra en la figura 3.6. Este brazo es el encargado de poner y quitar palés de una cinta transportadora u otro medio de transporte que sirva para la alimentación de piezas en las máquinas-herramienta. El sistema de control del ASRS, además de gestionar el inventario, va a ser el encargado de gobernar el movimiento del brazo robotizado.

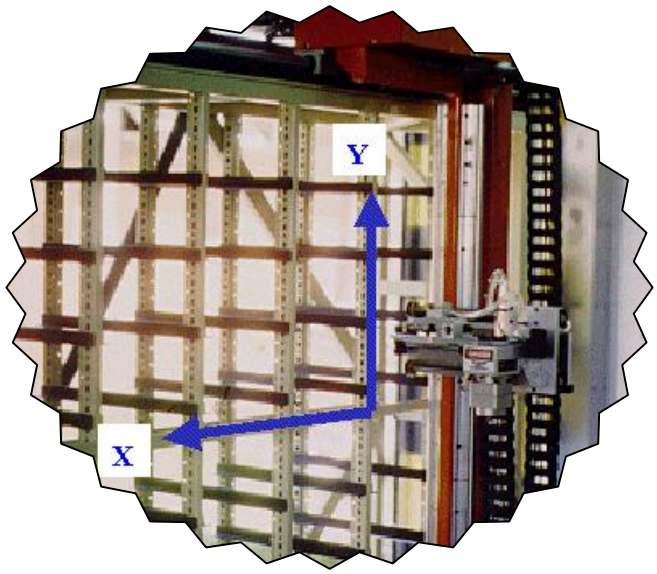


Figura 3.6. *Brazo robotizado de un almacén automático* [W2].

Las principales *ventajas* de los sistemas de almacenamiento y recogida de materiales pueden resumirse en los siguientes puntos:

- Hacen posible un control exhaustivo de todas las piezas que entran y salen de la FMC.
- Podemos conocer el estado del inventario en cualquier momento.
- Ahorro de espacio debido a la disposición matricial de este tipo de almacenes.
- Posibilidad de operar en ambientes perjudiciales, pues no necesitan de una supervisión continua por parte de un operario. Es suficiente con unas revisiones periódicas para reponer el inventario.

- Se puede crear cualquier tipo de inventario definiendo el tipo de pieza que queremos tener en cada celda y esto puede variarse en cada tarea que realicemos.

En definitiva, los sistemas ASRS nos permiten una gestión más rápida y eficiente del inventario.

3.3.2. EL almacén 863-ASRS del LOIP.

El almacén 863-ASRS de la FMC del LOIP es un sistema híbrido neumático/eléctrico diseñado para realizar funciones de inventario en sistemas que utilizan CIM. El almacén 863-ASRS tiene la capacidad de coger palés de las celdas para situarlos en la cinta transportadora y realizar el proceso inverso una vez acabado el mecanizado de las piezas en las diferentes máquinas que forman la célula en donde se encuentra integrada.

El objetivo final que cumple el almacén es proporcionar el material adecuado, en el lugar adecuado y en el momento adecuado.

Componentes del almacén.

El almacén ASRS 863 se compone de:

- 25 celdas metálicas para el almacenamiento de las piezas en sus correspondientes palés.
- Sistema mecánico que ejecuta los movimientos.
- 2 servomotores Allen-Bradley de corriente continua y paso a paso (uno para el eje "x" y otro para el eje "y"), voltaje máximo 100 voltios y velocidad máxima 2500 r.p.m.
- Sistema neumático alimentado por un compresor de aire.
- Sistema de frenado neumático que solamente actúa en movimientos en el eje vertical.
- Guías.

- Unidad de control “Amatrol”.
- Teclado y monitor de ordenador desde donde el usuario puede manejar el almacén.
- 6 sensores, 4 sirven para detectar los fuera de límites superior, inferior, derecha e izquierda y otros 2 que sirven para determinar la posición del *Home* (origen de coordenadas).
- Pulsador de emergencia que interrumpe el fluido eléctrico.
- Centro de transformación y alimentación de los materiales.

Algunos de estos elementos se pueden descubrir en la figura 3.7 que se muestra a continuación.

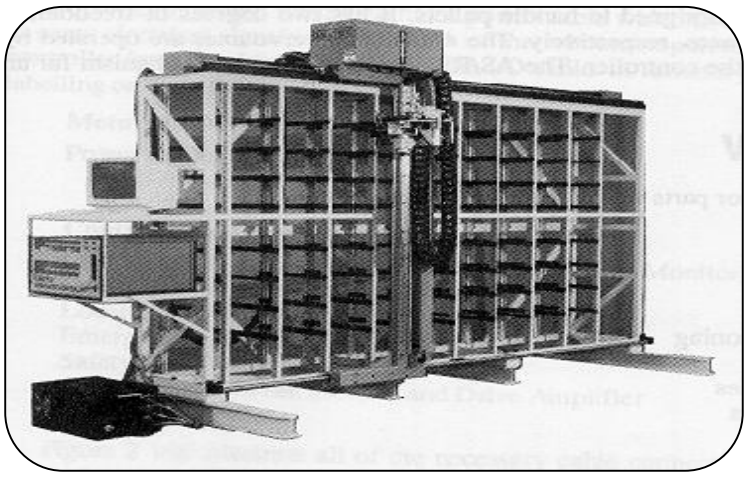


Figura 3.7. *Vista general de un almacén ASRS* [W5].

El ASRS es un sistema cuyos movimientos se describen usando un sistema de coordenadas rectangulares. Dos motores, uno para el eje X y otro para el eje Y, controlan el movimiento del brazo. Cada vez que se arranca el almacén es necesario desplazar el brazo al origen de coordenadas (*Home*). Dicho punto se encuentra determinado por dos sensores situados en la parte superior derecha del almacén.

El brazo realiza movimientos de entrada y salida de la celda, apertura y cierre de la pinza y giro horizontal del mismo (figura 3.8.). Estos movimientos se realizan con el sistema neumático. Es posible controlar la velocidad de estos movimientos con la utilización de unas pequeñas válvulas situadas en la base de los conductos de aire.

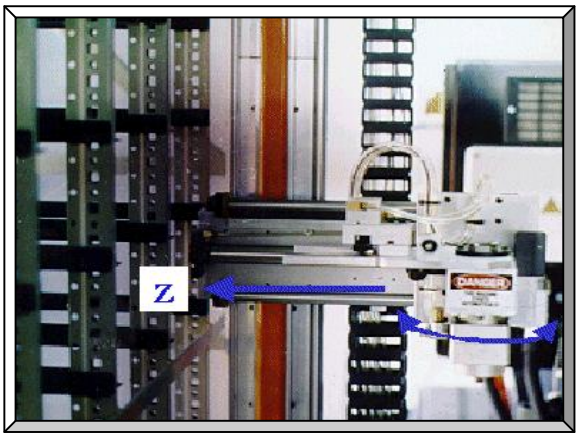


Figura 3.8. *Entrada y salida de la celda, giro hacia dentro y fuera, etc.*[W2]

El sistema de frenado tiene como principal función evitar la caída del brazo cuando los drivers han sido desconectados. Este sistema permite los movimientos en horizontal a pesar de estar conectado, pero no permite los movimientos verticales. Sin embargo no es aconsejable realizar movimientos con el sistema de frenado conectado para que el freno no se desgaste innecesariamente.

3.4. ROBÓTICA INDUSTRIAL.

A modo de introducción, debemos hacer referencia al origen de la palabra Robot. El término Robot fue acuñado por el escritor checoslovaco Karl Capek (1890–1938), que adquirió fama mundial con su obra RUR. En esta obra, Karl Capek presenta al obrero moderno como un esclavo mecánico, y es precisamente allí donde emplea la palabra Robot, tomada del eslavo Robota, que significa trabajo.

El trabajo repetitivo ha constituido la preocupación de todos los grandes hombres de todas las épocas. Aristóteles, el cerebro de más vasta concepción en la historia del pensamiento, refiriéndose a una forma particular de la tarea repetitiva acuño una frase que aún tiene vigencia, *"Cuando los telares tejan por sí mismos, habrá terminado la esclavitud del hombre"*.

Norber Winer, matemático norteamericano, que introdujo el término cibernética y su teoría, refiriéndose al mismo tema, expresó: *"Es una degradación para un ser humano encadenarlo a un remo y usarlo como fuente de energía; pero es casi igual de degradante asignarle tareas puramente repetitivas en una fábrica, que exigen menos de una millonésima de su poder cerebral"*.

La implantación del *Taylorismo* ha traído como consecuencia no sólo condiciones particulares de consumo y cultura, sino también resulta ser el responsable de la creación de condiciones de trabajo repetitivas, monótonas, alienantes y degradantes para quien las efectúa.

A día de hoy, sólo la *robotización del trabajo* o *robótica*, aparece como el medio capaz de superar la irracionalidad introducida por el Taylorismo. Esta irracionalidad, consistente en haber parcializado el trabajo, se manifiesta en el último eslabón del proceso, es decir, en el empleo de un *ser inteligente* en una operación estúpida.

En la actualidad, podemos decir que los robots han llegado a ser una parte integral y necesaria de un importante número de sistemas automatizados en la industria. Se usan en gran variedad de aplicaciones, tales como estaciones de ensamble, estaciones de pintado, estaciones de soldadura, estaciones de atención de máquinas CNC, etc. El reemplazo del hombre en estas actividades evita las tareas fatigosas, rutinarias, peligrosas o excesivas para las capacidades musculares del ser humano.

¿Podemos preguntarnos ahora qué es un robot Industrial?. Se entiende por robot industrial un dispositivo de maniobra destinado a ser utilizado en la industria y dotado de uno o varios brazos, fácilmente programable para cumplir

operaciones diversas con varios grados de libertad y destinado a sustituir la actividad física del hombre en las tareas repetitivas, monótonas, desagradables o peligrosas.

Una de las definiciones más utilizada últimamente es la que da la RIA (Robot Industries Association). Dicha asociación define al robot como *"Un manipulador reprogramable y multifuncional, diseñado para mover materiales, piezas, herramientas o dispositivos especiales, según trayectorias variadas, programadas para la realización de diferentes trabajos".*

Indudablemente, estas definiciones no abarcan todas las posibilidades de aplicación presente y futuras de los robots.

3.4.1. Evolución histórica. Las generaciones de robots.

Existen, a día de hoy, tres generaciones de robots claramente diferenciadas en el tiempo y en características:

a.- Robots de primera generación. Fueron los primeros robots introducidos para uso industrial. A esta generación pertenece la mayor parte de los robots usados actualmente en la industria. Este tipo de robots no disponen de captadores que recojan información sobre el entorno que les rodea, no tienen inteligencia incorporada y sólo están diseñados para accionar una secuencia de operaciones programadas previamente.

b.- Robots de segunda generación. Disponen de las mismas características que los robots de primera generación pero, además, añaden una pequeña capacidad de inteligencia que les permite el reconocimiento de su entorno. Están dotados de un conjunto de captadores capaces de generar señales a partir de las condiciones existentes en el entorno del robot, pudiendo detectar la proximidad de otros objetos que estén en su campo de acción. Estas señales serán analizadas e interpretadas por el microprocesador para así poder tomar decisiones.

c.- Robots de tercera generación. Estos robots están en plena fase de evolución y desarrollo. El objetivo que se pretende conseguir es el de fabricar robots que puedan tomar decisiones por sí mismos, sin necesidad de un factor externo, es decir, se trata de dotar a los robots de inteligencia para que ellos mismos puedan determinar las mejores técnicas de producción en cada momento y tomar decisiones en situaciones determinadas. Ligado a este desarrollo está el campo de la Inteligencia Artificial.

3.4.2. Principales características de los robots.

Son las que vienen definidas en el manual del robot. Para determinar algunas de ellas es preciso definir previamente un punto de referencia en el elemento terminal del robot.

1. ***Envolvente de trabajo.*** Es el conjunto de puntos espaciales a los que puede acceder el elemento terminal del robot. Normalmente se trata de un volumen de forma complicada ya que los ejes de rotación no suelen cubrir los 360^{o} completos. Como es lógico, todos los puntos de carga y descarga de las máquinas deben estar situados dentro de este espacio.
2. ***Capacidad de carga***. Peso máximo de las piezas o herramientas que puede transportar el robot de un punto a otro con el elemento terminal. El dato suministrado suele ser la carga nominal que corresponde a la que puede transportar sin que disminuyan sus prestaciones dinámicas. Es posible aumentar la carga por encima de este limite a costa de disminuir la velocidad y la precisión.
3. ***Grados de libertad***. Cuantos más grados de libertad tenga un robot mayor será la capacidad de éste para realizar desplazamientos y movimientos complejos, pero también mayor será su complejidad y coste. Un robot con seis grados de libertad podrá colocar la pieza situada en su elemento terminal en cualquier punto que esté dentro de su envolvente de trabajo y ponerla en cualquier orientación.

4. ***Repetibilidad***. Existen muchas definiciones de repetibilidad de posición, pero dado que los dos robots de la célula del LOIP utilizan la definición de la Norma Industrial japonesa, definiremos la repetibilidad como *"La medida de la exactitud con la que el robot se reposiciona bajo las mismas condiciones y los mismos métodos"*. La repetibilidad tiene una relación directa con la calidad de los componentes del robot, de la carga que tengamos en el elemento terminal y de la variación de temperatura.

5. ***Precisión***. Distancia entre la posición programada y el valor medio de las posiciones realmente alcanzadas por el elemento terminal, a la temperatura y carga nominales. La falta de precisión es debida a errores en la calibración, deformaciones dinámicas y térmicas, errores en las transformaciones cinemáticas, etc.

6. ***Resolución***. Mínimo incremento que puede aceptar la unidad de control. Su valor vendrá determinado por la resolución de los captadores y convertidores A/D y por el número de bits con que opera la CPU.

7. ***Velocidad***. Está relacionada de forma inversa con la carga transportada. Es un factor muy importante para el cálculo de los tiempos de ciclo, sobre todo en los robots dedicados a manipulación de piezas o ensamblaje. El dato suministrado suele corresponder a la velocidad nominal en régimen permanente. Para alcanzar este régimen, el movimiento debe ser lo suficientemente largo.

8. ***Aceleración***. Para conseguir que sea alta hay que procurar que los brazos sean ligeros y que sus centros de gravedad estén dentro del eje de giro. Los actuadores eléctricos o neumáticos son mucho más rápidos que los hidráulicos. Podemos disponer de distintos perfiles de velocidades: parabólico, trapezoidal, etc.

9. ***Deriva***. Tendencia al desplazamiento del punto medio de las posiciones alcanzadas por el elemento terminal después de muchos ciclos.

10. ***Rebasamiento***. Es el sobrepaso que se produce al intentar alcanzar la posición final. Su origen reside en el fenómeno oscilatorio amortiguado y es rápidamente corregido por los sistemas de control en lazo cerrado.
11. ***Fiabilidad***. Disponibilidad esperada de funcionamiento correcto del robot.
12. ***Vida útil***. Un robot puede desecharse por dos causas: obsolescencia o coste excesivo de las reparaciones. Un robot bien construido puede tener una vida útil de unos diez años, aunque esta cifra dependerá de múltiples factores.

3.4.3. El sistema de control.

Un robot está controlado por un microprocesador que actúa de *cerebro* de todas las operaciones que el robot realiza. Este microprocesador forma parte del controlador del robot que además deberá incluir, por lo menos, los componentes que se señalan a continuación:

- **Memoria.** Es el elemento donde almacenaremos los programas que queremos que el controlador del robot ejecute.
- **Tarjeta de conexiones con las bobinas.** Controla los diferentes motores de cada elemento del robot.
- **Tarjeta de conexiones con el entorno exterior,** tanto para entradas como salidas. Posibilita la transferencia de programas y órdenes desde un ordenador externo, así como la recepción y envío de señales a otras máquinas que formen parte de la FMC.
- **Unidad de control, mando a distancia o mando de programación.** Puede ser utilizada por el operario para realizar determinadas funciones en el robot de una forma manual (por ejemplo, para la definición de puntos a los que el robot debe acceder).

Es posible encontrar robots muy complejos que no disponen de un único microprocesador que controle todos sus movimientos, sino de una serie de microprocesadores con una tarea específica para cada uno de ellos (control de los brazos del robot, control de las señales enviadas y recibidas de los captadores, control del flujo existente entre todos los procesadores, etc.). La transferencia de programas con un ordenador central se realiza a través del puerto serie, normalmente regido por la norma RS-232-C de comunicaciones.

El *sistema de control* del robot se encarga de gobernar y coordinar toda las señales y movimientos de los diferentes elementos del robot con el objetivo de que realice unas tareas determinadas. Dicho sistema de control puede trabajar en lazo abierto o en lazo cerrado:

- El funcionamiento en *lazo abierto* es más sencillo y de menores prestaciones. Consiste en aplicar energía al motor y desconectar tras un periodo de tiempo predefinido. No podemos predecir cual será el error que se produce en la posición final.

- En *lazo cerrado* el sistema de control recibe continuamente información sobre la posición recogida por los *transductores* y aplica a los *accionadores* la energía adecuada para corregir el error de posición. Por tanto existe una realimentación. A este grupo pertenecen la mayoría de los robots industriales.

El control del robot es multivariable (porque hay varias articulaciones que controlar), acoplado (porque el estado de una articulación afecta al estado de las demás), fuertemente no lineal y de parámetros variables.

El sistema de control consta de dos partes fundamentales: control cinemático y control dinámico.

a. Control cinemático.

Se encarga de generar, a partir de la tarea encomendada, las consignas a aplicar a los servomotores para que el robot se desplace desde la posición inicial a la posición deseada. El sistema de *control cinemático* puede descomponerse en las siguientes etapas:

1. Cálculo de una trayectoria cartesiana que cumpla las especificaciones dadas por el usuario.
2. Muestreo de la trayectoria seleccionada. Se selecciona el número de puntos a tomar.
3. Transformación cinemática inversa: transforma los puntos en el espacio cartesiano al espacio articular.
4. Interpolación de los puntos obtenidos en el espacio articular para obtener la trayectoria que debe describir el robot.
5. Selección de un número adecuado de puntos de las trayectorias articulares obtenidas, que sirvan como referencia a los servomecanismos del motor.

El primer paso será plantear los modelos geométrico y cinemático del robot. Estos modelos asumen el equilibrio estático del robot a lo largo del tiempo.

Modelo geométrico.

Relaciona una determinada configuración en el espacio articular con la configuración correspondiente en el espacio cartesiano. Las ecuaciones que ligan ambos sistemas constituyen la *Transformación Homogénea (Directa* para pasar de coordenadas articulares a cartesianas e *Inversa* para pasar de cartesianas a articulares). Evidentemente, estos sistemas de ecuaciones tienen en cuenta la estructura geométrica del robot. La transformación inversa presenta el problema de indeterminaciones en la solución de sus ecuaciones.

Modelo cinemático.

Establece la relación entre las velocidades articulares y cartesianas. Se obtiene al linealizar la *transformación directa* en torno a un punto. Una matriz jacobiana J relaciona los incrementos cartesianos con los articulares.

Para obtener los incrementos articulares a partir de los cartesianos es necesario invertir la matriz Jacobiana. El problema radica en que determinadas configuraciones, denominadas singulares, poseen un determinante de la matriz J nulo, en otras palabras, la matriz jacobiana de estas configuraciones es no inversible. Además, si la matriz J no es cuadrada, porque hay redundancia de grados de libertad, existirán infinidad de soluciones y será necesario incorporar criterios adicionales para determinar una solución única.

Modelo dinámico.

Las elevadas velocidades y aceleraciones presentes en los robots dan lugar a la aparición de diversos fenómenos dinámicos que no son acogidos por los modelos geométrico y cinemático. Será necesario plantear un modelo dinámico que considere las fuerzas de gravedad, de rozamiento, de inercia, de Coriolis, centrífugas, etc. Las ecuaciones que representan estos modelos dinámicos se basan en las leyes de la mecánica clásica newtoniana y existen diversos formalismos para plantearlas. Los más usuales son:

- *Ecuaciones de Lagrange*. Permiten una formulación sencilla y hacen intervenir explícitamente los pares y las fuerzas, pero tienen el inconveniente de exigir un cálculo excesivo.
- *Ecuaciones de Newton-Euler.* La formulación de Newton-Euler es un conjunto de ecuaciones recurrentes, lo que facilita su resolución en un tiempo reducido, pero tiene el inconveniente de ser menos adecuada para la obtención de las leyes de control.
- Ecuaciones de Gibbs-Appel.
- Trabajos virtuales de d'Alambert.

b. Control dinámico.

A partir de las consignas generadas por el control cinemático, el control dinámico gobierna los accionadores para que ejerzan unas determinadas fuerzas o pares.

El sistema de control dinámico puede ser un simple PID o llegar a complejos sistemas de Control Adaptativo.

Control Proporcional, Derivativo e Integral (PID).

Consta de tres parámetros:

1. ***Proporcional***. Es la ganancia del sistema de control. El procesador multiplica el error en cada instante por este parámetro y aplica una tensión proporcional al motor para reducir dicho error. Por tanto cuanto mayor sea este parámetro más rápidamente se reducirá el error de posición. El problema es que no es capaz de anular completamente el error ya que una vez éste se ha hecho pequeño, el controlador no aplica la suficiente potencia al motor para vencer la fricción y, por tanto, queda un pequeño error estacionario.
2. ***Integral***. Este parámetro soluciona el problema anterior. Los errores se van sumando en el tiempo y se multiplican por este parámetro. Su efecto es más lento que el del anterior y menos apreciable durante el movimiento, sin embargo cuando el error se ha hecho pequeño toma el relevo al parámetro proporcional y es capaz de cancelar el error completamente. El uso de un valor excesivo puede causar sobrepasos.
3. ***Derivativo***. Es el responsable de reducir el error de velocidad a lo largo de la trayectoria. Un valor adecuado logrará un movimiento limpio y suave a lo largo de la trayectoria, con aceleraciones y deceleraciones perfectas. Un valor elevado provocará vibraciones.

El robot Scorbot ERVII de la célula del LOIP usa un control de este tipo y los tres parámetros pueden ser fijados por el usuario.

Control adaptativo.

La complejidad y naturaleza no lineal del comportamiento dinámico de la estructura mecánica articulada de los robots, unido a la variabilidad de las condiciones de trabajo, ha llevado al planteamiento de estrategias de control que eviten la dependencia directa del modelo dinámico permitiendo, sin embargo, un comportamiento uniforme en una amplia gama de situaciones operativas del sistema.

El *control adaptativo* ha demostrado su gran potencialidad en muy diversos campo de la automática y la aplicación de microprocesadores facilita su uso. En el campo de los robots los resultados son muy prometedores y demuestran sus excelentes características de funcionamiento.

Los algoritmos de control adaptativo pueden ser planteados bajo dos enfoques distintos: *modelo de referencia* y *ajuste de parámetros*, que conducen a resultados equivalentes en muchos casos.

Control adaptativo por modelo de referencia.

Permite obviar el uso del modelo dinámico del robot para el diseño del sistema de control. En su lugar se define un sistema desacoplado simple que permite establecer las características de comportamiento deseadas del sistema. La consigna se aplica simultáneamente al sistema y al modelo de referencia. La diferencia de respuesta entre ambos, denominada *error generalizado,* es usada por el mecanismo de adaptación para ajustar las ganancias del sistema a fin de anular ese error y lograr que el sistema se comporte igual que el modelo deseado.

La elección del modelo de referencia esta ligada a las especificaciones deseadas, a la complejidad de cálculo admisible y a las limitaciones mecánicas y energéticas del sistema. Es aconsejable adoptar un modelo de referencia lineal de segundo orden invariante en el tiempo y desacoplado para cada grado de libertad. De esta forma el esfuerzo computacional resulta pequeño.

Control adaptativo autoajustable.

Realiza cíclicamente en cada periodo de muestreo una identificación de los parámetros del sistema, la síntesis de una ley de control y la acción del controlador. Los parámetros del modelo y los coeficientes del controlador alcanzan valores estacionarios con el tiempo.

En consecuencia el sistema consta de tres elementos fundamentales:

1. *Estimador recursivo.* Identifica los parámetros de un modelo, previamente propuesto para el sistema, a partir de sus entradas y salidas.
2. *Algoritmo adaptativo.* Determina los coeficientes de la ley de control, en forma de ecuación en diferencias, a partir de los parámetros suministrados por el estimador recursivo.
3. *Controlador.* Utilizando la ley obtenida y la señal de consigna generada por el control cinemático se elabora la acción de control y la salida real del sistema.

3.4.4. Programación del controlador del robot.

Podemos distinguir dos componentes a la hora de hablar del software que controla el robot.

a. **Programa de usuario**, formado por una serie de instrucciones escritas en un lenguaje de programación de alto nivel. Este programa estará contenido en la memoria del controlador del robot y será el que se ejecute cuando encendamos e inicialicemos dicho controlador. En él estarán programadas todas las tareas que queramos que realice el robot. En la memoria del controlador también estarán las coordenadas de todos los puntos a los que queremos que llegue el elemento terminal del robot, etc. Esta serie de puntos deberá definirse antes de ejecutar el programa del robot.

b. Software de mando, que no es otra cosa que el sistema operativo capaz de realizar la traducción de las instrucciones que forman el programa de usuario en instrucciones *código máquina*, que son las que realmente entiende el microprocesador del controlador. Este componente del software viene prefijado por el fabricante y no puede ser alterado por el usuario del robot.

3.4.5. Robot MITSUBISHI.

El robot Mitsubishi MoveMasterEX RV-M1 del LOIP, es un brazo articulado que posee 5 grados de libertad (6 si le añadimos el que le proporciona el eje de deslizamiento o slide). Tal y como se muestra en la figura 3.9., dispone de un controlador y de una consola de programación portátil (Teaching Box). Dicho controlador puede ser gobernado por control remoto mediante la conexión a un ordenador exterior (HOST) a través del puerto serie RS-232C.

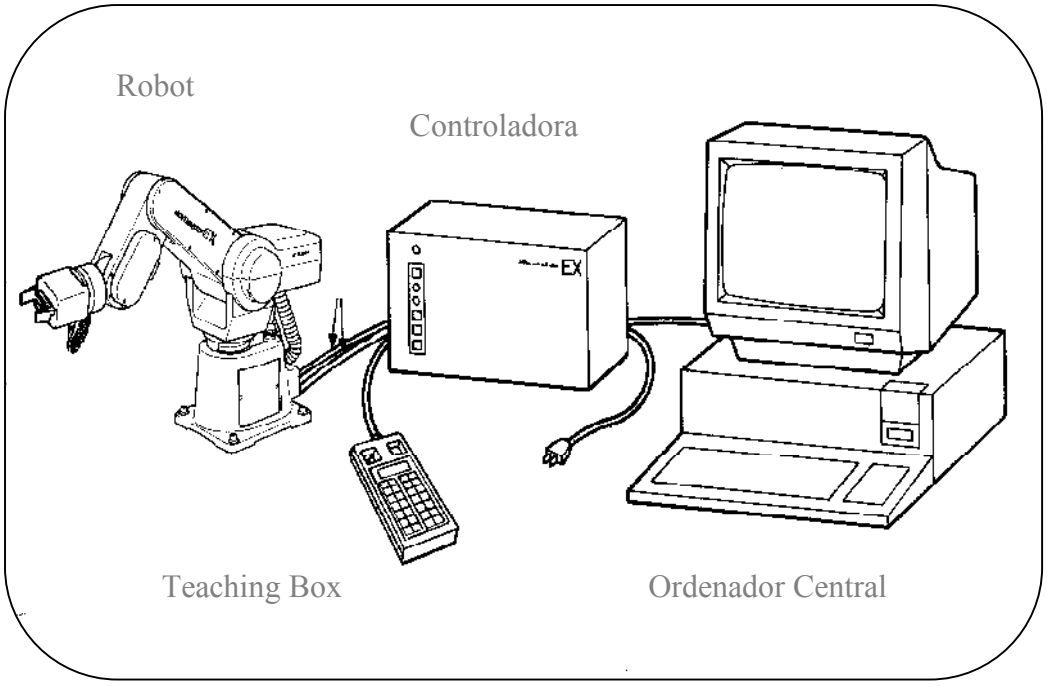


Figura 3.9. *Conjunto de elementos del robot Mitsubishi RV-M1* [M4].

A continuación se describirán brevemente algunos de los aspectos más relevantes del robot Mitsubishi.

a. El robot.

Posee 5 grados de libertad (J1,J2,J3,J4,J5), tal y como se muestra en la figura 3.10.:

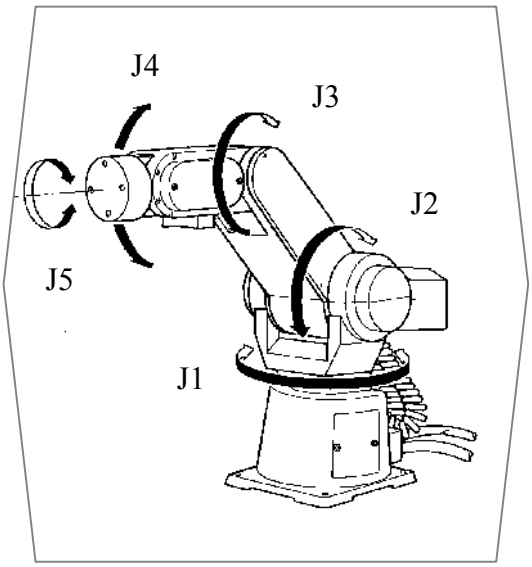


Figura 3.10. *Grados de libertad del robot [M4].*

- El g.d.l. J1 está relacionado con el giro de la base o cintura (*Waist*).
- J2 está asociado al giro del antebrazo (*Upper Arm*) sobre el hombro (*Shoulder*).

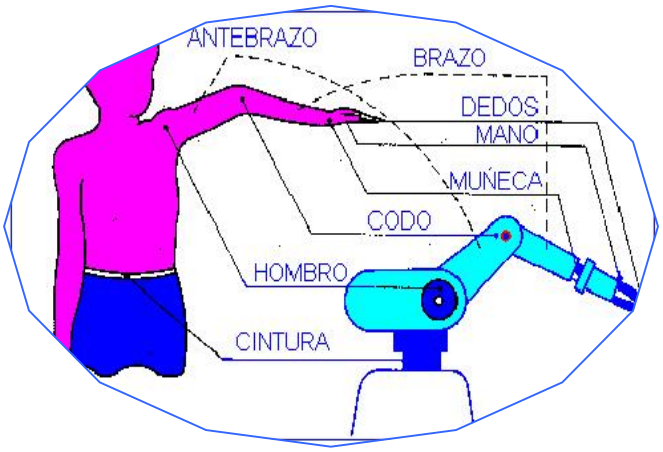


Figura 3.11. *Nomenclatura externa del robot* [W2].

- J3. Giro del brazo (Fore Arm) sobre el codo (Elbow).
- J4. Inclinación de la muñeca (Wrist Pitch).
- J5. Rotación de la muñeca (Wrist Roll).

El robot Mitsubishi nos permite realizar operaciones tanto en el sistema de ejes articulado como en el sistema de coordenadas cartesianas. Esta circunstancia se ve reflejada en la figura 3.12.

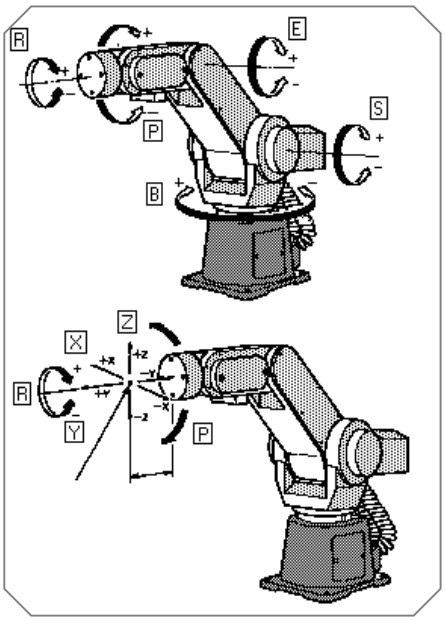


Figura 3.12. *Movimiento articulado y cartesiano* [M4].

El robot MoveMasterEX RV-M1 siempre debe ir a orígenes (*Home*) tras de ser encendido. Si durante la operación de envío a orígenes existiese la posibilidad de colisión con alguno de los equipos anexos al robot, tendremos la precaución de mover manualmente los brazos articulados hasta una posición en la que la probabilidad de colisión durante dicha operación sea nula.

Como consecuencia, conviene conocer los movimientos que realiza el robot para situarse en orígenes. Dicho procedimiento se compone de dos etapas, cada una de las cuales se descompone a su vez en varios pasos:

i. **Etapa 1. Desplazamientos**.

- Paso 1: el hombro (J2) se desplaza en el sentido positivo.
- Paso 2: el codo (J3) se desplaza en el sentido positivo.
- Paso 3: la muñeca (J4) se desplaza en el sentido negativo.

ii. **Etapa 2. Rotaciones**.

- Paso 1: la cintura (J1) rota en el sentido positivo.
- Paso 2: la muñeca (J5) rota en el sentido positivo.

El robot Mitsubishi de la FMC del LOIP dispone, además, de una mano motorizada que nos proporciona el poder de sujetar los objetos. En la figura 3.13. aparece representada dicha mano.

Figura 3.13. *Mano motorizada* [W7].

b. La controladora.

En la parte frontal de la *Unidad de Control* (véase figura 3.14.) se pueden observar varios interruptores y LEDs.

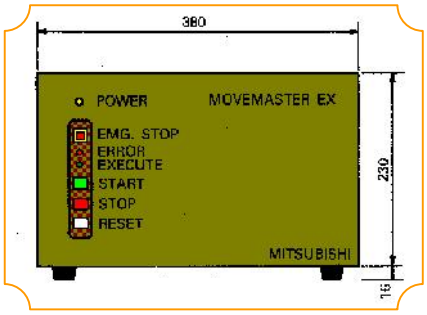


Figura 3.14. *Imagen frontal de la unidad de control del robot* [M4].

Power. Este LED nos indica si el controlador está encendido, en cuyo caso emite una luz amarilla, o apagado. Si hemos encendido el controlador y este LED no emite una luz amarilla, es muy probable que no llegue corriente al controlador (el fusible de la parte posterior se ha fundido, las conexiones eléctricas se han estropeado, etc.).

Emg. Stop. Este interruptor permite detener inmediatamente el movimiento del robot. Si presionamos este interruptor, el LED situado justamente por debajo de él (LED de Error) comenzará a parpadear.

Error. Este diodo se ilumina (en rojo) cuando existe un error. Si se produce un *Error de Tipo I*, esta luz parpadea en intervalos de medio segundo, mientras que si el *Error* es de *Tipo II* la luz brilla constantemente sin parpadear. Sin embargo, esta no es la única forma que tiene el controlador de avisarnos de la existencia de un error: también emite un sonido cada vez que se enciende este LED, es decir, para el error de tipo I emite un sonido intermitente, mientras que para el error de tipo II este sonido es continuo.

Execute. Este LED emite un color verde continuo mientras un comando está siendo ejecutado y se apaga cuando dicho comando ha sido completado. También luce mientras se está ejecutando el programa interno del controlador.

Start. Este interruptor, de color verde, comienza la ejecución del programa o lo reactiva si estaba suspendido.

Stop. Este interruptor, de color rojo, es el encargado de interrumpir la ejecución del programa que esté activo en ese momento. Cuando presiono este botón, el robot completa la ejecución de la línea de comando actual antes de parar.

Reset. Este botón, de color blanco, permite, entre otras cosas:

- Eliminar el error de tipo II.
- Restablecer el sistema tras una parada de emergencia.
- Reinicializar el programa suspendido (bien por que se produjo un error de tipo II o bien porque se presionó el interruptor de Parada de Emergencia). Tras presionar este botón, el controlador se sitúa en la primera línea del programa a la espera de que se presione el botón START.

Por lo tanto, cuando se presiona este botón, el programa regresa al comienzo y, si ha ocurrido algún error, el LED indicador de error se apaga.

Al explicar alguno de los interruptores y LEDs anteriores, hemos hablado del error de tipo I y del error de tipo II, pero, ¿qué son?. El error de tipo II tiene que ver principalmente con los *errores de software*. Las principales causas de su origen son:

- *El comando transmitido por el ordenador personal es erróneo*. Más concretamente, podemos haber enviado un comando indefinido, el formato de transmisión es erróneo o bien se ha producido un error durante la transmisión.
- *El comando no puede ser ejecutado*. Generalmente se debe a que los parámetros exceden un rango u ordenamos mover el robot a una posición no definida. Aquí se engloba el error más común que suele darse, y que consiste en intentar sobrepasar alguno de los límites del robot. Por ejemplo, imaginemos que la pinza del robot ha rotado hasta el ángulo 175^{o} siendo el mayor ángulo permitido el de 180^{o}. Si enviamos una orden al robot para que rote la pinza otros 10 grados, se producirá un error de tipo II indicándonos que dicho movimiento no es posible.

Tal y como se ha especificado anteriormente, para solucionar el error de tipo II vasta con presionar el botón RESET de la parte frontal del controlador, o como veremos más adelante, presionar el botón RESET del módulo de programa encargado del control remoto del robot Mitsubishi.

Por su parte, el error de tipo I tiene que ver principalmente con los *errores de hardware*. Aunque son varias las razones que originan un error de este tipo, a lo largo del proyecto han tenido lugar dos clases de incidentes que han generado este tipo de error. En concreto, estaríamos hablando de:

- Incidente 1. El robot choca con un equipo externo (torno, fresadora, cinta transportadora,...).

- Incidente 2. El cable que transmite la corriente a la mano motorizada impide su desplazamiento. Esta contingencia podría englobarse dentro del incidente anterior, dado que en definitiva se produce una colisión de una de las partes del robot con otro elemento que impide su desplazamiento. Sin embargo, esta es, sin duda alguna, la contingencia que más veces ha tenido lugar, por lo que resulta obligatorio definirla de forma aislada.

Para eliminar este tipo de error, la única solución consiste en apagar el controlador, eliminándose así la posible causa del error acontecido. De nuevo, al reiniciar el robot deberemos comprobar que el desplazamiento a orígenes se efectúa sin peligro de colisión.

Continuando con la descripción de la controladora, en el panel de costado encontramos los interruptores que se muestran en la figura 3.15.

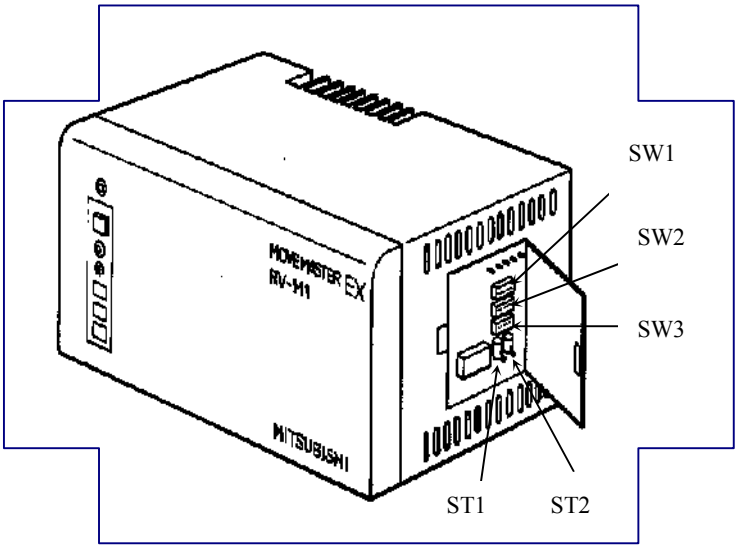


Figura 3.15. *Interruptores laterales* [M4].

A continuación se esbozarán brevemente los aspectos más relevantes de dichos interruptores:

Interruptor ST1. Establece el modo de control. Si se encuentra en la posición superior, es el microprocesador del propio controlador quien posee el control, mientras que si se encuentra en la posición inferior será la computadora personal la encargada de dicho control.

Interruptor ST2. Este interruptor nos indica si los datos de la EPROM son transferidos a la memoria RAM de la unidad controladora (posición superior) o si no son transferidos (posición inferior). En nuestro caso, al no disponer de memoria EPROM, este interruptor tendrá que estar siempre hacia abajo.

Consecuentemente, si queremos controlar el robot mediante el ordenador central, deberemos situar tanto el interruptor ST1 como el interruptor ST2 en su posición inferior. Por el contrario, si nuestra intención es ejecutar el programa del robot, deberemos poner en ON (posición superior) el interruptor ST1.

Situados justo por encima de los interruptores ST1 y ST2, encontramos otros tres interruptores, cada uno de los cuales está formado por 8 bits o patillas:

Interruptor SW1. Este interruptor nos permite, entre otras cosas, seleccionar el tipo de tarjeta de entrada/salida, establecer si se emitirá sonido cuando se produzca un error, seleccionar la terminación de los datos transmitidos a través del puerto serie, etc.

Interruptor SW2. Este es el interruptor encargado de establecer la tasa de baudios que va ha utilizar el controlador.

Interruptor SW3. Determina el formato para la transmisión de datos asíncrona. Nos permite determinar, entre otras cosas, el número de bits de parada o el número de bits de datos que usaremos.

En la figura 3.16. se muestra cómo se han establecido el estado de cada uno de las patillas de los interruptores anteriores.

SW1							
1	2	3	4	5	6	7	8
▲	▲	▲					▲
			▼	▼	▼	▼	
SW2							
1	2	3	4	5	6	7	8
	▲		▲	▲	▲	▲	
▼		▼					▼
SW3							
1	2	3	4	5	6	7	8
						▲	
▼	▼	▼	▼	▼	▼		▼

Figura 3.16. *Estado de las patillas de los interruptores SW1,SW2 y SW3.*

Con esta configuración, los parámetros del robot son los siguientes:

- Terminación CR+LF (‘\r’+’\n’).
- 4800 baudios.
- 7 bits de datos.
- Paridad par.
- 1 bit de parada.

Para finalizar con la unidad controladora, en la parte posterior encontramos, entre otros elementos, las conexiones para la comunicación en serie (RS-232C) y en paralelo (CENTRONICS), la conexión para el teaching box, el interruptor de encendido, el fusible, etc.

c. El teaching box.

La figura 3.17. nos permite conocer el aspecto del teaching box. En su parte superior encontramos un botón de parada de emergencia y el interruptor ON/OFF. Este interruptor es el que nos permite, o impide, utilizar las teclas de la consola de programación portatil. Cuando el robot deba ser manejado manualmente deberemos poner este interruptor en ON. Mientras el programa del robot esté ejecutándose o cuando el control del robot se lleve a cabo mediante el ordenador central, este interruptor deberá situarse en OFF.

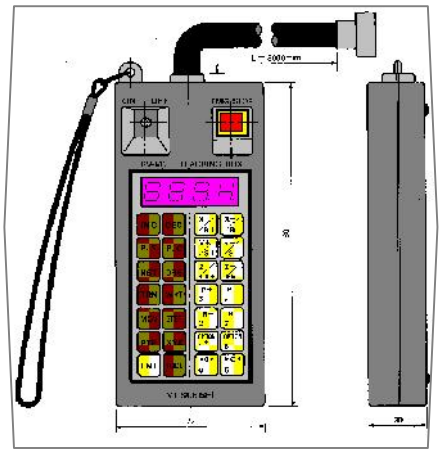


Figura 3.17. *Imagen del teaching box* [W7].

El teaching box se compone, además, de las diferentes teclas mostradas en la figura 3.17., y de una pantalla en la que se muestra el número de la posición, el número de la línea del programa y el indicador del estado del teaching box.

La figura 3.18. nos presenta al robot Mitsubishi suministrando material a la fresadora.

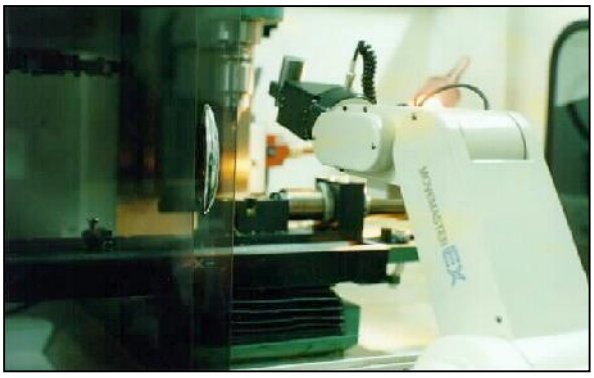


Figura 3.18. *Imagen del Robot Mitsubishi abasteciendo a la Fresadora* [W4].

3.4.6. Robot SCORBOT.

Al igual que el robot Mitsubishi, el robot Scorbot-ER VII del LOIP, es un brazo articulado que posee 5 grados de libertad. Dispone de un controlador y de un mando de enseñanza (Teach Pendant). Dicho controlador puede ser gobernado por control remoto mediante la conexión a un ordenador exterior (HOST) a través del puerto serie RS-232C.

a. El robot.

El robot Scorbot posee 5 grados de libertad tal y como se muestra en la figura 3.19. adjunta:

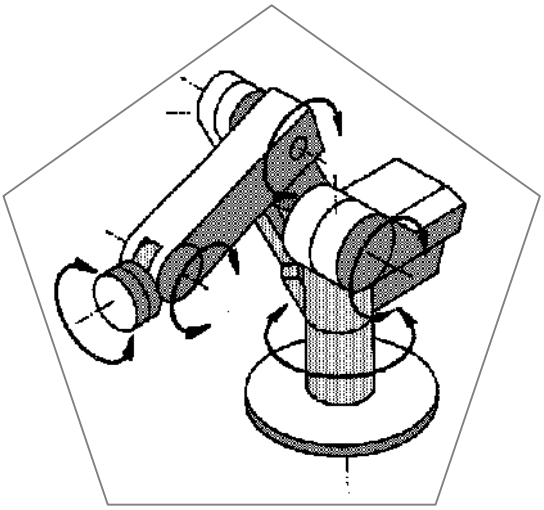


Figura 3.19. *Grados de libertad del robot Scorbot* [M6].

El robot Scorbot posee algunas características similares a las del robot Mitsubishi:

- Nos permite realizar operaciones tanto en el sistema de ejes articulado como en el sistema de coordenadas cartesianas.
- El robot Scorbot-ER VII siempre debe ir a orígenes (*Home*) después de ser encendido. Si en ese desplazamiento a orígenes existiese la posibilidad de colisión con alguno de los equipos anexos al robot, antes de llevar a cabo el *Home* deberemos mover manualmente los brazos articulados hasta una posición en la que la probabilidad de colisión sea nula.

3 El robot Scorbot de la FMC del LOIP dispone, además, de una pinza eléctrica como manipulador final, lo que nos facilita el proceso de sujeción de las piezas.

b. La controladora.

La figura 3.20. representa el aspecto exterior de la *unidad de control*.

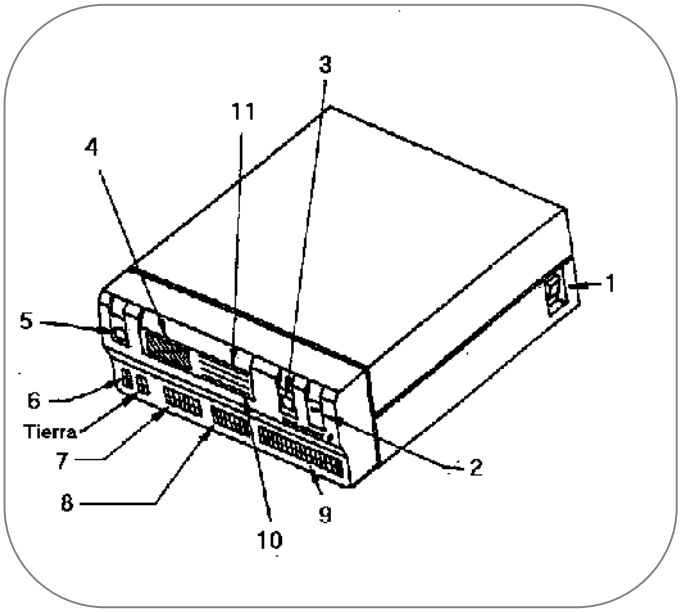


Figura 3.20. *Vista exterior de la unidad de control del robot Scorbot* [M6].

1. **Interruptor general de CONEXIÓN/DESCONEXIÓN de corriente**. Se encarga de conectar/desconectar la corriente al controlador.
2. **Indicador de diodo de corriente**. Este LED nos indica si el controlador está encendido, en cuyo caso emite una luz verde, o apagado. Si hemos encendido el controlador y este LED no emite una luz verde, es muy probable que no llegue corriente al controlador.
3. **Interruptor de los motores**. Conecta o desconecta la alimentación de corriente y la alimentación de corriente del usuario. Conectado se indica por un diodo verde.

4. **Ventilador**. Proporciona circulación de aire dentro del controlador.

5. **Interruptor de emergencia.** Incluye un indicador de luz roja. Cuando el interruptor está oprimido, las alimentaciones de corriente de los motores y del usuario están desconectadas y la CPU del controlador está en *Estado de Parada*. Al liberarlo, se reactiva la CPU y el sistema vuelve a arrancar.

6. **Patillas de salida (relés)**. Son 4 salidas, concretamente de la 1 a la 4, que incluyen relés en su parte final. Cada relé comprende lo siguiente: patilla común (COM), patilla Normalmente Cerrada (NC), patilla Normalmente Abierta (NO).

7. **Patillas de salida (colector abierto)**. Las salidas 5 a 16 incluyen un transistor con un colector abierto en su parte final. Estas salidas nunca deberán conectarse a una alimentación de corriente.

8. **Patillas de entrada**. Existen 16 entradas, 1 a 16, más cuatro puntos de conexión a tierra. Cuando no está conectada ninguna tensión o cuando está conectada la tensión alta, la entrada correspondiente está en el estado INACTIVO. Cuando está conectada tensión baja a la patilla, la entrada está en el estado ACTIVO.

9. **Indicadores de diodo de salida**. 16 diodos verdes lucen cuando la salida está ACTIVA.

10. **Indicadores de diodo de entrada**. 16 diodos amarillos lucen cuando la entrada está ACTIVA.

En la parte posterior del controlador hallamos, entre otras cosas, la entrada de corriente principal, el ventilador, el conector del mando de enseñanza, el conector RS-232C, etc.

Internamente, el controlador del SCORBOT-ER VII está formado por 4 tarjetas de accionamiento.

c. El mando colgante de enseñanza.

La imagen de la figura 3.21. muestra el aspecto del teach pendant o mando colgante de enseñanza.

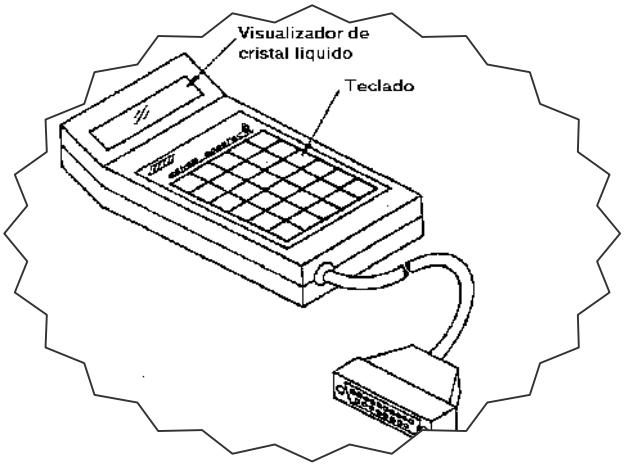


Figura 3.21. *Teach Pendant* [M6].

El mando colgante de enseñanza está dividido en dos zonas; el visualizador y el teclado.

a) **El visualizador**. Muestra el estado actual del controlador y los mensajes del sistema. Hay dos mensajes residentes para el usuario:

   - JUNTAS/XYZ: notifica al usuario el sistema de coordenadas actualmente activas. *Juntas* significa Sistema de Coordenadas de las Juntas del Robot, *XYZ* hace referencia al Sistema de Coordenadas Cartesianas.
   - A/B/EJE: Notifica al usuario el grupo de ejes actualmente activo.

b) En la parte inferior del mando colgante de enseñanza encontramos el **teclado**. Dicho teclado está formada por 30 teclas, la mayoría de las cuales son multifuncionales.

d. El sistema de control del Scorbot-ER VII.

A continuación se van a presentar algunas de las características peculiares del sistema de control del Scorbot-ER VII.

i. El controlador Scorbot-ER VII gobierna tanto la trayectoria (posición) como la velocidad. Además, nos permite elegir el perfil de control: trapezoidal o paraboloide (que es el tipo establecido por omisión).

ii. El diagrama de bloques del sistema de control se muestra en la figura 3.22.

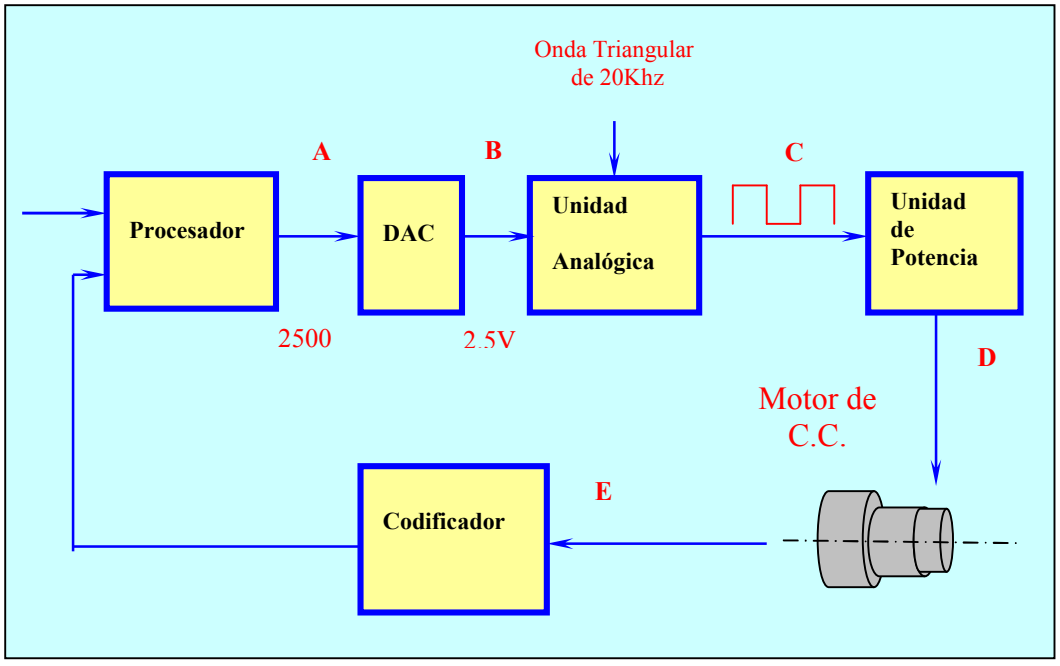


Figura 3.22. *Sistema de control del SCORBOT-ER VII* [M6].

√ **Etapa A**. El procesador calcula la posición y velocidad requeridas cada 10 ms. Emite un valor (digital) en el intervalo de ±5000 milivoltios (todo el intervalo es lineal).

- √ **Etapa B**. La unidad DAC (convertidor digital a analógico) emite una tensión analógica proporcional a la entrada digital. La salida analógica del DAC es -2,5 ≤V ≤ +2,5.
- √ **Etapa C**. La unidad analógica crea una señal continua a 20 Khz. El ciclo de trabajo varía proporcionalmente como sigue:
  - 100% = toda velocidad- en sentido horario (CW);
  - 50% = paro;
  - 0% = toda velocidad- en sentido antihorario (CCW);
- √ **Etapa D**. La unidad de potencia impulsa al motor proporcionándole ±24V c.c., a 20Khz, de acuerdo con el ciclo de trabajo producido por la unidad analógica. El motor no puede reaccionar a esta alta frecuencia de conmutación y es afectado por el valor medio del flujo de corriente.

iii. El proceso completo se conoce como *Control PWM* (Modulación de Anchura de Impulsos).

iv. Durante el giro del motor, el codificador fijado a él produce impulsos proporcionales a la cantidad de giro.

v. Cada 10 ms el procesador lee el codificador y calcula la posición y la velocidad del motor. La velocidad se calcula como la primera derivada de la trayectoria, es decir, como dx/dt. El procesador compara entonces los valores reales de posición, x, y velocidad, v, con los deseados y determina los valores de error, ejecutando la acción necesaria para cancelarlos.

vi. El ciclo de control completo dura 10 ms.

e. Los parámetros de control.

Los parámetros de control PID (proporcional, integral, diferencial) permiten al controlador adaptarse a diversas condiciones de funcionamiento, tales como el dominio de funciones no lineales en el sistema. El usuario puede definir estos parámetros a través del software FMC desarrollado en este proyecto.

Estos parámetros de control PID fueron analizados anteriormente en este mismo capítulo, por lo que a continuación sólo se esbozarán los aspectos más relevantes de cada uno de ellos con relación al robot Scorbot.

- El **parámetro proporcional** es la ganancia del sistema de control. Permite reacciones rápidas y potentes del brazo a órdenes de movimiento. No puede eliminar errores de estado estacionario de pequeño valor. Es el responsable de la repetibilidad del movimiento.
- El **parámetro integral** es la conexión del sistema de control con la historia del control a lo largo de la trayectoria. Ayuda al parámetro proporcional para eliminar errores de estado estacionario. Afecta al tiempo necesario para estabilizar en una posición dentro de la repetibilidad requerida.
- El **parámetro absoluto de desviación**. Permite al sistema vencer la no linealidad (debida a la fricción, gravedad,...) cuando el sistema trabaja con baja potencia de impulsión. La Desviación también es un nivel umbral del DAC.
- El **parámetro diferencial** (o **derivativo**) permite al brazo realizar curvas de movimiento limpias y suaves, con aceleraciones y desaceleraciones perfectas.

3.5. AUTÓMATAS PROGRAMABLES.

3.5.1. ¿Qué es un autómata programable?.

Según la asociación NEMA (National Electrical Manufactures Association) un autómata programable es un aparato electrónico digital que utiliza una memoria interna programable para almacenar las instrucciones que, mediante funciones lógicas, secuenciales, de temporización, de contador y aritméticas y a través de módulos de entrada y salida de señales digitales y analógicas, controla funciones especificas de varios tipos de máquinas o de procesos.

El autómata programable industrial es un equipo electrónico, programable en lenguaje no informático, diseñado para controlar, en tiempo real y en ambiente industrial, procesos secuenciales.

En realidad, un autómata programable, es un microordenador especializado en una tarea determinada. Poseen un procesador central, una memoria, unidades de entrada y de salida y una fuente de alimentación. Algunos más avanzados disponen además de consola de programación e impresora. Los autómatas programables actuales llevan un sistema de comunicaciones a través del que pueden ser conectados a un ordenador central.

Etapas en la evolución de los autómatas.

1. Los primeros equipos datan de finales de los años 60. Empleaban una memoria a base de núcleos de ferrita y un procesador cableado que estaba formado por circuitos integrados. Su aplicación consistía en el control de máquinas y procesos secuenciales (cadenas de transporte, distribución y almacenamiento de material).

2. A mediados de los años 70 se incorporó la tecnología del microprocesador. Este avance permitió aumentar las prestaciones de los autómatas: operaciones aritméticas más complejas, comunicación con un ordenador, etc.

3. A finales de los años 70 se mejoraron las prestaciones y el desarrollo de equipos especializados: se incrementó la capacidad de la memoria, se posibilitó las entradas y salidas remotas, se disponía de E/S analógicas y digitales, se mejoraron los lenguajes de programación con instrucciones más potentes y se desarrollaron las comunicaciones entre periféricos y ordenador. Ahora las aplicaciones son más variadas, ya que al admitir lazos de regulación se puede trabajar con equipos de instrumentación. Más tarde también se desarrolló el control adaptativo.

4. La década de los 80 ha representado un gran avance en todos los aspectos ligados a la tecnología del microprocesador que formaba la unidad central de proceso del autómata. En este sentido se ha conseguido alta velocidad de respuesta en el ciclo de ejecución (cada ciclo máquina dura unos pocos milisegundos), se han reducido las dimensiones de las cajas que contentan el autómata, se han desarrollado entradas y salidas inteligentes (servocontroladores, controladores PID, etc.), se ha aumentado la cantidad de datos que pueden almacenarse en la memoria, así como la capacidad de diagnósticos del funcionamiento (para detectar errores) y, por último, se han desarrollado nuevos lenguajes de programación basados en un entorno gráfico que admite bloques funcionales, lenguajes de diagramas de contactos y diagrama de fases (GRAFCET) y también, más recientemente, lenguajes de alto nivel como el BASIC, FORTRAN y CC++.

Ventajas e inconvenientes.

Las *ventajas* de un autómata programable pueden resumirse en los siguientes puntos:

- Supone un menor tiempo para la elaboración de proyectos.
- Se pueden introducir modificaciones en el programa sin necesidad de cambiar el cableado ni de añadir aparatos nuevos.
- Es una máquina que ocupa poco espacio (sobre todo los autómatas más modernos).
- Supone un bajo coste de mano de obra de instalación (se puede adquirir un autómata a partir de unas 100.000 ptas).
- Son máquinas que necesitan de muy poco o nada mantenimiento, por lo que salen rentables a largo plazo.
- Con un único autómata tenemos la posibilidad de gobernar varias máquinas.
- El autómata sigue siendo útil aunque la máquina que gobierna quede fuera de servicio.

Como *inconvenientes* tendremos los siguientes:

- Será necesario disponer de personal cualificado que conozca con detalle el lenguaje de programación de los autómatas (diagramas de contactos, lenguajes simbólicos, etc.).
- Su coste de adquisición será un problema en ciertas situaciones, pues si lo que queremos es un autómata muy potente que pueda controlar muchos procesos complicados su precio puede incluso superar los 4 millones de pesetas.

3.5.2. Principios de funcionamiento de los autómatas programables.

Los primeros sistemas fabricados se hacían con tecnología cableada. En estos, las ecuaciones de control (circuitos lógicos), se realizaban mediante uniones físicas de relés, de resistencias, de circuitos integrados, etc. Debido a esta forma de generar los circuitos que representaban la ecuación de control, la información se leía simultáneamente, es decir, se realizaba en paralelo. Todos los estados de las variables se coordinaban entre sí y con las variables internas para dar unas salidas, y todo esto al mismo tiempo para todas las variables.

Por el contrario, en los autómatas programables actuales hay incorporado un *microprocesador* que admite una programación, lo que nos permite interpretar una serie de códigos o instrucciones que se combinan entre sí para conseguir las ecuaciones de control deseadas. El microprocesador sólo puede interpretar y ejecutar una instrucción en cada momento, pero lo hace a gran velocidad (unos pocos microsegundos), por lo que el recorrido completo por el programa contenido en memoria tarda muy poco tiempo y es prácticamente como si se ejecutaran todas las instrucciones paralelamente. A este tipo de ejecución del programa se le conoce con el nombre de *Secuencial.*

En los autómatas programables modernos se guardan las instrucciones que forman el programa en una memoria de usuario a la que tiene acceso el microprocesador. Éste lee cada cierto tiempo los estados de las señales de entrada y los almacena en una memoria interna, llamada *Tabla de Entradas*, de donde los recupera a medida que los necesite durante la ejecución del programa de usuario. Según se va ejecutando este programa se generan respuestas de salida que se almacenan en una memoria llamada *Tabla de Salidas*. Una vez completada la ejecución de todo el programa, se actualizan los valores de los estados de las salidas con los valores almacenados en la tabla de salidas.

Por todo lo comentado anteriormente, podemos asegurar que en un autómata programable va a existir un tiempo mínimo de respuesta que será de unos pocos milisegundos, que será función del tiempo que tarde en realizar una pasada por todo el programa de usuario contenido en memoria. Esto se traduce en que durante la ejecución del programa el autómata ignorará los valores de las señales de entrada y de salida (se desconecta de estos puntos).

3.5.3. Componentes de los autómatas programables.

Podemos decir que los autómatas programables se componen de cinco bloques fundamentales:

- Unidad central de proceso.
- Unidades de entrada y de salida.
- Unidad de memoria.
- Fuente de alimentación.
- Unidad de comunicación.

Unidad central de proceso (CPU).

Es la unidad lógica que inicia y organiza todas las actividades. Podemos asegurar que es la inteligencia del sistema, el corazón del autómata. La unidad central de proceso suele ser un microprocesador.

Su trabajo consiste en leer la información que existe en los terminales de entrada a intervalos regulares y actualizar la información de los terminales de salida a partir de la información de los terminales de entrada. En memoria existirá un programa que habrá que ejecutar. Los resultados de la ejecución de este programa activarán o no las salidas en función de los valores que tengan las entradas.

Unidad de entradas/salidas.

Estas unidades convierten las señales externas, generalmente de tensión elevada, en señales de baja tensión que pueda manejar el controlador. Estas señales externas procederán de interruptores, limitadores, relés, sensores analógicos de señales continuas o variables de baja o alta frecuencia, etc. El controlador, en cada ciclo de trabajo, genera un conjunto de señales que la unidad de salida convierte en señales para los actuadores de motores, válvulas, relés, interruptores e indicadores.

Los circuitos de entrada/salida son independientes y están diseñados con criterios de modularidad y flexibilidad, de tal forma que pueda modificarse fácilmente el número de entradas y salidas, así como las funciones que éstas realizan. Además, las unidades de entrada/salida de muchos autómatas poseen dispositivos para la recepción, tratamiento y emisión de señales analógicas, por lo que necesitarán convertidores analógicos-digitales y digitales-analógicos.

- La sección de entradas, acepta y codifica de forma comprensible por la CPU las señales que proceden de los dispositivos de entrada o captadores (pulsadores, finales de carrera, sensores, etc.). También tiene la misión de proteger los circuitos electrónicos internos del

autómata, realizando una separación eléctrica entre éstos y los captadores.

- La sección de salidas trabaja de forma inversa a la de entradas: decodifica las señales que proceden de la CPU, las amplifica y gestiona con ellas los dispositivos de salida o actuadores (lámparas, relés, contactores, arrancadores, electroválvulas,...)

Unidad de memoria.

Las funciones de esta unidad son las mismas que las que realiza en cualquier ordenador. Almacena los datos y los programas de instrucciones tanto de aplicación como de ejecución. Los programas de aplicación se cargan en memoria mediante una consola de programación.

Sin embargo, la tendencia actual es usar ordenadores para la programación Estos llevan terminales gráficos y programas interactivos que facilitan la programación, ya que se puede realizar una comprobación y simulación del programa realizado evitando así posibles errores en tiempo de ejecución.

La unidad de alimentación.

Adapta la tensión de red de 220V y 50Hz a la tensión de funcionamiento de los circuitos electrónicos internos del autómata.

Unidad de comunicación.

A todos estos elementos anteriores debemos unir la unidad de comunicación del autómata con el mundo exterior. Los autómatas modernos permiten ser conectados a redes locales bien a través de un puerto serie, bien a través de una tarjeta más especializada. Gracias a esta conexión con el exterior podremos introducir a los autómatas programables en cualquier tipo de control de una FMC.

3.5.4. Programación.

El autómata actúa como sistema de control de una máquina o proceso. La forma en que el programador del autómata expresa la secuencia de control es a través de un conjunto de instrucciones que constituyen el programa que reside en la memoria del autómata. Estas instrucciones deben estar codificadas en un determinado lenguaje de programación e interesa que sea capaz de explotar al máximo las posibilidades que ofrezca la CPU.

Las instrucciones del programa se almacen*an en la memoria del autómata en forma de código máquina. Será necesario, por tanto, que el equipo de programación traduzca los nmónicos o instrucciones del lenguaje a esta secuencia de ceros y unos.

Tipos de instrucciones.

Las instrucciones pueden ser de varios tipos:

- *Lógicas.* Corresponden a los operadores lógicos booleanos (AND, OR, NOT, XOR). Permiten la implementación de ecuaciones de Boole.
- *Aritméticas. S*uma, resta, multiplicación, división,...
- *Manipulación de datos.* Permiten comparar variables, hacer rotaciones o desplazamientos de bits en un registro, etc.
- *Transferencia.* Permiten copiar datos de un registro a otro.
- *Temporizadores.* Una variable temporizada adquiere el estado de otra variable llamada de control transcurrido un cierto tiempo predeterminado que se denomina *preselección del temporizador*.
- *Contadores.* La variable de salida adquiere el estado '1' cuando se han producido un numero predeterminado de transiciones de '0' a '1' en la variable de cómputo, pero para que estas transiciones sean contabilizadas es necesario que una variable de control esté a '1' en ese momento. También existen instrucciones de contador que funcionan con una variable que lo incrementa y otra que lo decrementa.

- *Control del flujo del programa.* Permiten alterar la ejecución secuencial de las instrucciones del programa mediante saltos, que pueden ser condicionales o incondicionales. También permiten implementar subrutinas.
- *Comunicación.* Permiten la transferencia de información con otro autómata u ordenador a través del puerto serie. Será necesario especificar los datos a transmitir, velocidad de transmisión, etc., y pueden ser de lectura o de escritura.

Lenguajes de programación.

Existen varios tipos de lenguajes de programación:

i. *Esquemas de contactos.* Los programas están formados por una serie de escalones o *rung* que configuran una escalera o *ladder.* Cada escalón contiene símbolos que representan estados, entradas, salidas, temporizadores, contadores, etc. identificados con su correspondiente dirección y combinados de acuerdo con una serie de reglas. Estos lenguajes derivan de la tecnología cableada de relés que se usaba antes de la aparición de los autómatas y fueron adoptados para facilitar la adaptación de los operarios acostumbrados a cablear los armarios de relés. Los símbolos básicos son el contacto NA, el contacto NC y la salida. Son lenguajes sencillos pero poco potentes. Se desarrollaron principalmente en EEUU.

ii. *Booleanos.* Las instrucciones vienen dadas en forma de nmónicos, por tanto un programa será una secuencia ordenada de nmónicos.

En ambos tipos de lenguajes cada instrucción representa una ecuación booleana que expresa la condición de estado de una salida o variable del sistema.

En este sentido, un nuevo tipo de programación de autómatas completamente diferente es el *GRAFCET (Graphe de Comande Etape-Transition).* Se parte de un diagrama funcional que representa la evolución del

proceso fijando las acciones a realizar. Tendremos una secuencia de etapas cada una de las cuales lleva asociada una serie de acciones; para poder pasar de una etapa a la siguiente se deberá cumplir una determinada condición. Es más reciente y resulta muy útil en procesos complejos.

Etapas en la programación.

La tarea de programación del autómata resulta complicada y debe seguir las siguientes etapas:

1. Análisis de la tarea que pretendemos controlar.
2. Análisis del algoritmo de control empleado.
3. Asignación de direcciones a los dispositivos de *E/S* y a las variables de estado que se vayan a usar.
4. Transcripción del esquema lógico calculado al lenguaje del autómata.

3.5.5. El autómata del LOIP. Protocolo de comunicación.

Para finalizar, debemos mencionar que en la FMC del LOIP disponemos de un autómata programable de la casa Allen-Bradley modelo SLC-100. Este autómata se encarga de controlar varios dispositivos ubicados en el conveyor del LOIP. También gestiona las diferentes señales procedentes de las máquinas-herramienta, robots, almacén,... mediante un programa de funcionamiento desarrollado en el lenguaje de esquemas de contactos. Como el resto de elementos de la FMC del LOIP, el autómata se comunica con el exterior a través de un puerto serie RS-232-C.

Protocolo de comunicación del controlador programable SLC™ 100.

Inicialmente, el protocolo para comunicaciones del SLC fue concebido como un sistema de un solo dispositivo. Las comunicaciones entre el controlador SLC y los dispositivos externos se efectuaba inicialmente a través de un enlace de comunicaciones serie RS-422, pero gracias al conversor de interface SLC podemos conectarlo a cualquier puerto RS-232-C.

El protocolo utiliza caracteres ASCII, paridad par y trabaja a 9600 baudios. Los números vienen en formato binario codificado en hexadecimal. El controlador ignora toda transmisión que no aparezca en este formato. Las palabras van estructuradas tal y como sigue: 7 bits de datos, 1 bit de paridad y 1 bit de parada.

Formato de transmisión.

El formato de comando que sigue se utiliza para las comunicaciones entre un dispositivo externo y el controlador.

<carácter a la izquierda><recuento de bytes><código de comando><parámetros>|ID||CRC|<cr>

Los diversos campos se definen como sigue:

- **<carácter a la izquierda>.** Es un carácter ASCII. El símbolo # se utiliza para todos los comandos.
- **<recuento de bytes>.** Dos caracteres ASCII hexadecimales que representan la suma total de los bytes comprendidos dentro del <código de comando> y campos de <parámetros>.
- **<código de comando>.** Se trata de dos caracteres ASCII hexadecimales que son específicos del comando. A lo largo de este capítulo iremos viendo algunos.
- **<parámetros>.** Caracteres ASCII hexadecimales con arreglo a lo que necesite el código de comandos.

- **|ID|.** Dos caracteres ASCII digitales y hexadecimales que identifican el número de mensajes. El dispositivo de programación debe aumentar dicho número en uno por cada transmisión que se efectúe al controlador, a excepción de los mensajes de transmisión que contengan "Z". El controlador verificará que se mantiene el orden frecuencial en todas las transmisiones en las que no aparezca "Z". Cualquier otro mensaje válido que tenga un |ID| de 00 enviado al controlador se aceptará siempre y reiniciará la secuencia de numeración identificadora del mensaje, esperándose que el siguiente mensaje tenga un |ID| de 01. Al efectuar el encendido, el controlador espera un |ID| de 00.

- **|CRC|.** Es un número hexadecimal de 2 dígitos que representa la comprobación cíclica de redundancia para el mensaje, que se deriva de los campos <recuento de bytes>, <código de comando>, <parámetros> y |ID| que usan el polinomio

  X*8 + X*6 + X*5 + X*4 + 1

  Obsérvese que el carácter inicial no se envía a través del cálculo CRC. Así mismo, este cálculo se efectúa en el carácter, antes de que se aplique la paridad a la transmisión o después de que se haya eliminado en la recepción.

- **<cr>.** Retorno de carro.

¿Cómo responde el controlador?

El SLC no responde a ningún mensaje que se reciba con una paridad o CRC incorrectos, o con un error de formato.

Hay ciertos comandos y/o <parámetros> que deben satisfacer las condiciones de rango y/o modo. Si un comando o <parámetro> es ilegal, el SLC responde con el siguiente mensaje:

;E|ID||CRC|<cr>

Si el SLC detecta un fallo, su respuesta sería la siguiente:

;T|ID||CRC|<cr>

Puesto que este mensaje no se envía como respuesta a un estímulo procedente del usuario, el ID no tiene significado alguno.

Si una respuesta procedente del SLC fuese incorrecta (p. Ej.: error de formato, paridad o CRC), podemos solicitar una retransmisión mediante:

;Z|ID||CRC|<cr>

El SLC ignora el ID residente en este mensaje y retransmite su mensaje de datos o reconocimiento más reciente con el mismo ID que se envió anteriormente. Esto permite que el usuario determine si el SLC recibió el último mensaje, puesto que el mensaje ID no casará si fue incorrecto. Por lo general el SLC responde a toda transmisión en el plazo de 0,3 segundos.

3.6. MOVIMIENTO AUTOMÁTICO DE MATERIAL.

En un entorno CIM no es posible entender la existencia de un flujo de información entre todos los niveles que forman el sistema productivo sin una manipulación y transporte de material coordinada por un ordenador u otro elemento de control como puede ser un autómata programable. Debido a esto, podemos asegurar que la manipulación y transporte de materiales y herramientas es tan importante como la manipulación y movimiento de la información.

Los sistemas de transporte automático de materiales unen los diversos elementos de la fábrica, por lo que la disposición de este medio de transporte va a influir directamente en el grado de flexibilidad que queramos conseguir en la instalación. Por lo tanto, es fundamental una disposición que pueda modificarse fácilmente en la que podamos tener un control, en tiempo real, de todos los materiales que están en proceso de fabricación.

Un sistema de movimiento automático de materiales está formado por todos aquellos elementos que intervienen en esta función, es decir, las cintas

transportadoras, los caminos de rodillos, los vehículos guiados automáticamente, las carretillas elevadoras, etc.

Todos estos medios de transporte automáticos están teniendo una gran difusión y desarrollo, pues con ellos el tiempo que una pieza está en proceso de fabricación disminuye. También se consigue rebajar la cantidad de material que se está procesando en un determinado momento, disminuyendo así los tiempos de espera entre puestos o entre estos y el almacén de productos acabados.

3.6.1. Cintas transportadoras. El conveyor del LOIP.

Las cintas transportadoras son sistemas basados en una cinta sin fin, accionada por un motor eléctrico, sobre la que se disponen una serie de palés con piezas y cuyo objetivo es llevar éstos a las máquinas en donde serán procesadas las piezas. Por lo tanto, *su función principal es el transporte de materiales*.

Normalmente los palés se ubican en la cinta por medio de un manipulador situado en el almacén. Una vez situados, la cinta se encarga de transportarlos hasta las estaciones de proceso (workstations). Allí, las piezas son retirados de la cinta momentáneamente para realizar las operaciones de mecanizado, inspección, ensamblaje,... propias de la estación de trabajo. Una vez finalizadas estas operaciones las piezas vuelven a la cinta para continuar su camino. Tras una serie de paradas en las diferentes estaciones que conforman el proceso productivo llegan de nuevo al almacén donde son retirados como piezas ya fabricadas.

El control de estos sistemas de transporte suele realizarse por medio de un autómata programable. Este es el caso de la célula del LOIP, en la que el autómata se encarga de controlar los siguientes elementos del conveyor:

- **Límites de entrevía (Gage Stops)**. Se usan para detener los palés en las diferentes estaciones de trabajo.
- **Pernos posicionadores o posicionadores de palés (Pallet Positioner).** En ocasiones es necesario elevar el palé sobre la superficie del conveyor para que el robot correspondiente pueda

acceder a él más fácilmente. En estos casos, una vez que los límites de entrevía han detenido el palé, se levantan los pernos posicionadores y, en consecuencia, el palé que se encuentra justo encima. Este arreglo evita tener que parar el motor que mueve el conveyor cada vez que un palé llegue a una estación.

- **Pernos de restricción (Restraint Pin).** Son elementos que impiden el paso de palés a una estación.

El autómata también evalúa continuamente el estado de los sensores que detectan la presencia de los palés cuando estos llegan a las estaciones.

El proceso que sigue un palé en cada estación es el mostrado a continuación:

1. Cuando un palé llega a una estación de trabajo presiona un gatillo (sensor de presencia) que le indica al autómata la presencia de dicho palé en la estación. Como consecuencia, el programa del autómata ordena subir los límites de entrevía y el perno de restricción de la estación correspondiente. Los límites de entrevía se levantan inmediatamente, pero el perno de restricción tarda unos segundos para permitir que el palé entre totalmente en dicha estación.
2. Si la pieza que porta el palé requiere algún tipo de operación en la estación, se levantan los pernos posicionadores de palés.
3. Una vez finalizado el proceso de fabricación (o inspección), o bien si no se requiere algún tipo de operación en la estación de trabajo, bajaremos los límites de entrevía y los pernos posicionadores, dejando que el palé pueda continuar su camino. Instantes después, el perno de restricción descenderá para permitir que otro palé entre en la estación.

3.7. ELEMENTOS AUXILIARES. COMPRESOR DE AIRE.

A parte de los equipos que hemos mencionado anteriormente, en una célula de fabricación flexible podemos encontrar una serie de dispositivos auxiliares como pueden ser los sitemas de refrigeración, compresores de aire, generadores de corriente, sistemas de identificación, etc. En el LOIP sólo disponemos de un equipo auxiliar: el compresor de aire.

Los compresores de aire son elementos muy comunes dentro de las fábricas. Esta circunstancia se debe, principalmente, a que multitud de equipos industriales utilizan el aire comprimido para efectuar algunas operaciones. A continuación se enumeran algunas de las razones que explican su elevada utilización:

i. Facilidad de transporte.
ii. No necesita circuito de retorno.
iii. Puede ser almacenado.
iv. No puede arder.
v. Es limpio y no contamina.
vi. No es muy sensible a la temperatura.

Sin embargo no todo son ventajas:

i. Necesita de una adecuada preparación y tratamiento previo.
ii. Desarrolla una fuerza limitada.
iii. Al ser compresible, se limita la calidad del movimiento.

3.7.1. Elementos de una instalación de aire comprimido.

Se pueden diferenciar varias partes en la instalación de aire comprimido:

1 - Compresores.

2 - Depósitos acumuladores.

3 - Unidad de acondicionamiento.

4 - Conducción.

5 - Unidades de mantenimiento.

Compresores.

Un compresor neumático es un dispositivo que aspira aire a presión atmosférica y la eleva hasta la presión que requiere la instalación. Los datos técnicos necesarios para adecuar el grupo compresor a una instalación son el caudal suministrado y la relación de compresión.

Depósitos acumuladores.

Los depósitos acumuladores tienen como misión almacenar el aire a una determinada presión. Entre las principales razones por las que es necesario esta acumulación se encuentran las siguientes:

- Eliminar el rizado en la presión del aire originado por el carácter pulsante de los compresores.
- Disponer de una cantidad de aire a presión suficiente para responder a las demandas punta de caudal por parte de los consumidores.
- Permitir que el aire se enfríe para que el vapor de agua, que inevitablemente contiene el aire, condense.

Unidad de acondicionamiento.

Esta unidad es necesaria debido a que el aire comprimido transporta impurezas y vapor de agua. Estos elementos son perjudiciales para la instalación ya que pueden provocar obstrucciones en secciones de paso estrechas, oxidación de partes metálicas y una disminución de la lubricación de las partes móviles.

Las partículas de polvo son parcialmente eliminadas por un filtro de aire, pero para eliminar la humedad se emplean varios métodos:

- *Enfriamiento*. Ya se hizo mención a este método en el apartado de los depósitos acumuladores por tratarse del procedimiento empleado en el LOIP.
- *Adsorción*. Separa el vapor de agua por un filtrado físico mediante un gel de óxido de silicio que adsorve la humedad.
- *Absorción*. Se basa en una reacción química que elimina el vapor de agua del aire.

Conducción.

Una vez realizado el tratamiento del aire, éste se distribuye por la línea principal. De esta línea parten las diferentes derivaciones que conducen el aire hasta las máquinas que lo precisen.

Las conducciones de la célula son de cobre, pero se pueden realizar en acero o plástico. Se suele aconsejar el uso del plástico, principalmente porque se evitan virutas procedentes de la soldadura o de la oxidación de los tubos, pero presenta el inconveniente de ser más caro.

Las conducciones deben tener una pendiente mínima del 2% que permita la purga del agua. También se deben tener en cuenta la adecuada estanqueidad, resistencia a la corrosión, mínimas caídas de presión, etc.

Unidades de mantenimiento.

Justo antes de que el aire acceda a cada área de la instalación se intercala esta unidad, cuyas funciones son:

3 Filtrado de partículas de tamaños superiores a 10 micras.

3 Regulación de la presión, reduciéndola a un valor fijo.

3 Lubricación que facilite el movimiento de partes móviles.

Podemos apreciar esta unidad en la parte posterior de las máquinas-herramienta de la célula del LOIP.

3.7.2. Compresor del LOIP.

Cuatro de las máquinas de la FMC precisan del aire comprimido para su funcionamiento. El *almacén* necesita aire comprimido para la manipulación de su brazo robotizado: abrir y cerrar pinza, entrar y salir de las celdas, girar hacia fuera y hacia dentro, etc. El *torno* y la *fresadora* utilizan el aire comprimido para controlar las mordazas del cabezal y la puerta de protección, así como para llevar a cabo el cambio de herramienta. Por último, el aire comprimido nos permite levantar y bajar los pernos de restricción, pernos posicionadores y límites de entrevía de las diferentes estaciones situadas en la *cinta transportadora*.

En el LOIP se dispone de un compresor que proporciona aire comprimido a una presión de entre *5,5* y *6* bares que nos va a permitir la realización de todas las operaciones mencionadas anteriormente.

Es aconsejable, aunque no es imprescindible, encender y apagar el compresor cada vez que se utilice, y vaciar su depósito. Lo que se pretende mediante esta operación es evitar la humedad propia del proceso de compresión. Este mantenimiento repercute positivamente en la duración de los elementos neumáticos de las máquinas.

CAPÍTULO CUARTO:

EL CONTROL REMOTO DE LAS MÁQUINAS.

4.1. INTRODUCCIÓN.

Tras conocer las características más relevantes de cada una de las máquinas que componen la célula del LOIP, es el momento de describir el funcionamiento de cada uno de los módulos de control desarrollados.

En este capítulo sólo nos centraremos en los aspectos relacionados con el control remoto de los elementos y nos olvidaremos de aquellos que están vinculados a la integración de la FMC. Dicha materia será tratada en el capítulo 5 de esta memoria, donde descubrirnos el resto del cuerpo del programa FMC.

En consecuencia, los capítulos cuarto y quinto podrían constituir el *manual de usuario del programa 'FMC'.*

4.2. ALMACÉN ASRS.

Siempre que necesitemos realizar alguna operación con los palés, o debamos modificar el inventario del almacén, tendremos que seleccionar la opción *Almacén* del menú *Control Remoto* de la ventana principal, o bien presionar la combinación de teclas <Ctrl+F2>. El programa FMC se encargará de abrir y configurar el puerto seleccionado para el almacén.

Tras establecerse la comunicación entre el ordenador central (host) y el sistema controlador del almacén, en nuestra pantalla aparece la ventana que se muestra en la figura 4.1.:

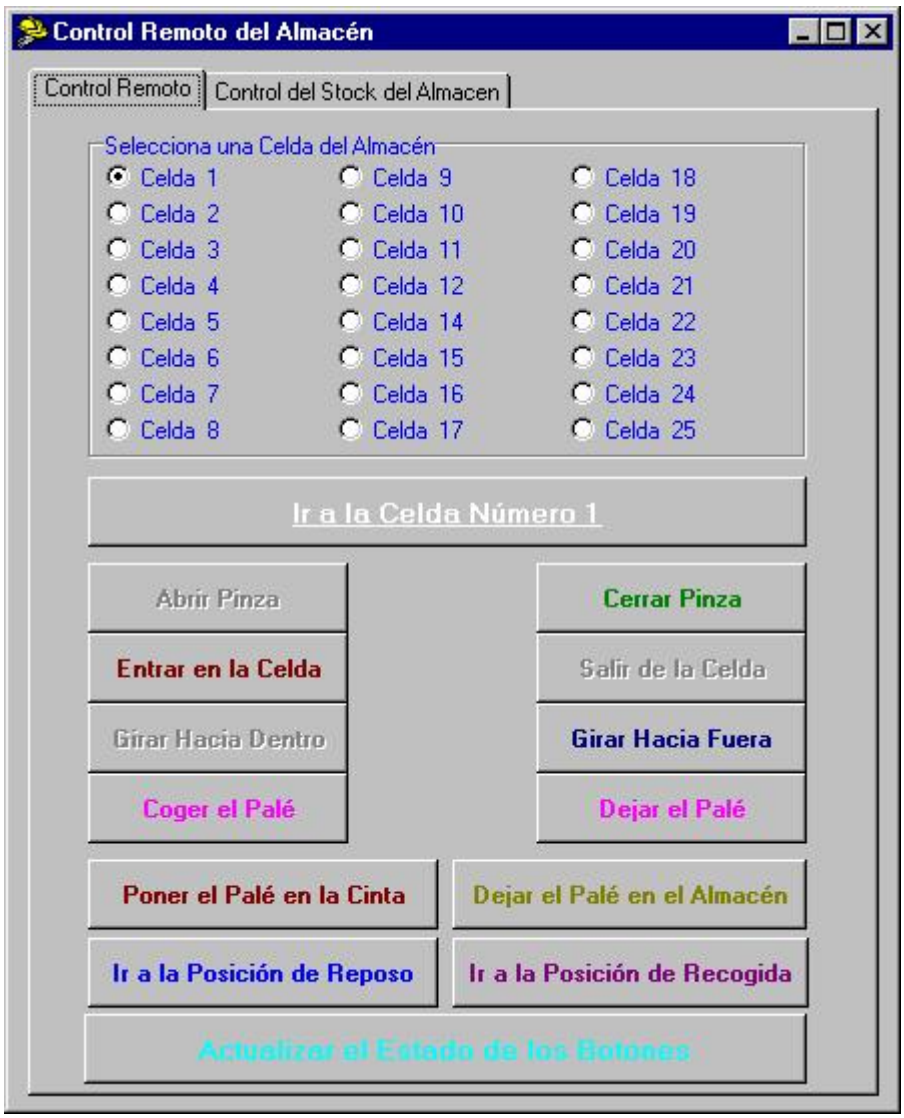


Figura 4.1. *Aspecto inicial de la ventana de control remoto del almacén.*

El elemento principal de esta ventana es un control *TPageControl* constituido por dos hojas o páginas. La primera de ellas, *Control Remoto*, nos va a permitir gobernar el movimiento del brazo robotizado del almacén.

En la parte superior de esta hoja se encuentra un control *TRadioGroup* con 24 botones que representan 24 celdas del almacén. Se ha elegido este formato para poder acceder de forma rápida y cómoda a cada una de las celdas.

Pero, si el almacén tiene 25 celdas, ¿por qué sólo podemos acceder a 24?. Esta situación se debe a un problema con el software actual del almacén. Este software utiliza un formato de transmisión cuyos caracteres finales son <número de celda> y <\r>, y es en este punto donde aparece el conflicto. El carácter ASCII <\r>, o retorno de carro, equivale al valor 13, por lo que cualquier operación que suponga un desplazamiento hasta la celda número 13 no se va a efectuar correctamente. Esta situación se origina porque el controlador del almacén recibe una orden incompleta (todos los caracteres que reciba después del primer retorno de carro forman parte de otra sentencia). Para evitar esta contrariedad se ha optado por impedir el acceso a esta celda.

Inmediatamente por debajo del control *TRadioGroup* encontramos los diferentes botones encargados del control remoto del brazo robotizado. Conozcámoslos:

- ***Ir a la Posición #.*** Este botón ordena el desplazamiento hasta la posición cartesiana que define la celda marcada en el *TRadioGroup*.
- ***Abrir y Cerrar Pinza***. Estos botones se encargan del control de la pinza del robot.
- ***Girar hacia Dentro/ Girar hacia Fuera.*** Se encargan de poner el brazo robotizado orientado hacia el almacén o hacia el conveyor respectivamente.
- ***Entrar/Salir.*** Con estos botones podemos ordenar al robot robotizado que entre y salga de las diferentes celdas del almacén.
- ***Dejar/Quitar Palé en la Cinta.*** Son los encargados de trasladar los palés a la cinta y de retirarlos.
- ***Posición de Espera***. Envía el brazo del robot a la posición de espera del almacén.
- ***Posición de Recogida.*** Nos permite situar el brazo en la posición de recogida del almacén.

La siguiente hoja del control *TPageControl* es *Control del Inventario del Almacén,* que aparece en la figura 4.2.

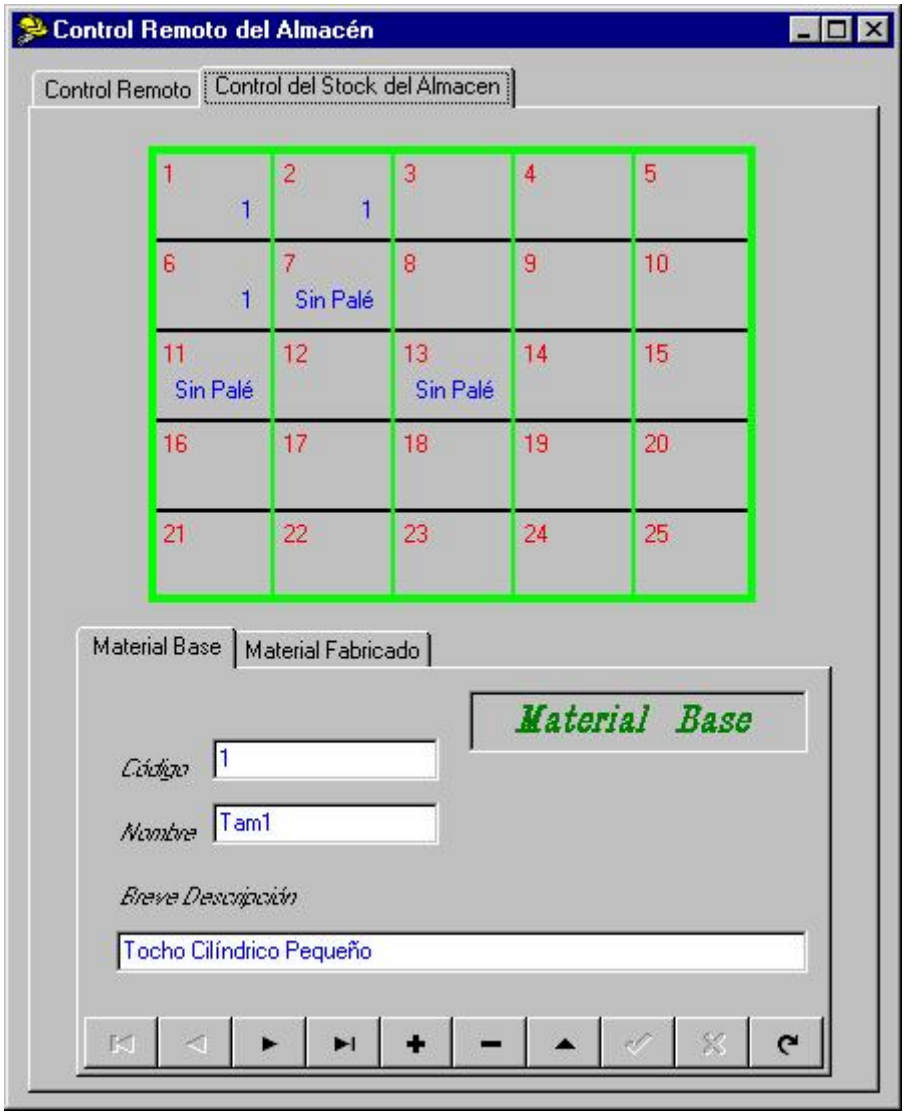


Figura 4.2. *Aspecto de la ventana encargada de la gestión del stock.*

En la parte superior de esta segunda página descubrimos una figura que representa el almacén con sus celdas. Podemos comprobar, visualmente, que en el área destinada a cada celda se muestra el tipo de material que la ocupa o la inexistencia de palé. Inmediatamente por debajo de la panorámica del almacén, encontramos otro control *TPageControl* que nos va a permitir acceder a las bases de datos de las piezas que vamos a fabricar y de los materiales de partida necesarios para su elaboración.

Tal y como se muestra en la figura 4.2., inicialmente aparece activa la hoja correspondiente a las piezas de partida (página *Material Base*). En esta página vamos a poder introducir las piezas que van a servir como base en el proceso de producción: su código, su nombre y, si se considera necesario, una breve descripción. El control *TDBNavigator,* situado en la parte inferior, nos va a facilitar la navegación por los datos de esta tabla. Además, nos permitirá realizar la mayoría de las operaciones que están permitidas en una tabla con tan sólo una pulsación.

Tras haber introducido los datos relativos a las piezas en bruto, en la página *Material Fabricado* se definen las características de las piezas que se van a fabricar, es decir, el producto elaborado en la FMC (véase figura 4.3.).

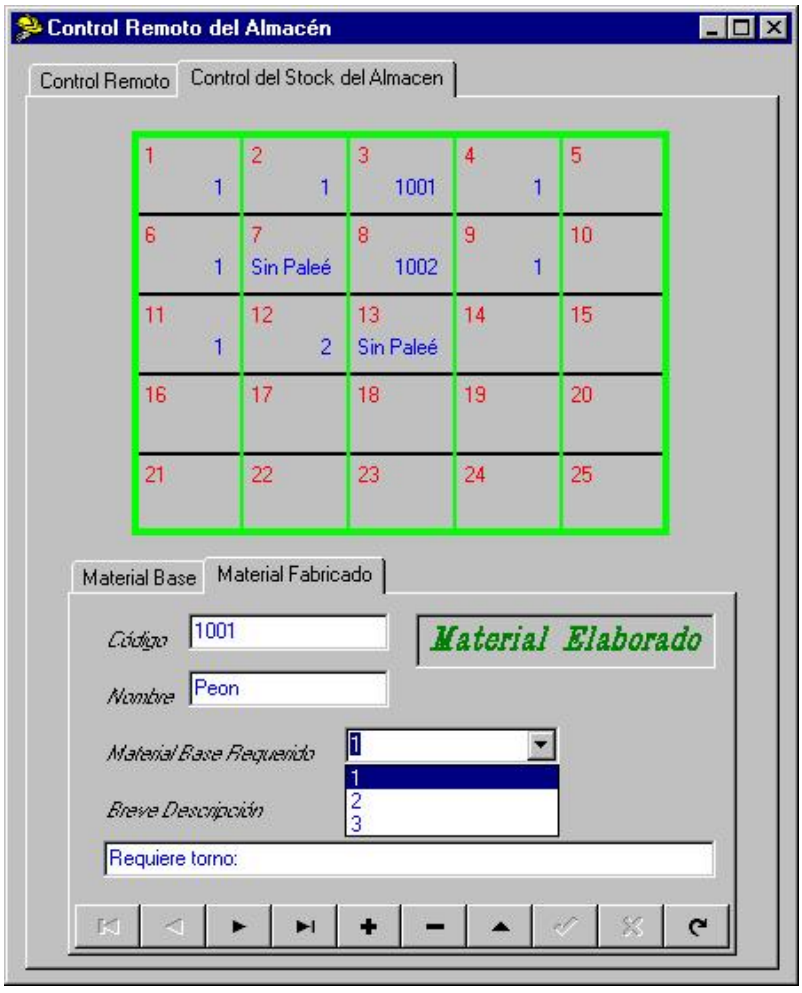


Figura 4.3. *Datos relativos a los materiales elaborados.*

Al igual que ocurría con las piezas en bruto, en esta nueva hoja vamos a encontrar los campos necesarios para describir nuestro producto: código de identificación, nombre, pieza en bruto requerida y una breve descripción. El control en el que introducimos el material de partida para la fabricación del producto es un control *TComboBox*. Dicho elemento despliega una lista en la que aparece el código de todos y cada uno de los diferentes elementos que introdujimos en la tabla anterior. Esto nos permite seleccionar el material base más rápida y fácilmente. Como antes, un control TDBNavigator nos permite el acceso y control de la tabla.

Tras rellenar todos los datos anteriores, es necesario modificar el estado de las celdas del almacén con el fin de gestionar correctamente el inventario. Para ello, tan sólo debemos situar el ratón sobre la celda que queramos modificar y presionar el botón derecho. Aparece un menú emergente con tres opciones:

- La primera, ***Introducir Nuevo Material en la Celda***, nos permite introducir el tipo de pieza que hay en el almacén. Esta opción da paso a un pequeño cuadro de diálogo en el que podremos seleccionar el tipo de pieza a introducir de entre los muchos disponibles (materiales en bruto y piezas fabricadas).

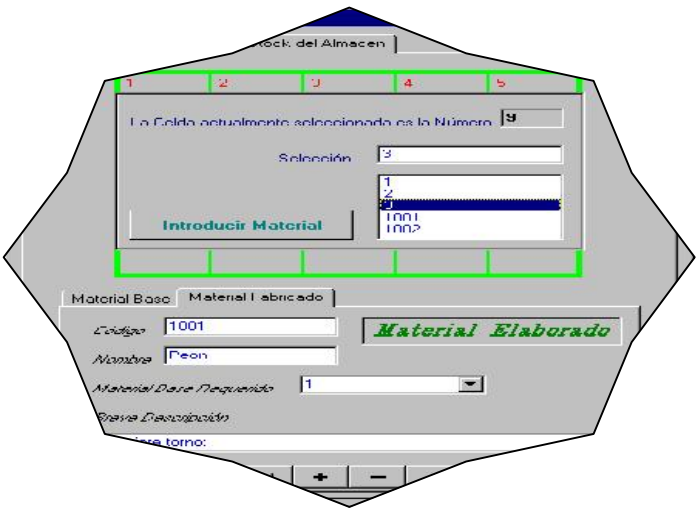


Figura 4.4. *Cambiar los valores de las celdas.*

Tras introducir el material apropiado, el cuadro de diálogo desaparece y el material seleccionado se muestra en la celda correspondiente.

- La segunda opción del menú emergente es ***Inicializar el Valor del Material de la Celda***. Esta opción nos permite incicializar los valores de la celda, es decir, al seleccionar esta alternativa la celda deja de estar ocupada por una pieza, aunque no por un palé. Para indicar que una celda se encuentra sin un palé deberemos presionar la tercera opción, ***Actualmente NO hay Palé en esta Celda***.

Para finalizar este apartado destinado al control remoto del almacén vamos a concluir con una breve observación. Como sabemos del capítulo 3, el controlador del almacén es, normalmente, el encargado de su gestión. Hasta la ejecución de este proyecto, el almacén de la FMC del LOIP era gestionado de esta forma. Sin embargo, el software FMC permite administrar el almacén desde el propio ordenador central.

En este sentido, si el sistema controlador del almacén puede gestionar el inventario por sí mismo, ¿por qué gestionarlo completamente desde el ordenador central?. Existen diversas razones, pero sin duda alguna las dos más importantes son la comodidad que esto supone y la imposibilidad actual, debido al software de control del almacén, de transmitir los ficheros de stocks entre el controlador del ASRS y el host. Este último impedimento había sido solucionado en los proyectos anteriores mediante el transporte físico de los datos en disquetes, lo que suponía una pérdida importante de flexibilidad y tiempo. Por lo tanto, al gestionar el almacén desde el propio ordenador central conseguimos aumentar la flexibilidad y la transparencia de nuestro sistema productivo.

4.3. CONTROL REMOTO DEL AUTÓMATA.

Antes de proceder a analizar el módulo encargado del control remoto del autómata, conviene revisar las características del protocolo de comunicaciones del autómata que vimos en el apartado 3.5.5. del capítulo anterior. Estas peculiaridades nos servirán para comprender mejor su funcionamiento.

Este módulo del programa es el encargado de controlar el autómata Allen-Bradley SLC-100. Este autómata gobierna el estado de los pernos posicionadores de palés, los pernos de restricción y los límites de entrevía de la cinta. También es el encargado de distinguir el estado en el que se encuentran las máquinas asociadas a cada estación.

En el menú *Control Remoto* de la ventana principal de FMC encontramos una opción, llamada *Autómata*, que da paso a la ventana mostrada en la figura 4.5. Obtenemos el mismo resultado presionando la combinación de teclas <Ctrl+F1>.

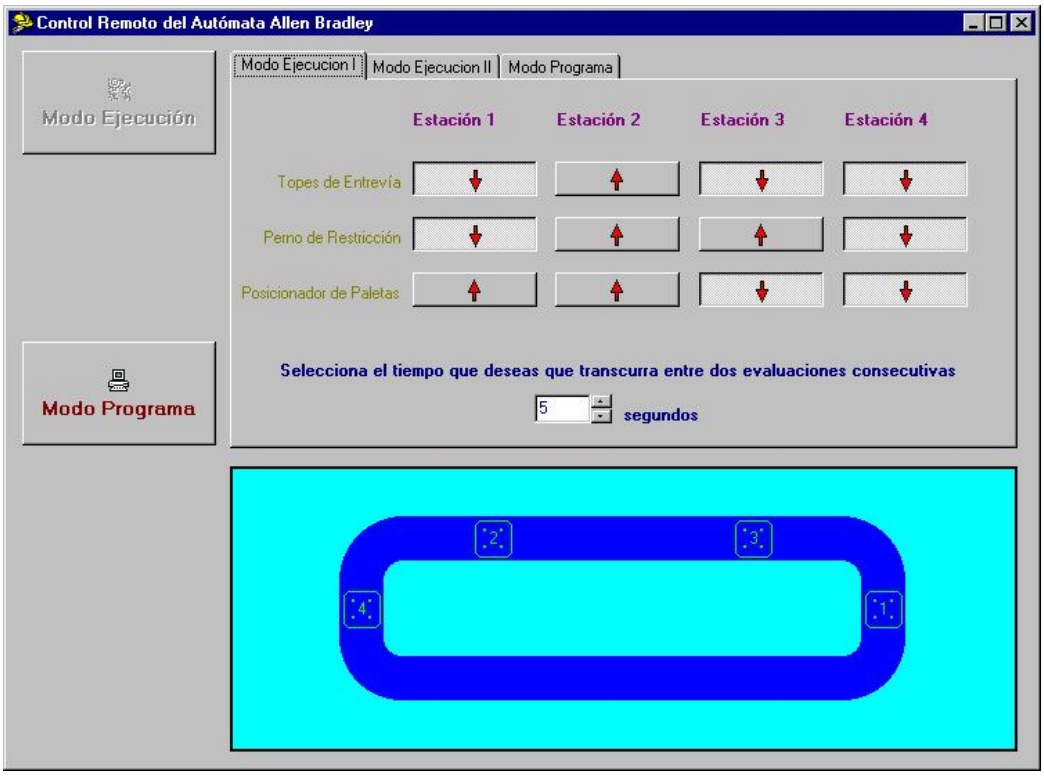


Figura 4.5. *Aspecto inicial de la ventana de control del autómata.*

En la sección izquierda de dicha ventana podemos apreciar dos botones. El primero de ellos, *Modo Ejecución,* se encarga de poner al autómata en el *Modo de Funcionamiento,* es decir, se encarga de ejecutar el programa residente en la memoria del PLC. El segundo botón, *Modo Programa*, es el botón que nos permite detener la ejecución del programa del autómata, pasando del *Modo de Funcionamiento* al *Modo de Programa*. Al iniciar una sesión de trabajo en este módulo, el programa se encarga automáticamente de poner al autómata en modo funcionamiento, por lo que el botón *Modo Ejecución* aparece inicialmente inactivo.

Para poder seleccionar el *Modo de Ejecución*, el código asociado se encarga de enviar al controlador del PLC la siguiente cadena:

#0403010180|ID||CRC|<cr>

mientras que el botón *Desactivar Programa* establece el *Modo de Programa* enviando la cadena:

#0403010106|ID||CRC|<cr>

A continuación vamos a conocer las opciones correspondientes a estos dos modos de trabajo.

Modo programa.

En la zona superior derecha encontramos un control *TPageControl* con tres páginas. Una de ellas, *Modo de Programa*, contiene las herramientas que nos van permitir tanto borrar la memoria del PLC como enviar y recibir el programa de usuario.

La figura 4.6. nos presenta el aspecto de la página *Modo de Programa* dentro del módulo encargado del control remoto del autómata.

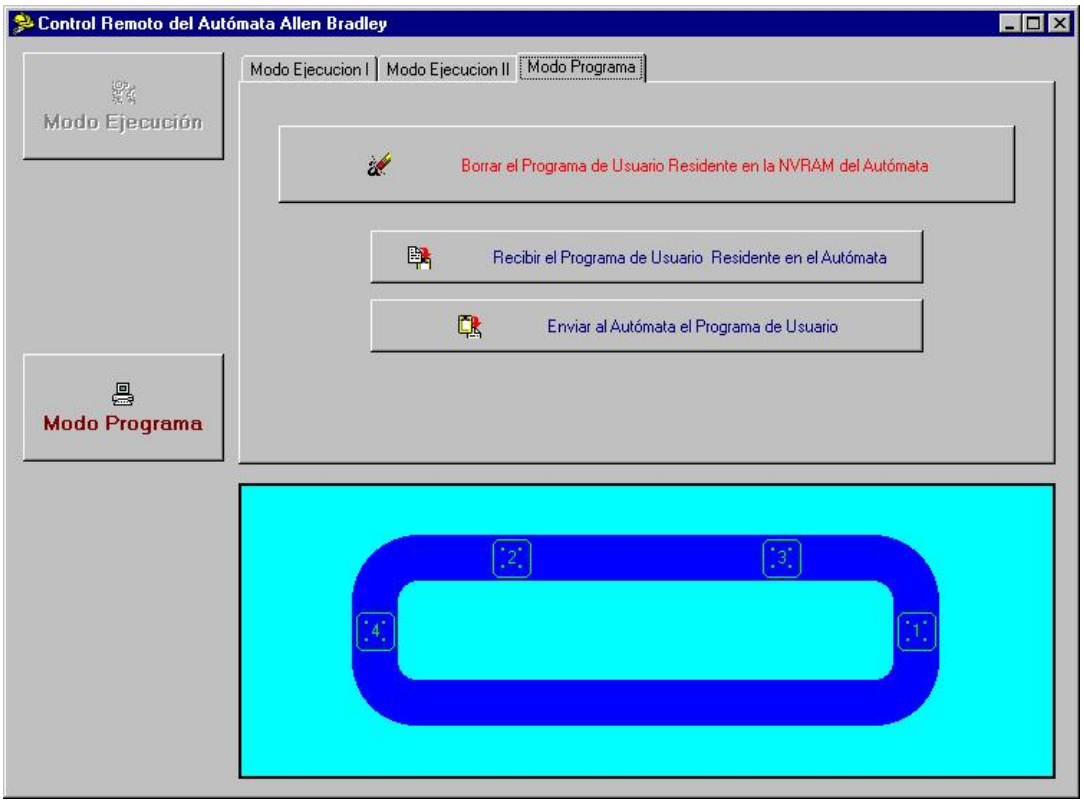


Figura 4.6. *Ventana Modo Programa.*

- Botón **Borrar el Programa de Usuario Residente en la NVRAM del Autómata**. Este control nos permite borrar el programa de usuario residente en la NVRAM del SLC 100. El programa FMC nos muestra un cuadro de diálogo solicitándonos confirmación para efectuar dicha operación. En caso afirmativo, el código de programa envía al controlador del autómata la siguiente cadena:

  :01019600|CRC|<CR>

  Este comando es válido tan solo en el *Modo de Programa*, lo que quiere decir que el programa de usuario no puede estar ejecutándose mientras le estamos borrando. Por este motivo, la primera acción que realiza el programa es detener la ejecución del programa residente (el botón *Modo Programa* se deshabilita mientras que el botón *Modo Ejecución* se activa) y entrar en *Modo de Programa*.

- Botón **Recibir el Programa de Usuario Residente en el Autómata.** Este es el control encargado de recibir el programa de funcionamiento que actualmente reside en la memoria del autómata. Al pulsar este botón aparece en la parte inferior de la imagen una barra en la que se muestra la información de las diferentes líneas del programa que se está recibiendo del controlador. Tras unos instantes, la barra desaparece, y en su lugar surge un cuadro de diálogo que nos da la posibilidad de guardar los datos recibidos en un archivo. Este cuadro de diálogo nos muestra todos los archivos que contienen programas de usuario del autómata (archivos de extensión CIM), los cuales están ubicados en el directorio "C:\Celula00_01\Automata". Actualmente, el programa almacenado en la memoria del controlador es *Residente.cim*.

  Para poder recibir las líneas del programa debemos enviar la siguiente cadena:

  #0100|ID||CRC|<cr>

  lo que significa enviar el programa de usuario residente en la NVRAM del SLC 100 a la terminal de usuario (ordenador central o 'Host'). Este comando es válido tan sólo en el *Modo de Programa*, por lo que antes de poder enviar la cadena anterior, la aplicación cambia automáticamente a este modo desactivando la ejecución del programa residente.

  El SLC responde con la siguiente cadena:

  :<bloque #><15 bytes>|ID||CRC|<cr>

  donde <bloque #> representa dos caracteres ASCII hexadecimales comprendidos entre 00H y 7AH (el carácter H nos indica que estamos hablando de valores Hexadecimales). El proceso de recepción es un proceso secuencial en el que deberemos solicitar las 123 líneas que conforman el programa una por una: pedimos la línea número 0, la recibimos, pedimos la línea 1, la recibimos,... y así hasta recibir las 123 (7B en hexadecimal) líneas que el controlador permite almacenar.

Tras recibir todos las líneas del programa, la aplicación se encarga de volver a poner al autómata en *Modo Ejecución* (se ejecuta el programa residente en la memoria del PLC).

- Botón **Enviar al Autómata el Programa de Usuario.** El código asociado a este botón se encarga de enviar al autómata un programa de funcionamiento. Al pulsar dicho componente aparece un cuadro de diálogo como el de la figura 4.7., que nos da la opción de elegir el archivo que contiene el programa de usuario que deseamos enviar al controlador programable SLC 100. Este cuadro de diálogo nos mostrará todos los archivos que contienen programas de usuario para el autómata, que están ubicados en el directorio "C:\Celula00_01\ Automata".

Figura 4.7. *Cuadro de diálogo Enviar Programa al Autómata.*

Tras seleccionar un archivo y presionar el botón Aceptar, el cuadro de diálogo desaparece y en la parte inferior de la imagen surge un panel donde se muestra la información que se está enviando al controlador. Tras unos instantes, la barra desaparece, y en su lugar se presenta un mensaje que nos informa del éxito de la transferencia.

El formato de transmisión, en este caso, es el siguiente:

#1101<número de bloque><15 bytes>|ID||CRC|<cr>

lo que implica que vamos a recibir un bloque de programa procedente de la terminal de usuario y lo vamos a cargar en la NVRAM del SLC 100. Al igual que los dos comando anteriores, éste es válido tan solo en el *Modo de Programa*, por que lo antes de poder enviar la cadena anterior, la aplicación cambia automáticamente a este modo desactivando la ejecución del programa residente. <número de bloque> está formado por dos caracteres ASCII hexadecimales, comprendidos entre 00H y 7AH, y deben enviarse secuencialmente.

Si la transmisión se ha efectuado correctamente, el SLC responde después de cada transmisión con la siguiente cadena.

:R|ID||CRC|<cr>

Pero, ¿qué información es la que se guarda en cada una de las 123 líneas? Los datos se cargan tal y como se muestra en la tabla 4.1.

Numero de bloque.	Datos.
00H	Los primeros 15 bytes de la memoria del programa de usuario.
01H	Los siguientes 15 bytes de la memoria del programa de usuario.
...	
...	
75H	Los últimos 15 bytes de la memoria del programa de usuario.
76H	Tres bytes que contienen el final de la información del programa más los primeros 12 bytes de la tabla preseleccionada.
77H	Los próximos 15 bytes de la tabla preseleccionada.
78H	Los próximos 15 bytes de la tabla preseleccionada.
79H	Los próximos 15 bytes de la tabla preseleccionada.
7ªH	Los 7 bytes de la tabla preseleccionada más 8 bytes de relleno.

Tabla 4.1. *Información que se guarda en cada línea del programa.*

Modo ejecución.

Junto a la página *Modo Programa*, el control *TPageControl* incluye otras dos páginas, *Modo Ejecución I y Modo Ejecución II*, que nos van a permitir controlar las diferentes señales de cada una de las estaciones de la cinta transportadora.

Ambas páginas están formadas por cuatro filas de botones, cada una de ellas asociada a una de las cuatro estaciones que posee la cinta transportadora. Para conocer en todo momento la estación sobre la que estamos operando, en la parte inferior de la ventana se muestra un dibujo del conveyor en el que se enumeran las diferentes estaciones.

En cada una de las cuatro estaciones existen 7 variables binarias que vamos a poder controlar.

- **Límites de Entrevía.** Los límites de entrevía son dos elementos metálicos que impiden que el palé sobrepase la estación correspondiente. Si este botón esta pulsado, y con una flecha apuntando al suelo, el límite de entrevía estará bajado, mientras que si el límite de entrevía ha sido accionado y se encuentra en su posición superior, el botón asociado a dicho límite estará levantado y la flecha dibujada estará orientada hacia arriba. Pulsando en cualquier momento sobre el botón podremos levantar o bajar los límites según lo consideremos oportuno.

- **Perno de Restricción.** El perno de restricción es un pequeño cilindro metálico que sube para impedir el paso de palés a la estación. Al igual que ocurría para los límites de entrevía, si el botón correspondiente aparece pulsado o con la flecha hacia abajo, nos estará indicando que el perno no se ha levantado, mientras que si no aparece pulsado, y la flecha orientada hacia arriba, el perno está levantado impidiendo el paso de cualquier palé.

- **Pernos Posicionadores de Palés.** Los posicionadores son cuatro cilindros metálicos que permiten elevar el palé por encima de la cinta transportadora. Están situados formando un cuadrado perfecto.

Estos tres tipos de botones que acabamos de comentar (límites de entrevía, pernos posicionadores de palés y pernos de restricción) pueden ser modificados en la página *Modo Ejecución I* (figura 4.5.). Por el contrario, los botones que se muestran en la figura 4.8. corresponden a la hoja *Modo Ejecución II.*

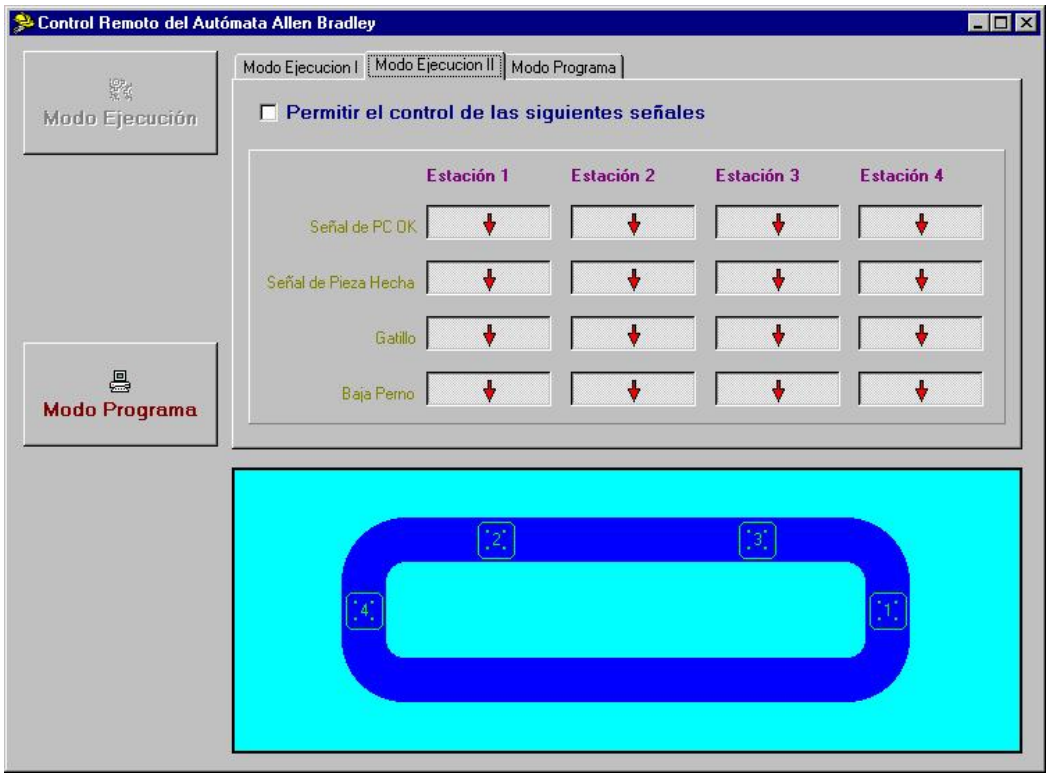


Figura 4.8. *Modo Ejecución II.*

- **Señal PC OK.** Es la señal que genera el ordenador central y que se envía al autómata programable para comunicarle que el ordenador se ha percatado de que la máquina asociada a la estación correspondiente ha finalizado la fabricación de una pieza. Al enviar esta señal, el autómata ordena bajar los límites de entrevía y el perno de restricción de la estación, dejando que el palé siga su movimiento por la cinta hasta la siguiente estación.

- **Pieza Hecha.** Este campo tiene diferente significado dependiendo de la máquina a la que nos estemos refiriendo. Así, para el caso del almacén, este campo representa que dicha máquina está ocupada cuando el botón está pulsado, y que está ociosa cuando el botón está levantado. El significado para el resto de las máquinas es algo diferente, ya que si estos botones están levantados indicarían que la máquina asociada a esa estación habría acabado de mecanizar la pieza y que todavía no se habría atendido, por parte del ordenador central, dicha acción. Una vez que el ordenador atienda la información de *pieza hecha*, este botón aparecerá presionado.

- **Gatillo.** Nos ofrece información de cómo está el interruptor o sensor de presencia. Si el detector de presencia ha sido accionado por el paso de un palé, el botón correspondiente no aparecerá pulsado. Por el contrario, si no ha sido accionado aparecerá en su posición inferior. Cuando un palé acciona el interruptor de presencia, inmediatamente el programa del autómata se encarga de levantar los límites de entrevía para impedir que el palé traspase la estación. Unos instantes después le toca el turno al perno de restricción, impidiendo así el paso de otro palé.

- **Baja Perno.** Pulsando sobre este tipo de botones podemos hacer que el perno de restricción baje si se encuentra en su posición superior. Este resultado también lo obtendríamos presionando el botón Perno de Restricción. Entonces, ¿por qué tener dos botones iguales?. Realmente no son iguales, dado que Baja Perno no puede levantar el perno de restricción. Además, se decidió introducir este botón porque, como se verá más adelante, las direcciónes de los relés asociados a cada estación poseen un bit que controla este tipo de señal.

Para poder modificar el estado de alguno de estos últimos botones deberemos activar el campo que se encuentra en la parte superior de esta página.

Tras conocer para qué sirven cada uno de los botones de ejecución surge una pregunta, ¿cómo actúa el programa para ordenar al autómata el cambio de las señales?. Cada una de las estaciones está gobernada por un relé interno que tiene asociada una dirección hexadecimal. En concreto, las direcciones correspondientes a cada una de las cuatro estaciones se muestran en la tabla 4.2.

Estación		Dirección hexadecimal
Relés Internos	Estación 1	A6
	Estación 2	A7
	Estación 3	A8
	Estación 4	A9

Tabla 4.2. *Direcciones de los relés de cada estación.*

Las direcciones para las instrucciones se agrupan en el controlador en número de 8 por cada byte. Los bits se disponen en el byte, partiendo desde la dirección inferior de usuario en el bit menos significativo, hasta la dirección superior de usuario en el bit más significativo. En concreto, para los relés anteriores, cada bit controla alguno de los elementos y las señales que hemos visto anteriormente (tabla 4.3.).

Bit	Significado
0	Controla el Límite de Entrevía de cada estación.
1	Controla el Perno de Restricción de cada estación.
2	Señal que habilita / deshabilita el posicionador.
3	Señal de PC OK.
4	Señal de Pieza Hecha / Almacén Libre.
5	Señal asociada al interruptor de presencia de cada estación.
6	Baja el perno de restricción.
7	No se utiliza

Tabla 4.3. *Información de cada byte.*

Para poder modificar alguna de estas señales en el autómata, debemos enviar la cadena siguiente:

#040300<dirección><byte de datos>|ID||CRC|<cr>

donde <byte de datos> son dos caracteres ASCII hexadecimales y <dirección> es un valor ASCII de dos caracteres hexadecimales que representan la dirección de los relés internos (tabla 4.2). Si la escritura se realiza de forma correcta el autómata nos devolverá la cadena:

:R|ID||CRC|<cr>

Debemos resaltar que el autómata no nos informa directamente de un cambio en alguno de los bits anteriores, es decir, para conocer si un bit ha sufrido alguna modificación debemos solicitar al autómata que nos proporcione el estado actual de los bits de cada relé. Por este motivo, el programa se encarga de evaluar el estado de los relés cada ciertos segundos. El tiempo que transcurre entre cada evaluación puede ser determinado por el usuario en la página *Modo Ejecución I*. En este caso, la cadena que enviamos al PLC es la siguiente:

#0502000100<dirección>|ID||CRC|<cr>

donde, al igual que antes, <dirección> son dos caracteres ASCII hexadecimales que representan la dirección de los relés internos correspondientes. Si el formato es correcto, el SLC responde con:

:000100<dirección><bytes de datos>|ID||CRC|<cr>

De esta forma podemos conocer el estado actual de los bits y, comparándoles con el estado anterior, podemos conocer si alguno ha sufrido alguna modificación en el transcurso de los segundos que han pasado.

4.4. CONTROL REMOTO DE LAS MÁQUINAS-HERRAMIENTA.

Como sabemos, en el LOIP existen dos máquinas-herramienta de control numérico: un torno y una fresadora.

4.4.1. Control remoto del torno.

Dentro del menú *Control Remoto* de la ventana principal encontramos una opción, denominada *Torno*, que da paso a la ventana de la figura 4.9. Podemos conseguir el mismo resultado presionando la combinación de teclas <Ctrl+F3>.

Figura 4.9. *Ventana destinada al control remoto del Torno.*

. En la parte superior de esta ventana encontramos dos botones, *Enviar Proyecto.mir* y *Enviar Inicio.mir*, que se encargan, precisamente, de transmitir al torno la orden de ejecutar los programas *Proyecto.mir* e *Inicio.mir* respectivamente.

Inmediatamente por debajo de estos botones encontramos un control *TFileListBox* que contiene todos los archivos de control numérico que el torno puede ejecutar. Tras seleccionar alguno de ellos, podremos ordenar al torno su ejecución gracias al botón *Enviar Archivo Seleccionado.*

En la parte inferior de esta ventana hallamos otro botón que nos permite un acceso rápido al programa MEditor que nos permite visualizar el código de los programas de control numérico disponibles.

Antes de continuar con la ventana asociada a la fresadora, vamos a analizar porqué son necesarios los botones *Enviar Proyecto.mir e Inicio.mir.*

- El programa *Inicio.mir* es el programa indispensable para poder iniciar un proceso de ejecución de la célula.
- El programa *Proyecto.mir* tiene una función más complicada. Puede darse el caso de que mientras estemos ejecutando una tarea se produzca un error. Si en ese preciso momento el brazo del robot Mitsubishi se encuentra dentro del centro de mecanizado, no abandonará dicha ubicación dado que el controlador del robot estará esperando una señal que nunca va a llegar. Una posible solución sería reiniciar las máquinas implicadas manualmente, pero existe otra solución más cómoda: enviar el programa Proyecto.mir. De esta forma el robot regresará a su posición de espera al igual que el torno.

  Este problema suele producirse, principalmente, cuando el compresor no proporciona el aire a la presión suficiente, lo que genera la aparición en el torno del mensaje *"error: Low Air Pressure"*. En este caso se deberá proceder a purgar el compresor hasta que empiece a cargar de nuevo, y se desbloquearán las máquinas tal y como se indicó anteriormente.

4.4.2. Control remoto de la fresadora.

Para poder acceder al módulo encargado del control remoto de la fresadora tan sólo debemos pulsar la opción *Fresadora* del menú *Control Remoto* de la ventana principal, o bien presionar la combinación de teclas <Ctrl+F4>. El resultado será la aparición de la ventana de la figura 4.10.

Figura 4.10. *Ventana destinada al control remoto de la Fresadora.*

Su funcionamiento y descripción es similar al módulo de control remoto del torno, por lo que todos los aspectos comentados en el apartado 4.4.1. tienen validez ahora.

4.5. CONTROL REMOTO DEL ROBOT MITSUBISHI.

Siempre que necesitemos realizar alguna operación sobre el robot Mitsubishi deberemos seleccionar la opción que con el mismo nombre se encuentra en el menú *Control Remoto* de la Ventana Principal. Esta opción tiene asociado el conjunto de teclas rápidas <Ctrl+F5>. A continuación, el programa se encarga de abrir y configurar el puerto seleccionado para el robot. Si durante este proceso se produce algún fallo, el programa muestra el mensaje de la figura 4.11., en el que se enumeran algunas de las posibles causas que han podido generar dicho error.

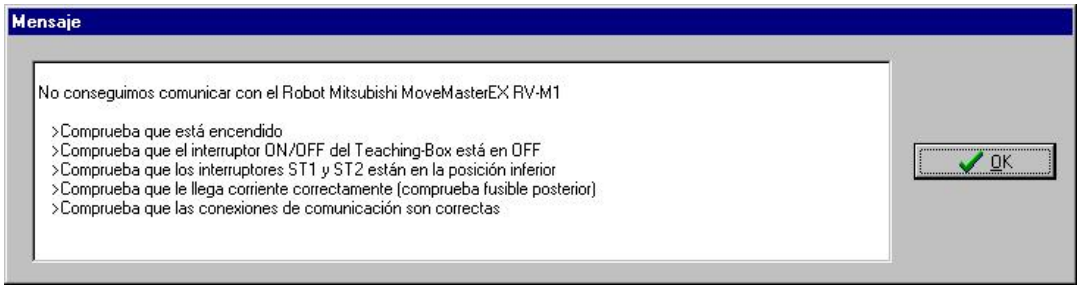


Figura 4.11. *Mensaje con las posibles causas del fallo en la comunicación host↔robot..*

Generalmente, el fallo en la conexión entre el robot y el ordenador central se debe a que el interruptor ON/OFF del Teaching Box está en ON o bien porque el interruptor ST1 del panel lateral del controlador está en su posición superior.

Si durante la apertura y configuración del puerto no se produce ningún error, el programa muestra un mensaje en el que se nos da la posibilidad de realizar el *Home* (figura 4.12.).

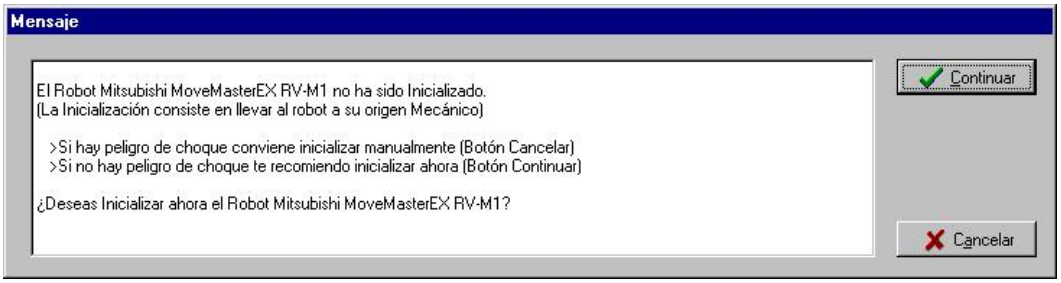


Figura 4.12. *Mensaje Home.*

Tal y como se describió en el capítulo anterior, nada más encender el controlador del robot debemos llevarle a orígenes. Si esta operación puede ocasionar una colisión con algún otro equipo circundante (bien sea el torno, la fresadora,...) el programa nos da la posibilidad de realizar el *Home* manualmente seleccionando la opción *Cancelar*. De no ser así, es decir, si no existe riesgo de colisión, presionaremos el botón *Aceptar*, con lo que será el propio programa el encargado de llevar al robot a orígenes.

En cualquier caso, la siguiente ventana que surge es la mostrada en la figura 4.13.

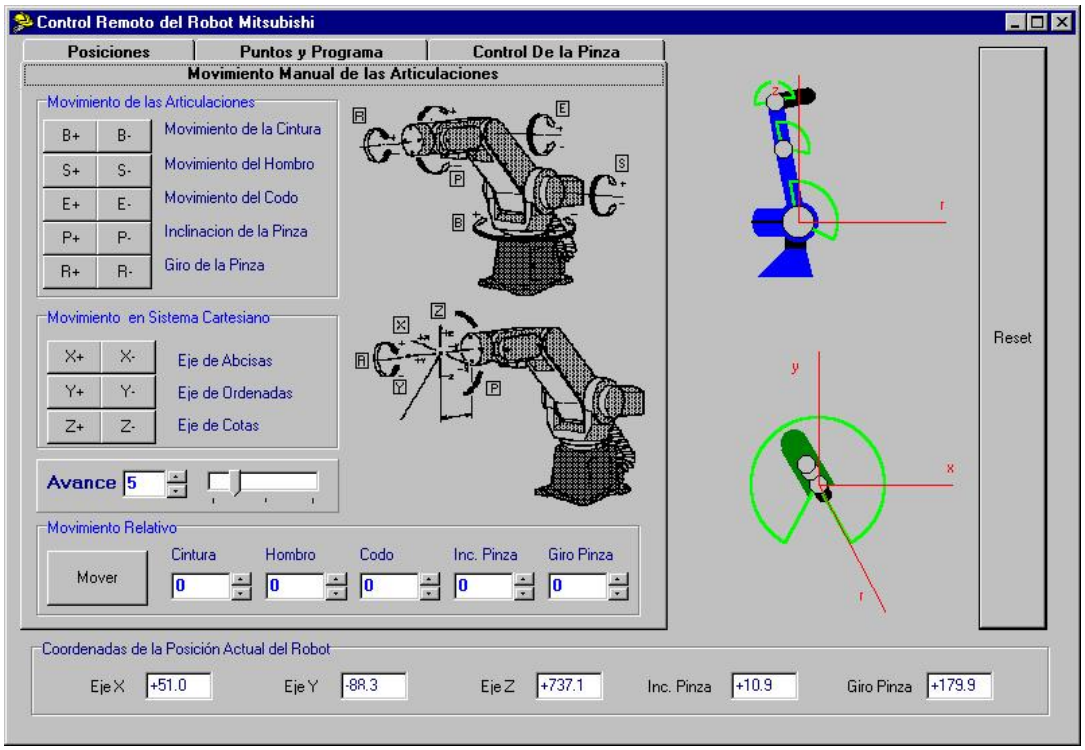


Figura 4.13. *Aspecto inicial de la ventana encargada del control remoto del Mitsubhishi.*

El módulo *Control Remoto del Robot Mitsubixhi MovemasterEX RV-M1* se divide en tres zonas importantes:

En la **parte inferior** se muestra en todo momento las coordenadas cartesianas del extremo del robot (Coordenadas X,Y,Z) expresadas en milímetros, así como los valores P (pitch wrist) y R (roll wrist), expresados en grados, que proporcionan tanto la inclinación como el giro de la mano del robot.

En la **parte derecha** de la ventana podemos observar las vistas, lateral y superior, del robot. Estas imágenes simulan el comportamiento del robot y nos muestran las regiones en las que sus brazos pueden desplazarse. Junto a estos dibujos está el botón RESET. Su función es la misma que la del botón con este mismo nombre que está situado en la parte frontal del controlador (capítulo 3), y dado que su uso es frecuente, se ha optado por hacerle bien visible.

En la **parte superior izquierda** encontramos el bloque principal del programa. Se trata de un componente *TPageControl* que nos permite acceder a cada una de las posibles opciones de este módulo. Conozcámoslas.

4.5.1. Movimiento manual de las articulaciones.

Esta es la página que se muestra nada más acceder a este módulo de programa. En ella se encuentran las siguientes opciones:

- **Movimiento de las Articulaciones.** Nos permite mover cada uno de los brazos del robot por separado. Manteniendo el botón pulsado, la parte involucrada en el desplazamiento se moverá continuamente hasta que encuentre un tope (se produce un Error del Tipo II) o hasta que soltemos dicho botón. Se ha intentado que los nombres que figuran en cada botón sean los mismos que aparecen en el Teaching Box. Así, los botones B+ y B- hacen que la base del robot gire en un sentido o en otro, provocando una variación del ángulo 'A' de la figura 4.14. Los botones S+ y S- causan una variación en la inclinación del hombro (Shoulder), es decir, producen un cambio del ángulo 'B'. Los botones E+ y E- originan una variación en el estado del codo del robot, o lo que es lo mismo, una variación del ángulo 'C'. Los botones P+ y P- originan una variación en la inclinación, en uno o en otro sentido, de la mano del robot, lo que plasmado en la figura 4.14. supone una modificación del ángulo 'D'. Por último, los botones R+ y R- ocasionan un giro de dicha mano. Este giro no afecta a las coordenadas X, Y, Z del extremo del robot, pues no implica desplazamiento alguno.

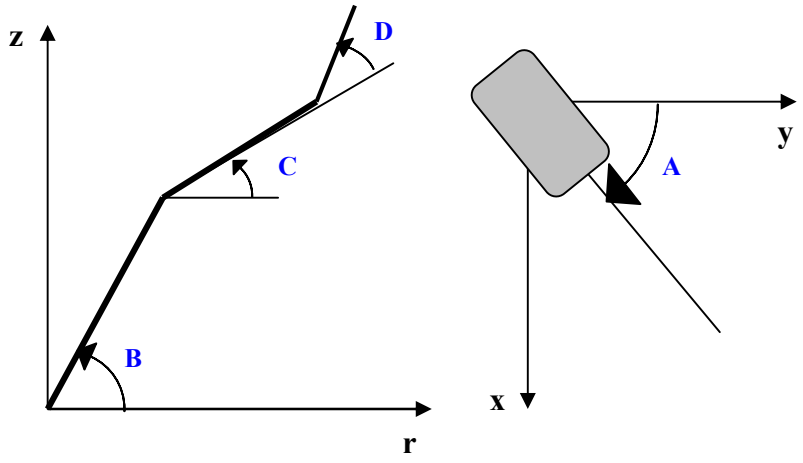


Figura 4.14. *Interpretación de los valores A,B,C y D.*

- **Movimiento en el Sistema Cartesiano.** Esta opción nos permite desplazar el extremo del robot en la dirección de los ejes coordenados. En este caso, el desplazamiento implica un movimiento múltiple de los brazos del robot. Si presionamos los botones X+ o X-, estaremos indicando al robot que se debe mover según la dirección X (sentido positivo o negativo respectivamente), manteniendo las coordenadas Y, Z constantes. Algo similar ocurre si pulsamos los botones Y+, Y-, Z+, Z-.

- **Avance.** Todos los botones anteriores están relacionados con este panel. En él se especifica el valor (en grados o en milímetros) que queremos que el robot se mueva en cada pulsación de los botones de movimiento. En este momento debemos aclarar un aspecto importante: aunque los botones de movimiento están preparados para ejecutarse mientras estén pulsados, el desplazamiento del robot no es del todo continuo, es decir, el host envía al robot la orden de desplazarse o girar un determinado valor dado por el incremento. Cuando el robot se ha desplazado, de forma continua, hasta la posición especificada, el ordenador central se percata de esa situación y si en ese momento el botón permanece presionado el ordenador central envía al controlador

del robot la misma señal ocasionando un nuevo desplazamiento del robot. En caso de que el botón no se encuentre presionado, el ordenador no envía ninguna señal y el robot permanece inmóvil. Podemos modificar el valor del incremento una pequeña cantidad usando el control *TUpDown* situado al lado del comando de edición que nos muestra el incremento que se está usando actualmente, mientras que si nuestra intención es modificarlo notablemente, disponemos de un control *TTrackBar*. Ambos controles están interrelacionados, por lo que una variación en uno de ellos origina un cambio en el otro.

- Finalmente, disponemos de la opción **Movimiento Relativo.** Esta opción nos permite introducir el incremento que deseamos, positivo o negativo, para cada uno de los brazos del robot, es decir podemos desplazar, de forma relativa, todos los brazos que deseemos y la cantidad que deseemos. Para conseguir dicho movimiento tan solo debemos presionar el botón *Mover* después de seleccionar los incrementos deseados.

4.5.2. Posiciones.

En este panel, que se puede ver en la figura 4.15., el primer botón que encontramos nos permite ordenar al robot que se desplace hasta la posición de *Home* o posición de *Origen Mecánico*. Esta posición viene definida de fábrica, es decir, no podemos modificarla. Además, el *Origen Mecánico*, o *Home*, es la posición a partir de la cual se toman las referencias del resto de las posiciones que nosotros podamos definir con el ordenador central o con el teaching box.

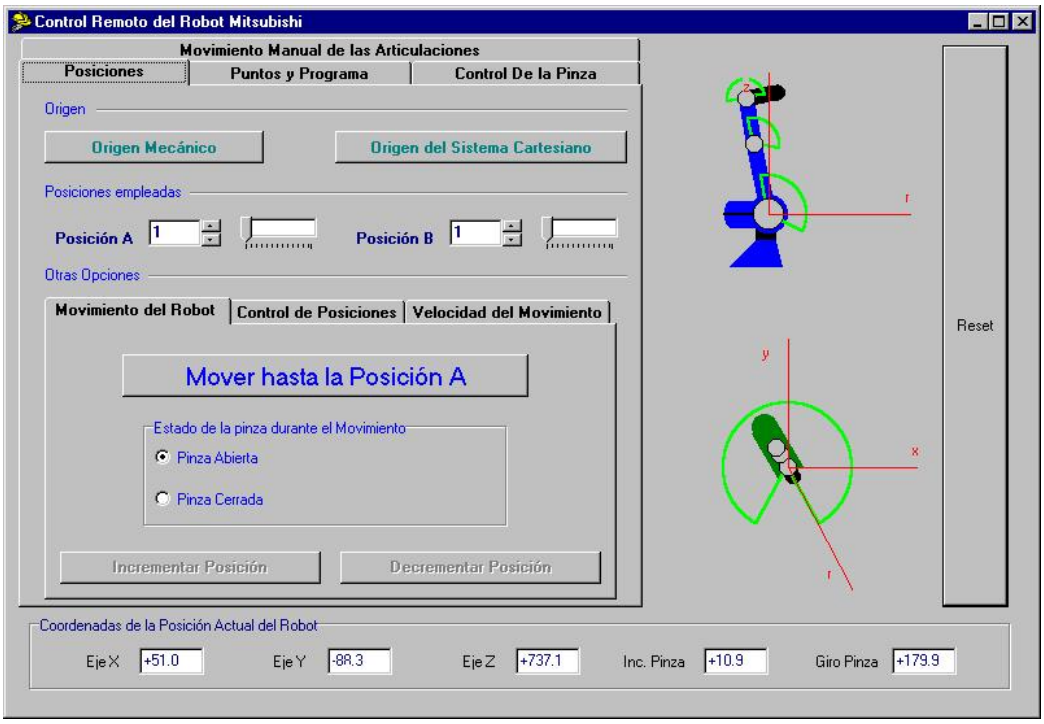


Figura 4.15. *Página Posiciones.*

Junto al botón de *Home* encontramos el control *Origen del Sistema Cartesiano*, que también viene definido de fábrica. En esta posición, los ángulos A, B, C y D que vimos en la figura 4.14. valen cero, al igual que la coordenada X. Y sería la longitud del robot en su posición estirada, y la coordenada Z valdría 300, que es la distancia (en milímetros) que el robot está elevado respecto al cero absoluto.

Inmediatamente por debajo de estos dos botones encontramos la posibilidad de definir dos puntos o posiciones: A y B. Dado que el robot puede almacenar hasta 629 posiciones[2], es necesario el uso de controles *TTrackBar* que nos permitan un acceso más rápido a todas y cada una de las mismas. Pero, ¿Por qué se definen dos posiciones?. La mayor parte de las operaciones que vamos a realizar en este panel, son operaciones que requieren definir previamente una posición: ir a la posición 13, borrar la posición 56, etc. Para estas funciones se define la Posición A. Sin embargo, existe otro tipo de funciones que necesitan la definición de dos posiciones (copiar las coordenadas de la posición 3 en la

[2] El controlador del robot permite almacenar hasta 629 posicones y hasta 2048 líneas de programa.

posición 10, etc.), y por lo tanto, necesitarán utilizar tanto la posición A como la posición B.

Cabe resaltar que cualquier cambio en la posición A afecta a la posición B, aunque lo contrario no es cierto, es decir, cuando establezco la posición A en 30, la posición B también toma el valor 30, pero si cambio la posición B a 31, A no se verá afectada y continuará valiendo 30. Este comportamiento se basa en el hecho de que, en la mayor parte de los casos, las posiciones necesarias son muy cercanas.

Tras definir una posición podemos realizar cualquiera de las operaciones que se nos muestran a continuación.

Movimiento del robot.

Aquí encontramos las opciones básicas relacionadas con el movimiento del robot a unas posiciones establecidas de antemano:

- **Movimiento hasta una Posición.** Seleccionando esta opción obligaremos al robot a desplazarse hasta la posición definida en Posición A. En caso de que esta posición no haya sido definida previamente (guardada en la memoria del controlador), o bien el robot ya se encuentre en esa posición, el robot no se desplazará. En caso contrario, se producirá el movimiento de los brazos del robot hasta la posición deseada. Este movimiento podrá realizarse bien con la pinza abierta o bien con la pinza cerrada, según lo indiquemos.

- **Movimiento hasta la Posición Inmediatamente Superior.** Presionando el botón *Incrementar Posición* el robot se desplaza hasta la posición inmediatamente superior a la que se encuentra actualmente. Obviamente, si dicha posición no está definida no existirá desplazamiento.

- **Movimiento hasta la Posición Inmediatamente Inferior (Decrementar Posición).** Presionando este botón el robot se desplaza

hasta la posición inmediatamente inferior a la que se encuentra actualmente. Igual que antes, si dicha posición no está definida no existirá desplazamiento.

Dado que los botones *Incrementar Posición* y *Decrementar Posición* requieren habernos desplazado previamente hasta una posición cualquiera, van a permanecer inactivos hasta que no hayamos pulsado, al menos una vez, el botón **Movimiento hasta la Posición A**.

Control de posiciones.

- **Establecer Posición Actual como Posición #A**. Esta opción nos permite guardar, en la controladora del robot, la definición de la posición en la que se encuentra actualmente el brazo del robot.
- **Borrar las Posiciones Comprendidas entre las Posiciones #A y #B.**
- **Mostrar las Coordenadas de la Posición #A.**
- **Copia en la Posición #B las coordenadas de la posición #A.**

Para poder llevar a cabo alguna de las operaciones anteriores, tan sólo deberemos seleccionar el botón de radio correspondiente y presionar el botón *Ejecutar Operación*.

Velocidad de movimiento.

Esta es la página en la que estableceremos los datos relativos a la velocidad del movimiento. En concreto definiremos:

- *La velocidad de desplazamiento*. Es un valor comprendido entre 1 y 9.
- *El tiempo de Aceleración/Deceleración*. Podemos seleccionar una aceleración/deceleración rápida (botón de radio H), o una aceleración lenta (botón de radio L).

4.5.3. Puntos y programas.

En este panel tenemos las opciones que nos permiten tanto enviar y recibir el programa del controlador, como enviar y recibir el fichero con la definición de puntos o posiciones que dicho programa pueda necesitar.

Enviar.

Antes de proceder a enviar cualquier fichero a la memoria del controlador, deberemos borrar dicha memoria. Por este motivo, hasta que no presionemos el botón *Borrar la Memoria del Robot* el botón *Enviar* permanecerá inactivo.

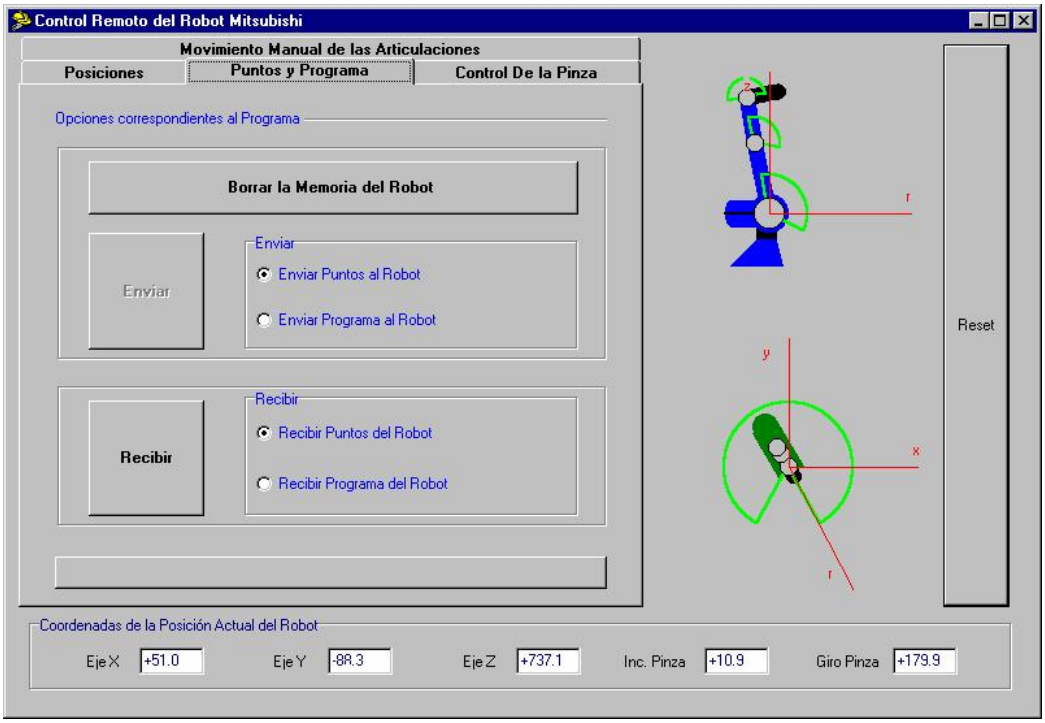


Figura 4.15. *Página Posiciones.*

Tras borrar la memoria del controlador, el siguiente paso es seleccionar la información que deseamos enviar: el archivo que contiene el programa que se deberá ejecutar, o el archivo que contiene la definición de los puntos. En la mayor parte de los casos, los programas necesitan que el controlador tenga almacenadas las definiciones de algunos puntos. Esto es lo que ocurre, por ejemplo, con el

programa *R-4Pos.pgm*[3], que necesita la definición de puntos almacenados en el archivo *R-4Pos.crd*. Continuando con la explicación, si seleccionamos la opción *Enviar Puntos al Robot* y presionamos el botón *Enviar*, aparecerá un cuadro de diálogo que nos permitirá seleccionar el archivo de puntos que queremos abrir. En principio, este cuadro de diálogo mostrará los archivos de puntos situados en el directorio "C:\Celula00_01\RobotMit\" (extensión CRD), y en concreto, aparecerá seleccionado el archivo *R-4Pos.crd*, ya que es el archivo que más necesitaremos. Dado que no deseamos seleccionar otro archivo, sólo nos restará presionar el botón Aceptar.

Inmediatamente después, el cuadro de diálogo desaparece, y al mismo tiempo surge en la parte inferior una barra en al que se muestra la información que estamos transmitiendo.

Si la transmisión de datos al robot se ha producido de forma correcta aparecerá un mensaje indicándonoslo, y la barra desaparecerá.

Acto seguido, seleccionaremos la opción *Enviar Programa al Robot* y presionaremos el botón *Enviar*. Al igual que ocurría anteriormente, aparecerá un cuadro de diálogo que nos permitirá seleccionar el programa que queremos enviar. En principio, este cuadro de diálogo mostrará los programas (extensión PGM) situados en el directorio "C:\Celula00_01\RobotMit\", y en concreto, aparecerá seleccionado el archivo R-4Pos.pgm, ya que es el programa que más utilizamos. Dado que no deseamos seleccionar otro archivo, sólo nos restará presionar el botón Aceptar. Inmediatamente después, el cuadro de diálogo desaparece y en la parte inferior surge una barra en la que se mostrará la información que estamos transmitiendo. Al igual que antes, si la transmisión de datos al robot se ha producido de forma correcta aparecerá un mensaje indicándonoslo, y la barra desaparecerá.

[3] R-4Pos.pgm es el programa que necesitaremos cuando la célula funcione en su conjunto.

Recibir.

Si por cualquier circunstancia deseamos conocer las coordenadas de los puntos almacenados en el robot o recibir el programa que actualmente se está ejecutando, debemos recurrir a la parte inferior del panel. El proceso de recepción de datos es muy similar al proceso seguido para enviar archivos, solo que ahora:

- No deberemos borrar la memoria del controlador.
- El cuadro de diálogo que nos permite guardar los datos recibidos en un archivo no aparecerá hasta haber recibido todos los datos.
- Inicialmente, en el cuadro de diálogo no aparece ningún archivo seleccionado.

4.5.4. Control de la pinza.

La página que se representa en la figura 4.16. permite abrir y cerrar la pinza, así como establecer las fuerzas que gobernarán el agarre

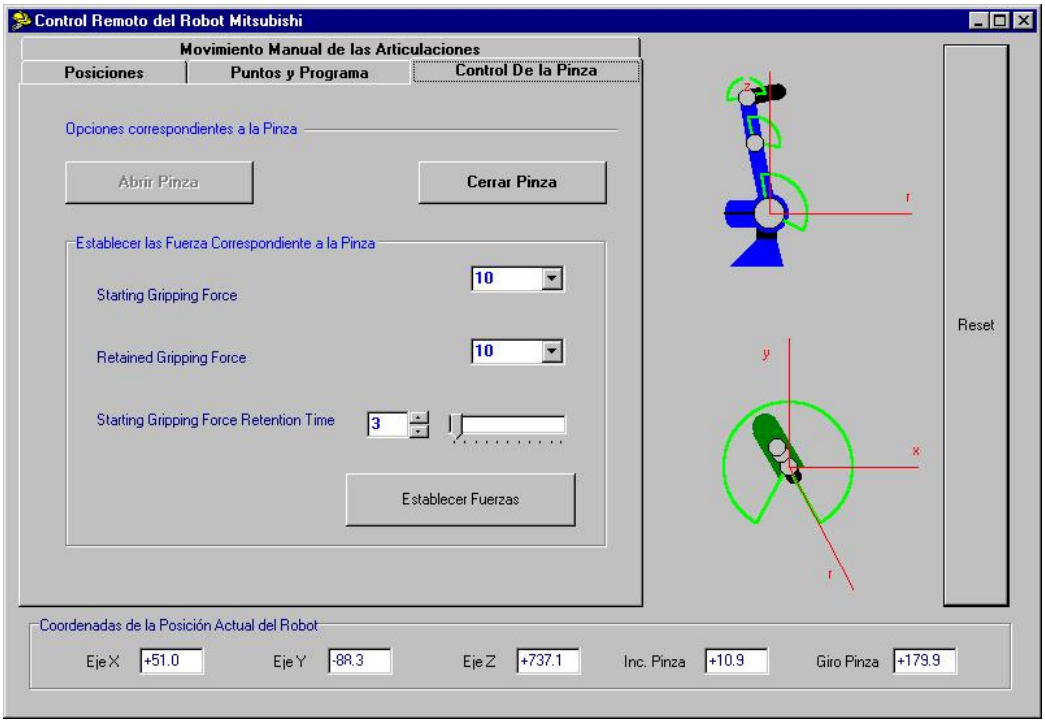


Figura 4.16. *Página Control de la Pinza.*

Tanto la fuerza de agarre inicial como la sostenida pueden tomar un valor comprendido entre 0 (mínimo) y 15 (máximo). El tiempo de retención de la fuerza de agarre inicial toma un valor entre 0 y 99, aunque el tiempo real vendrá dado por la décima parte del valor seleccionado. Inicialmente, tras encender el controlador, estos parámetros toman los valores 10,10 y 3 respectivamente, que son los valores que nos aparecerán al mostrar por primera vez esta página. Estos valores permanecerán como válidos hasta que les modifiquemos.

4.6. CONTROL REMOTO DEL SLIDE.

Para poder acceder a dicho módulo, deberemos elegir la opción *Slide* del menú *Control Remoto,* o bien pulsar la combinación de teclas <Ctrl+F7>. Tal y como ocurría en los casos anteriores, el programa se encarga de abrir y configurar el puerto asignado al slide. Si durante este proceso ocurre algún error, el programa muestra un mensaje en el que se indican algunas de las posibles causas que han podido generarlo.

Si durante la apertura y configuración del puerto no se produce ningún fallo, el programa nos muestra un mensaje en el que se nos indica que se va a proceder a enviar el programa de movimiento del slide y, a continuación, ejecutar una inicialización o *Home*. Dado que el robot Mitsubishi se encuentra montado sobre el slide, deberemos tener cuidado ante una posible colisión del mismo con otros equipos periféricos.

El programa de movimiento que se envía en este momento, y que nos va a permitir un control remoto sobre el slide, no es el mismo que el programa que está permanentemente en la memoria del controlador (o *Caja Negra)* del slide. En concreto, el programa que utilizamos para el control remoto se denomina ControlRemoto.prg, mientras que el programa de ejecución es, precisamente, Ejecución.prg.

Tras realizar el *Home* aparece la ventana *Control Remoto del Slide* que puede verse en la figura 4.17.

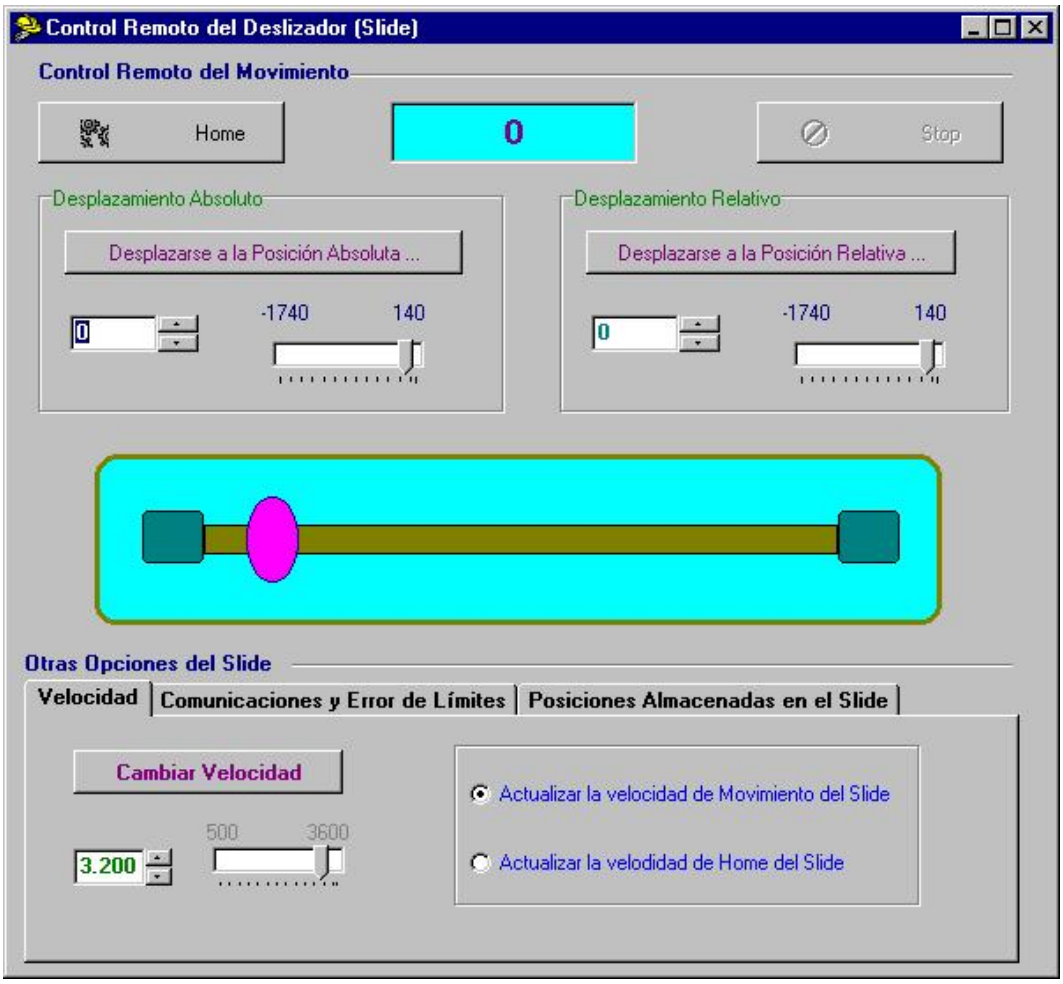


Figura 4.17. *Ventana del control remoto del Slide.*

En la zona superior se encuentran las opciones que me van a permitir desplazarme, tanto en coordenadas relativas como absolutas, realizar el *Home* o parar cualquier movimiento previamente iniciado.

- **Desplazamiento Absoluto.** El desplazamiento del slide se lleva a cabo en términos absolutos, es decir, si queremos desplazarnos a la posición 500, en el cuadro de edición correspondiente deberá aparecer el valor 500. Este desplazamiento puede darse entre los puntos –1740 y 140. Dado el amplio margen de valores comprendidos entre estos dos

números, se ha creído conveniente ubicar un control *TTrackBar* que nos permita una rápida selección. El campo de edición, al que está unido un control *TUpDown* que nos permite una ubicación del slide más selectiva, nos muestra la posición a la que queremos desplazar el robot. Para iniciar el movimiento tan solo deberemos hacer clic sobre el botón *Desplazarse a la Posición Absoluta #*.

- **Desplazamiento Relativo.** El desplazamiento ahora se lleva a cabo en términos relativos, es decir, si queremos desplazar el robot 50 unidades 'por encima' del punto actual, en el cuadro de edición correspondiente al movimiento relativo deberá aparecer el valor 50. Al igual que le ocurre al movimiento absoluto, el movimiento relativo está restringido entre unos valores, que, por supuesto, no van a ser constantes. El programa, en todo momento, nos irá mostrando el rango de valores permitidos encima del control *TTrackBar* correspondiente. Por supuesto, para iniciar el desplazamiento deberemos hacer clic sobre el botón *Desplazarse a la posición Relativa #*.

- ***Home.*** Este botón me permite llevar al eje de deslizamiento a su posición de origen mecánico. Este es el punto a partir de la cual se definen el resto de las posiciones, es decir, se trata de la posición 0.

- **Stop.** Este control permanece inactivo hasta que se envía al slide una orden de movimiento, momento en el que se activa. Mientras está activo, si presionamos sobre él conseguiremos detener el desplazamiento del eje de deslizamiento. En definitiva, se trata del botón de parada.

La posición actual del slide se muestra en el cuadro de edición que se encuentra entre los botones *Home* y *Stop*. Además podemos ver gráficamente la posición que ocupa el robot Mitsubishi sobre el eje de deslizamiento en la imagen gráfica de la parte intermedia de la ventana. No obstante, esta representación gráfica no se limita tan sólo a mostrarnos la posición actual del robot sobre el slide, sino que también nos permite ordenar su desplazamiento. Para conseguir esto, deberemos presionar el botón izquierdo del ratón sobre el óvalo que

representa al robot Mitsubishi y mantenerlo pulsado mientras lo desplazamos. Podremos desplazarnos a cualquier punto del eje de deslizamiento, pero ¿cómo sabemos a donde nos estamos desplazando?. En el cuadro de edición correspondiente al movimiento absoluto, se va a mostrar en todo instante, la posición a la que nos moveríamos si soltásemos el dedo del ratón en ese momento. De esta forma, si soltamos el ratón, el programa ordena un desplazamiento inmediato a la posición deseada.

En al parte inferior encontramos varias opciones complementarias incluidas en un control *TPageControl*:

- **Velocidad.** Me permite modificar la velocidad de desplazamiento, tanto la del desplazamiento normal, como la del desplazamiento que se produce al llevar al slide a su posición de origen mecánico. El valor de la velocidad puede variar entre 500, velocidad mínima, y 3600, velocidad máxima. Al iniciar una sesión con el slide la velicidad establecida por defecto para ambos tipos de deslizamientos es de 3200 (es la velocidad que establece el programa de movimiento). Al presionar el botón *Cambiar Velocidad* modificamos el tipo de velocidad seleccionada.
- **Comunicaciones y Error de Límites.** En ocasiones, la comunicación ordenador central ↔ slide se interrumpe por un fallo, impidiéndonos ejecutar cualquier acción. Para solucionar este defecto tenemos el botón *Reanudar Comunicación*. En otras ocasiones, se puede producir un error en la definición de los límites del eje de deslizamiento. Podremos solucionar este error mediante *Eliminar un Error de Límites*.
- **Posiciones del slide.** El programa Ejecución.prg define 4 posiciones sobre el eje de deslizamiento, posiciones en las que el robot Mitsubishi realiza diversos movimientos. Por este motivo, necesitamos una herramienta que nos permita acceder a estas posiciones mientras se está ejecutando el programa de control remoto. La página *Posiciones del Slide* es la herramienta que buscamos.

4.7. CONTROL REMOTO DEL ROBOT SCORBOT.

Siempre que necesitemos realizar alguna operación sobre el robot Scorbot deberemos seleccionar la opción que con el mismo nombre aparece en el menú *Control Remoto* de la Ventana Principal. Esta opción tiene asociado el conjunto de teclas rápidas <Ctrl+F6>. A continuación, el programa se encarga de abrir y configurar el puerto seleccionado para el robot. Si durante este proceso se produce algún fallo, el programa muestra un mensaje en el que se enumeran algunas de las posibles causas que han podido generar dicho error.

Si durante la apertura y configuración del puerto no se produce ningún error, el programa muestra un mensaje en el que se nos da la posibilidad de realizar el *Home*.

Tal y como le ocurría al robot Mitsubishi, nada más encender el controlador del robot debemos llevarle a orígenes. Si esta operación puede ocasionar una colisión con algún otro equipo circundante (centro de inspección, cinta transportadora,...) el programa nos da la posibilidad de realizar la inicialización manualmente seleccionando la opción *Cancelar*. Si no existe riesgo de colisión, presionaremos el botón *Aceptar*, con lo que será el propio programa el encargado de llevar al robot a orígenes.

En cualquier caso, la siguiente ventana que surge es la mostrada en la figura 4.18.

Figura 4.18. *Aspecto inicial del control remoto del robot Scorbot.*

Dado que los controles y funciones son muy similares a las que se utilizaron para el control del robot Mitsubishi, en este apartado trataremos el módulo encargado del control remoto del robot Scorbot de una forma más breve.

En la página *Control Manual* que aparece en la ventana de la figura 4.18. encontramos:

- Los diferentes botones que nos permiten mover al robot de forma articular o cartesiana.

- Los controles que definirán el avance de cada desplazamiento.
- Los elementos que permiten activar o desactivar el control sobre los ejes del robot.
- Los botones con los que controlar la apertura y cierre de la pinza incorporada en el robot.

Por otro lado, en la página *Posiciones* (véase figura 4.19.) podemos encontrar, entre otras cosas:

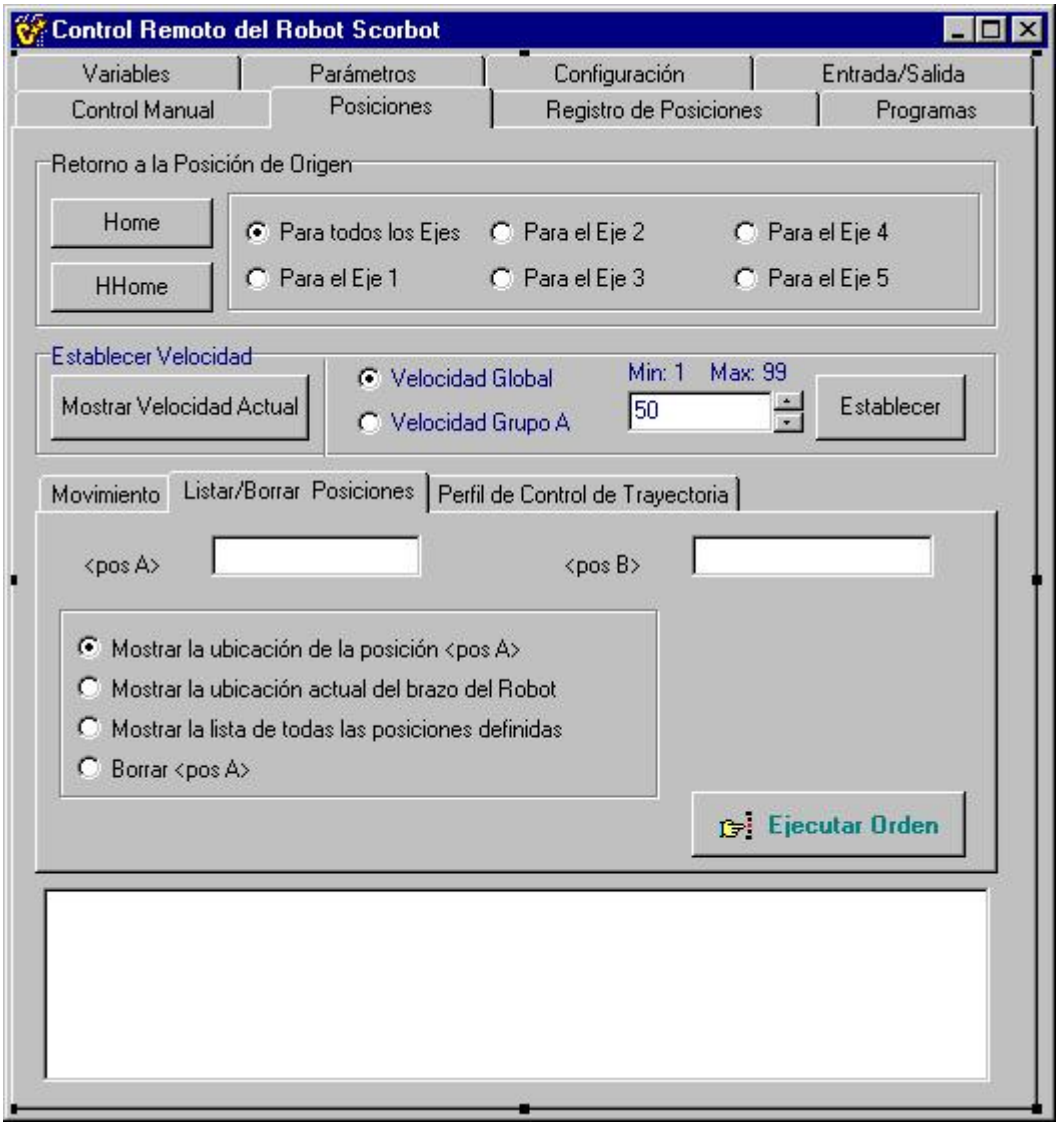


Figura 4.19. *Aspecto de la página Posiciones*

- Los botones encargados de la inicialización de los ejes del robot. Cuando presionamos las teclas *Home* o *HHome* lo que hacemos es ejecutar un programa que está grabado de forma permanente en la EPROM del Scorbot.
    - *Home*. Lleva a los ejes del Scorbot a su posición de origen buscando el microinterruptor de cada eje.
    - *HHome*. Lleva a los ejes a una posición de origen pero buscando un tope rígido en lugar de un microinterruptor.

Podemos realizar la inicialización de un solo eje o la de todos los ejes al mismo tiempo.

- Por debajo de los botones *Home* y *HHome* vamos a encontrar los elemento encargados de establecer la velocidad del movimiento. Dicha velocidad se especifica en porcentajes, siendo el valor 50 la velocidad establecida por defecto.
- Por último, un control *TPageControl* nos permite acceder a las siguientes opciones.
    - *Definir el Perfil de Control de la Trayectoria*. Podemos elegir entre un perfil paraboloide o uno trapezoidal. (a) cuando elijo el perfil trapezoide lo ejes aceleran y deceleran rápidamente al comienzo y final del movimiento, con una velocidad constante durante la trayectoria.(b) cuando elijo un perfil paraboloide, los ejes aceleran lentamente hasta alcanzar la velocidad máxima, decelerando después al mismo tiempo.
    - *Listar/Borrar Posiciones*. Podemos conocer la ubicación de una posición determinada o la posición actual del brazo robotizado. También podemos mostrar la lista con todas las posiciones definidas o borrar una en concreto.
    - *Movimiento*. Gracias a esta ventana podemos ordenar el desplazamiento del robot a una posición determinada. Este desplazamiento también puede realizarse bien a través de una trayectoria lineal, o bien mediante de una trayectoria circular que pasa por un segundo punto definido por el usuario.

En la figura 4.20. se muestra la apariencia de la página *Registro de Posiciones*.

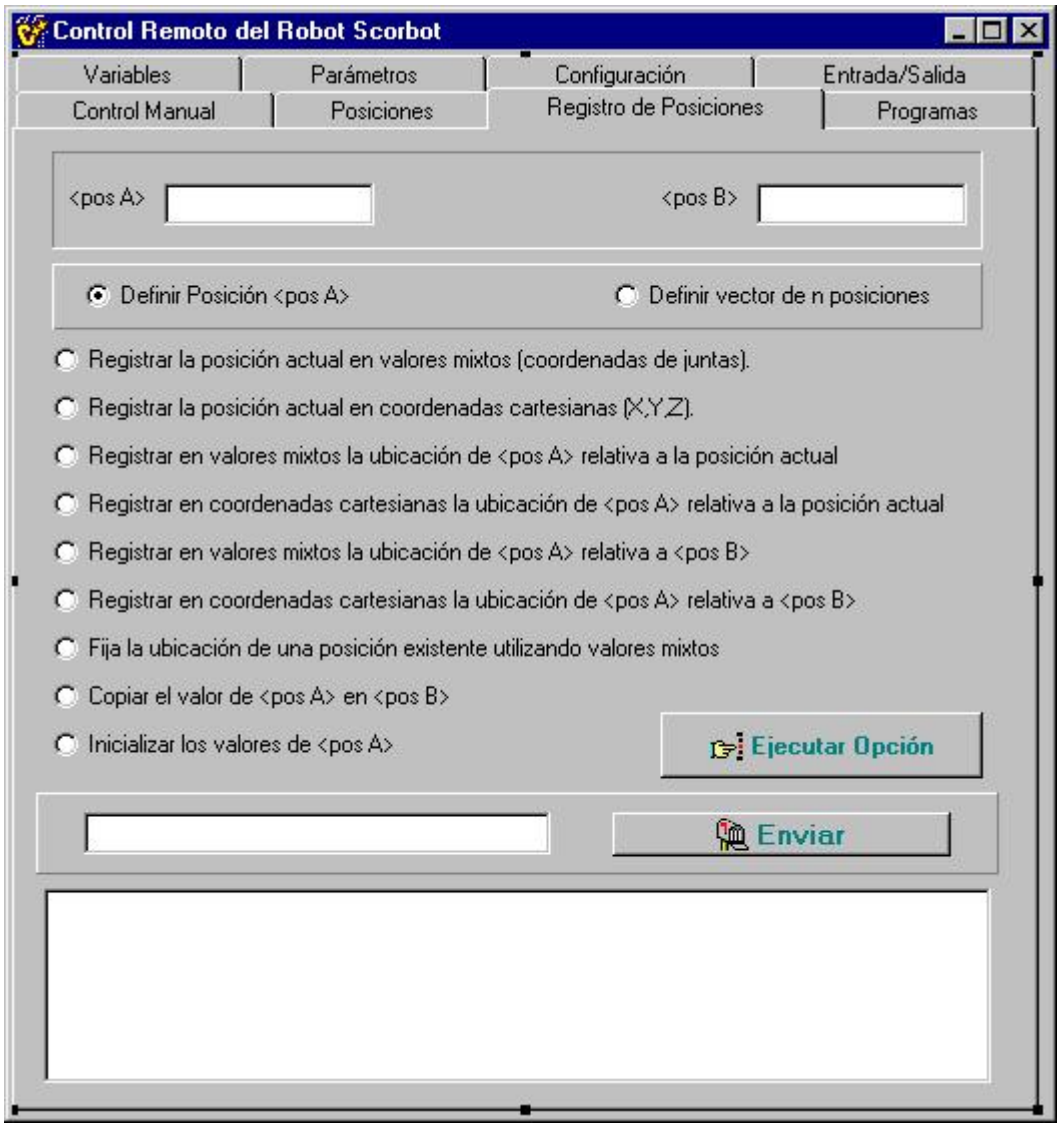


Figura 4.20. *Aspecto de la página Registro de Posiciones*

En esta página vamos a encontrar todos aquellas opciones que nos van a permitir definir y registrar una posición. En concreto:

- Vamos a poder definir una posición o un vector de posiciones. Esta operación es necesaria antes de poder registrar cualquier posición o vector.

- Vamos a poder registrar una posición en valores mixtos (coordenadas de juntas) o en coordenadas cartesianas.
- Vamos a poder copiar los datos de una posición en otra.
- Vamos a poder inicializar los valores de una determinada posición.

Por lo que ser refiere a la página *Programas* (figura 4.21.), descubrimos una serie de elementos que nos van a permitir:

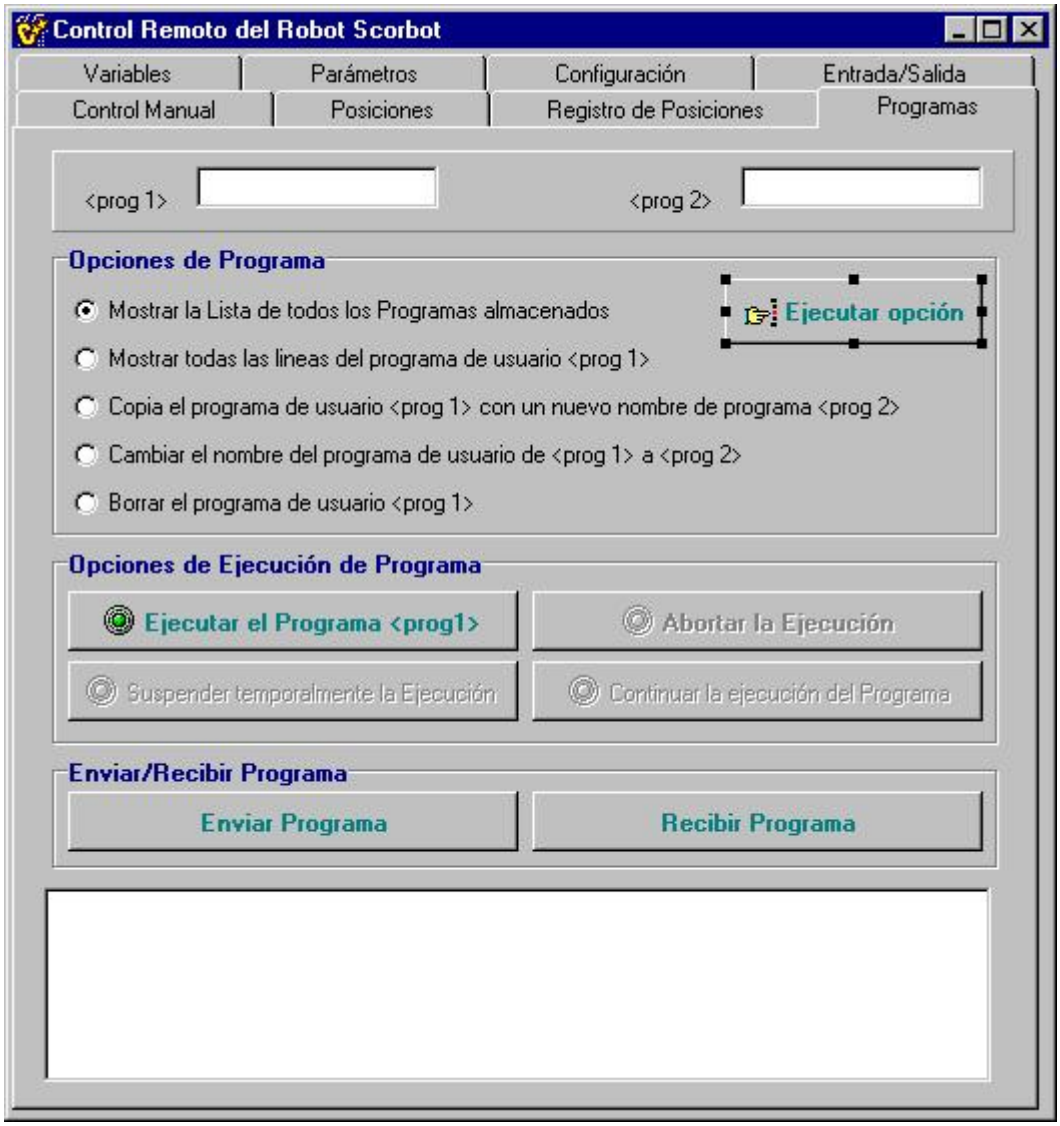


Figura 4.21. *Aspecto de la página Programas.*

a. *Opciones de Programa*.

- Podemos ver el listado de los programas almacenados en la memoria del controlador.
- Podemos mostrar las líneas de un determinado programa.
- Podemos copiar las líneas de un programa en otro.
- Podemos cambiar el nombre del programa.
- Podemos borrar un programa dado.

b. *Opciones de Ejecución de Programa*.

- *Ejecutar un Programa*. Inicia la ejecución del programa.
- *Abortar un Programa*. Finaliza definitivamente la ejecución del programa que estaba ejecutándose en ese momento.
- *Suspender Temporalmente la Ejecución*. Detiene duratne el tiempo que desee el usuario la ejecución del programa.
- *Continuar la Ejecución del Programa*. Reinicia el programa que fue suspendido con el botón anterior.

c. Enviar/Recibir Programa. Estos botones nos permiten enviar (y recibir) programas de usuario a la memoria del controlador del Scorbot.

A continuación se describen brevemente el resto de páginas de esta ventana.

- *Parámetros*. Muchas de las funciones del controlador dependen del valor de los parámetros del sistema. Los parámetros del sistema determinan operaciones tales como: servo control, envoltura de trabajo, límites de velocidad, operación de la pinza, cálculos cinemáticos y cartesianos, etc. Todos estos parámetros pueden ser modificados en esta página, aunque no es recomendable, sobre todo si el robot está en movimiento. Una explicación detallada de cada uno de los parámetros

se puede encontrar en el anexo B de la Guía de Referencia del ACL (Lenguaje de Control Avanzado).

- *Configuración*. En esta página se muestra la configuración actual del robot: número de entradas y salidas, número de codificadores, tipo de robot, número total de servo ejes, memoria de usuario, líneas de programa de usuario, puntos de usuario,...

- *Entrada/Salida*. El controlador del Scorbot tiene 16 puertos de entrada y 16 de salida. Puede darse un nombre simbólico a cada uno de esos puertos, pudiendo ser diferente dependiendo de si se encuentra en ON o en OFF.

- *Variables*. En esta página vamos a poder definir las variables que luego se utilizarán en los programas de usuario, pudiendo realizar con ellas las operaciones que queramos.

CAPÍTULO QUINTO:

SUPUESTO PRÁCTICO.

5.1. INTRODUCCIÓN.

En el capítulo anterior pudimos conocer parte del software desarrollado en este proyecto. En concreto, descubrimos los módulos relacionados con el control remoto de las diferentes máquinas que configuran la FMC del LOIP.

Para explicar el funcionamiento del resto del programa *FMC* vamos a imaginarnos que queremos fabricar las piezas de un tablero de ajedrez en la célula que nuestra empresa acaba de adquirir recientemente pero que todavía no ha sido utilizada (esta hipótesis no obliga a definir todo el proceso de definición y ejecución desde el principio).

La FCM del LOIP dispone, como ya sabemos, de:

i. Un ***Centro de Torneado***, modelo Mirac, perteneciente a la casa Denford.

ii. Un ***Centro de Fresado***, modelo Triac-VMC, de la casa Denford.

iii. Un ***Sistema de Almacenamiento y Recuperación de Materiales***, comúnmente denominado almacén.

iv. Un ***Sistema de Transporte Automático*** basado en una cinta transportadora de palés de velocidad fija e invariable.

v. Dos ***Robots Industriales***: un robot Mitsubishi encargado de alimentar a los centros de mecanizado de control numérico, y un robot Scorbot encargado de alimentar al centro de inspección.

vi. Un ***Eje de Deslizamiento*** sobre el que está montado el robot Mitsubishi. Este eje de deslizamiento proporciona un sexto grado de libertad a dicho robot.

vii. Una ***Unidad Compresora*** que nos proporciona el aire comprimido a la presión deseada.

viii. Un ***Autómata Programable (PLC),*** de la empresa Allen-Bradley, encargado del control de las señales de los relés.

ix. Un ***Ordenador Central (Host)*** en el que está insertada una tarjeta de comunicaciones *National Instrument PCI-232/8* que nos permite comunicarnos con los controladores de cada máquina.

Tal y como hemos comentado, el objetivo será la fabricación de un juego de ajedrez (en total 32 piezas). Tendremos dos grupos de 16 piezas cada uno, que estarán formados por ocho peones, dos torres, dos caballos, dos alfiles, un rey y una reina.

En principio, como nos encontramos con la limitación establecida por el almacén de 24 piezas, vamos a dividir el proceso en dos tareas de 16 piezas cada una, es decir, en una primera ejecución fabricaremos las piezas de color negro, y en una segunda ejecución haremos lo correspondiente con las de color blanco. Las características de estas tareas se recogen en las tablas 5.1. y 5.2.

Nombre de la Pieza	Código de Identificación	Cantidad a Fabricar	Breve Descripción
Peón	1001	8	Sólo necesita Torno
Torre	1002	2	Torno y Fresadora
Alfil	1003	2	Torno y Fresadora
Caballo	1004	2	Torno y Fresadora
Rey	1005	1	Torno y Fresadora
Reina	1006	1	Torno y Fresadora

Tabla 5.1. *Tarea1. Primer juego de piezas (equipo negro).*

Nombre de la Pieza	Código de Identificación	Cantidad a Fabricar	Breve Descripción
Peón	1001	8	Sólo necesita Torno
Torre	1002	2	Torno y Fresadora
Alfil	1003	2	Torno y Fresadora
Caballo	1004	2	Torno y Fresadora
Rey	1005	1	Torno y Fresadora
Reina	1006	1	Torno y Fresadora

Tabla 5.2. *Tarea2. Segundo juego de piezas (equipo blanco).*

Analizando las tablas anteriores se puede observar que se van a fabricar un total de seis piezas diferentes, que son comunes tanto para el juego de piezas del equipo blanco como para el juego de color negro.

A continuación vamos a intentar describir el proceso de fabricación de la forma más precisa posible.

5.2. PUESTA EN MARCHA DE LAS MÁQUINAS. MODO COMUNICACIÓN.

En este apartado vamos a describir la secuencia de operaciones que debemos llevar a cabo con el fin de arrancar e inicializar las diferentes máquinas, así como aquellas operaciones destinadas a lograr que puedan comunicarse con el ordenador central, que es lo que denominaremos a partir de ahora "*Estar en Modo Comunicación*". La secuencia correcta de arranque de las máquinas será la siguiente.

5.2.1. Compresor.

Sin duda alguna, la primera máquina que debemos poner en marcha es el compresor de aire. Cuatro de las máquinas de la célula del LOIP utilizan el aire comprimido para diversas tareas:

⇒ El *almacén* necesita aire comprimido para las operaciones que realiza el brazo robotizado: abir y cerrar pinza, entrar y salir de las celdas, girar hacia dentro y hacia fuera, etc.

⇒ El *conveyor* requiere el aire para levantar los pernos de restricción, los pernos posicionadores de palés y los límites de entrevía de cada una de las estaciones.

⇒ El *torno* y la *fresadora* lo precisan en operaciones tales como abrir y cerrar las mordazas del cabezal, abrir y cerrar la puerta de protección, realizar el cambio de herramienta, etc.

El compresor de aire se encuentra ubicado en una pequeña habitación situada a la entrada del LOIP. Su puesta en funcionamiento es bastante sencilla y consta de tres etapas:

i. Como primer paso, debemos conectar la corriente eléctrica.

ii. Posteriormente, presionaremos el botón verde situado en la parte frontal del compresor.

iii. Tras esto, la unidad de compresión empezará a funcionar hasta conseguir que la presión del acumulador alcance los $5{,}5 \div 6$ bares, que es la presión requerida por los centros de mecanizado (almacén y cinta transportadora necesitan una presión inferior).

5.2.2. Fresadora.

Tras encender el compresor y esperar a que el sistema alcance la presión adecuada, podemos proceder a encender la fresadora. Para ello debemos girar el interruptor rojo, situado en la caja negra anexa a la fresadora, hasta la posición 1. La pantalla de la fresadora se enciende y, tras unos segundos, el programa que controla la fresadora hace su aparición.

Una vez dentro del programa de control, es el momento de que la fresadora retorne a su posición de origen mecánico (*Home*). El proceso de inicialización es el siguiente:

- Presionando la tecla *HOME* del teclado de la fresadora el programa cambia a la ventana encargada de la inicialización.
- Es el momento de inicializar cada uno de los ejes de la fresadora, y para ello pulsaremos las teclas *+X (o –X), +Y (o –Y) y +Z (o -Z)*. El orden en el que se pulsen estas teclas es indiferente, pero en cualquier caso deberemos esperar a que el eje correspondiente alcance su origen antes de presionar la tecla siguiente.

Una vez situada la herramienta en orígenes, es el momento de cargar el fichero de desplazamientos de las herramientas (offsets). Las etapas a seguir en este nuevo proceso son las siguientes:

- El primer paso consiste en presionar la tecla *F9*, lo que provoca la aparición de un menú de opciones del que seleccionaremos la alternativa *Load Tool Offsets*.
- Surge un cuadro de diálogo donde podemos introducir el nombre del archivo que contiene los offsets que necesitamos. Sin embargo, si pulsamos la tecla *EOB* accederemos a otro cuadro de diálogo con todos los posibles archivos que contienen los desplazamientos de las herramientas.
- Con los cursores nos situamos sobre el archivo *0001.fao* y presionamos la tecla *EOB*.

La realización de esta etapa es muy importante, ya que los offsets permiten que el cero máquina y el cero pieza coincidan, es decir, los offsets definen el centro de la pieza de manera que coincida con el cero máquina.

El último paso consiste en poner la fresadora en modo comunicación. Al igual que en las operaciones anteriores, vamos a tener que seguir una determinada secuencia de trabajo.

- El primer paso de esta secuencia consiste en pulsar la tecla *F10*.
- En el centro de nuestra pantalla aparece en un menú desplegable del que seleccionaremos, con los cursores, la opción *CNC Files.*
- Un nuevo menú de opciones se exhibe ante nuestros ojos. Esta vez la opción requerida es *Load CNC Files*.
- El siguiente cuadro de diálogo nos permite introducir el nombre del fichero que contiene el código de control numérico que nos hace falta. Como ya sabemos, presionando la tecla *EOB* podemos acceder a una nueva ventana que muestra todos los ficheros de control numérico disponibles.
- Seleccionamos mediante los cursores el fichero *Inicio.fnc* y pulsamos, otra vez, la tecla *EOB*.
- Para salir de los diferentes menús que hemos abierto utilizamos la tecla *Reset*.
- Para finalizar la puesta en modo comunicación sólo nos resta presionar las teclas *Auto* y *Cycle Start*.

De esta forma la fresadora ya se encuentra en disposición de recibir y enviar información.

5.2.3. Torno.

El proceso que debemos seguir para encender y poner al torno en modo comunicación es muy similar al seguido para la fresadora. A continuación se enumeran las diferencias más relevantes con respecto a la fresadora.

- El interruptor que enciende el torno se encuentra situado en la parte posterior de la máquina.
- Como ya sabemos, el torno sólo utiliza dos ejes, X y Z, por lo que cuando realicemos el *Home* del torno sólo tendremos que inicializar dichos ejes.
- El archivo que contiene los offsets a cargar es *0001.mio*.
- Para poner al torno en modo comunicación seleccionaremos el archivo *Inicio.mir*.

5.2.4. Almacén.

Los pasos encaminados a encender el almacén son los siguientes:

i. El primero de todos es encender el transformador situado en la parte inferior derecha del almacén y comprobar que la presión que recibe del compresor es la correcta.

ii. Tras comprobar que le llega corriente eléctrica, es decir, que el piloto rojo del transformador está encendido, y que la presión es la correcta, el siguiente paso consiste en encender el controlador del almacén. Con esta acción el sistema entra directamente en Windows 95.

iii. Debemos introducir entonces el disquete con el programa ALMACEN en la disquetera y seleccionar el icono *Almacén* situado en el escritorio de Windows 95.

iv. Tras unos segundos de espera, surge la ventana de presentación del programa. Presionando cualquier tecla esta ventana desaparecerá dando paso a la ventana que puede verse en la figura 5.1.:

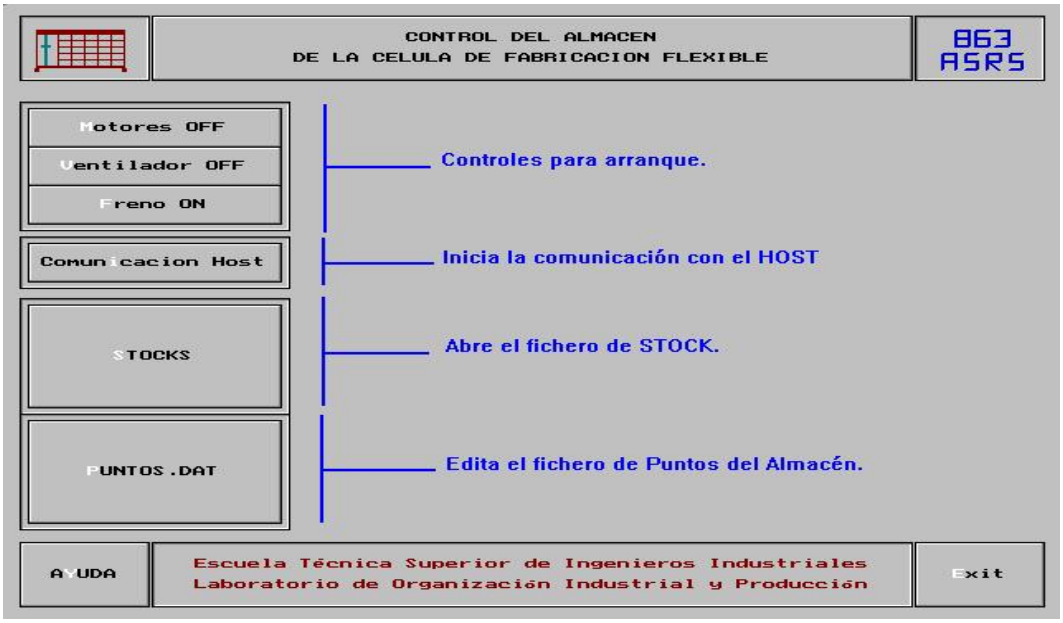


Figura 5.1. *Ventana de inicialización del programa que controla el almacén.*

Debido a la ausencia de ratón, para activar cualquier campo deberemos pulsar en nuestro teclado la letra resaltada en la pantalla.

v. Al iniciar una sesión de trabajo con el almacén tenemos que activar los motores, el ventilador y desactivar el freno. Para conseguirlo, presionaremos las letras m, v, f consecutivamente. Inmediatamente después, surge una ventana similar a la mostrada en la figura 5.1 pero con más opciones.

vi. El paso siguiente consiste en poner el almacén en modo Comunicación. De esta forma podremos establecer una comunicación directa entre el sistema controlador del almacén ASRS y el ordenador central (host). Sin embargo, antes de llevar a cabo esta operación comprobaremos que el brazo robotizado del almacén se encuentra en posiciones negativas de los ejes coordenados. Si no es así, utilizaremos los cursores para desplazar el brazo hasta esas posiciones mencionadas. Tras esta breve

comprobación, presionaremos la tecla i, tras lo cual, el almacén realizará automáticamente un *Home*.

vii. Una vez que el robot ha llegado a la posición de origen mecánico, el programa se encarga de informarnos:

⇒ Si el *Home* se ha realizado correctamente deberemos pulsar la tecla '1'.

⇒ En caso contrario, pulsaremos la tecla '2'. Deberemos, entonces, ubicar manualmente el brazo robotizado en coordenadas negativas antes de volver a intentar poner el almacén en modo comunicación. En definitiva, retornaremos al paso número vi.

viii. Tras unos segundos, el robot cartesiano se desplaza hasta una posición de espera, lo que nos indica que el controlador del almacén se encuentra en disposición de recibir y enviar información al ordenador central.

Pero antes de continuar con la inicialización de otro elemento, ¿qué significa posicionar el robot en coordenadas negativas? y, ¿por qué debemos hacerlo antes de llevar el almacén a su origen mecánico?. Es improtante conocer que antes de realizar un *Home* (posicionar el robot en el origen de coordenadas) se debe comprobar que el brazo robotizado está dentro de las coordenadas X e Y negativas, es decir, en la zona blanca de la figura 5.2.

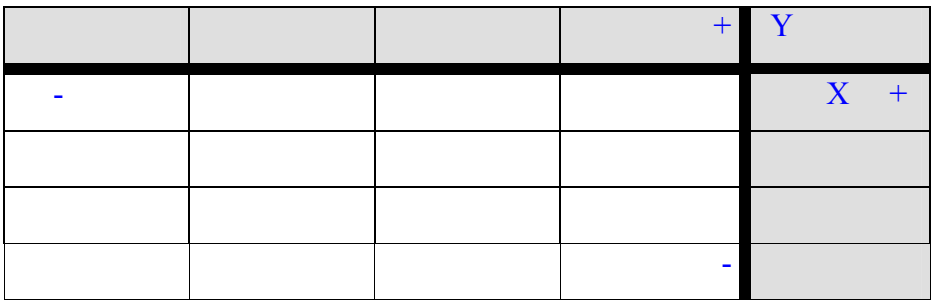


Figura 5.2. *Áreas positivas y negativas del Almacén.*

Cuando el brazo se encuentra situado en las coordenadas positivas del eje "X" o del eje "Y" y se ordena al mismo que se posicione en el origen de coordenadas, el brazo sale sin control y no para hasta que se va fuera de límites. Para solucionar este inconveniente, simplemente debemos ubicar manualmente el brazo en posiciones negativas del eje "X" e "Y" antes de que se ejecute el *Home*. Para poder conseguir esto, utilizaremos los cursores situados en la parte inferior derecha de la pantalla mostrada en la figura 5.3. El botón de las flechas quedará hundido mientras tengamos apretada la flecha del cursor y se levantará cuando lo soltemos. Estos movimientos se realizan a la velocidad indicada en la parte inferior derecha de la ventana.

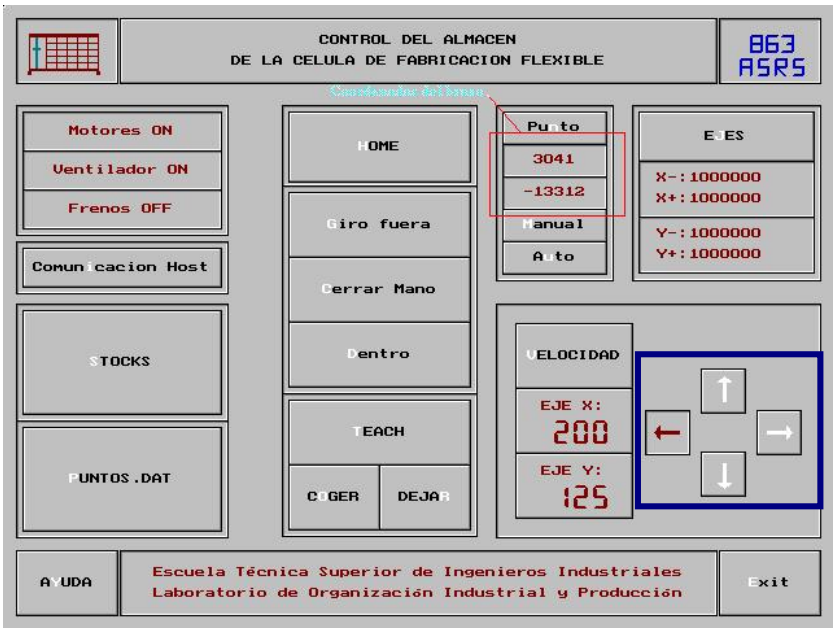


Figura 5.3. *Con las flechas podemos situar el brazo robotizado en coordenadas negativas.*

5.2.5. Autómata Allen-Bradley.

Para conectar el autómata basta con encender el transformador que se encuentra en la parte delantera de la cinta.

⇒ Si existe un programa residente en la memoria del autómata, éste empezará a ejecutarse directamente. Los pernos de restricción, pernos posicionadores de palés y límites de entrevía de cada una de las estaciones del conveyor se levantarán si la última vez que se apagó el autómata estaban accionados.

⇒ Si no existe un programa en la memoria, deberemos acceder a la parte de la aplicación principal encargada del control remoto del autómata y enviar un programa de ejecución a la memoria SVRAM de dicho autómata. Tal y como se recordará, el programa actualmente en uso es *Residente.cim*.

5.2.6. Cinta transportadora.

Encender la cinta transportadora es bien sencillo.

1) La primera medida encaminada a lograr este fin consiste en encender el transformador. Sin embargo, dado que autómata y cinta transportadora comparten el mismo dispositivo, este paso ya fue dado en el apartado anterior.

2) Por lo tanto, para que el motor de la cinta empiece a funcionar, tan sólo debemos presionar el botón de arranque (*Start*) que se encuentra cerca del motor. Este botón de arranque es de color verde para diferenciarlo del botón de parada (*Stop*) que es de color rojo.

5.2.7. Robot Scorbot.

Para poner en funcionamiento el robot Scorbot, primero hay que encender el controlador del robot, y para ello pondremos en ON el interruptor que se encuentra en su lado derecho. Después, para que la corriente llegue a cada uno de los motores que accionan los ejes, pulsaremos el interruptor MOTORES que está situado en el frontal del controlador. Como consecuencia de esta acción, el diodo situado encima de dicho interruptor empieza a lucir en un color verde que nos indica que los motores están conectados.

La siguiente etapa consiste en inicializar el robot y habilitar los ejes:

⇒ Primero debemos comprobar que el robot no colisionará con el centro de calidad durante su inicialización. Si hay riesgo de colisión realizaremos un desplazamiento de los ejes del robot hasta una posición segura. En ocasiones, nada más arrancar el robot, el control está desactivado, lo que nos impide mover los brazos. Para recuperar dicho control tan sólo debemos pulsar la tecla *CON* del Teach Pendant.

⇒ Desde el Mando de Enseñanza (Teach Pendant) presionaremos la secuencia de teclas RUN 0 [Enter]. De esta forma, el robot comenzará su proceso de retorno hasta la posición de origen o *Home*.

⇒ Una vez que el *Home* se ha completado satisfactoriamente, el control se CONECTA automáticamente. Si esto no fuese así, deberemos pulsar nuevamente el botón CON.

Estas etapas no son necesarias si lo único que deseamos es gobernar el robot mediante el programa de control remoto correspondiente: este programa ya se encarga de dar los pasos anteriores previa autorización del usuario.

5.2.8. Robot Mitsubishi.

En al parte trasera del controlador del robot encontramos un interruptor blanco encargado de encender dicho controlador. El proceso que sigue a continuación difiere según sea el objetivo para el que hemos arrancado el robot.

- *Queremos controlar manualmente el robot mediante el teaching box.* Sitúo el interruptor ST1 en posición superior y al interruptor ST2 en su posición inferior. Al mismo tiempo, pongo el interruptor del teaching box en On. Tras esto, presionaremos la secuencia de teclas NST + ENT que origina un desplazamiento del robot hasta su posición de origen, es decir, estamos realizando un *Home*. De esta forma, tras la inicialización, ya podemos empezar a controlar el robot mediante el mando de enseñanza.

- *Nuestra intención es controlar el robot mediante el programa del ordenador central.* Las patillas ST1 y ST2 deberán encontrarse en su posición inferior, y el switch del teaching box en OFF. Sólo nos restará acceder al programa del ordenador, el cual se encargará de inicializar el robot automáticamente.

- *Queremos que se ejecute el programa residente en la memoria del controlador*. Evidentemente, esta última opción es la que más nos interesa en este momento ya que durante la fabricación de las piezas es necesario que se ejecute el programa residente en la memoria del controlador. Por este motivo describiremos más detalladamente el proceso:

    - Como en los pasos anteriores, establecemos correctamente la posición de los interruptores ST1, ST2 y el interruptor del teaching box: el interruptor ST1 se levanta, el interruptor ST2 se baja y el interruptor del teaching box se sitúa en Off.

- El siguiente paso consiste en poner en ejecución el programa residente en la memoria del controlador. En la parte frontal del mismo encontramos varios botones: botón *Stop* (rojo), botón *Reset* (blanco) y botón *Start* (verde). Presionándolos en la secuencia *Stop* → *Reset* → *Start* (Rojo → Blanco →Verde) conseguiremos que el controlador del robot lea la primera línea del programa almacenado y empiece a ejecutarse a partir de la misma. En el caso de que el programa residente sea *R-4pos.pgm* el robot retornará inmediatamente hasta su posición de origen y, tras lograrlo, se situará en una posición de espera. A partir de este momento, el controlador entrará en un estado de evaluación continua de las señales de entrada, pero ¿por qué?. La respuesta la tenemos en el programa residente. Está estructurado de tal manera que los diferentes bloques de código sólo se ejecutan ante una secuencia de entrada determinada.

5.2.9. Controlador del slide.

La última etapa en el proceso de encendido de las diferentes máquinas consiste en encender el controlador del eje de deslizamiento. Sin embargo, antes de proceder a encender el controlador del slide debemos comprobar que la ruleta de posiciones, situada en una de las caras del controlador, está señalando la posición cero. Si no está en esta posición tan sólo hay que girarla hasta que logremos que aparezca dicho número.

Tras esta breve operación tendremos que girar la ruleta de encendido del controlador hasta su posición 1, y tras el encendido, el controlador del robot recibirá unas señales que provocarán la ejecución de un determinado bloque del programa. En concreto, este bloque envía unas señales al controlador del slide para que éste se desplace hasta la posición de Inicio (*Home*), y una vez logrado esto, hasta la posición +10.

5.3. ARRANCAR EL PROGRAMA FMC. ASIGNACIÓN DE PUERTOS.

Una vez encendidas todas las máquinas de la célula, le toca el turno al ordenador central (host). Presionando el botón *Power* del panel frontal, el sistema se enciende y tras unos segundos de espera entramos en Windows 95. En el escritorio de Windows se halla el icono FMC que nos permite entrar en el programa del mismo nombre.

5.3.1. Ventana principal del programa FMC.

Tras presionar el icono FMC aparece la ventana de presentación, cuya imagen está reflejada en la figura 5.4.

Figura 5.4. *Ventana de presentación.*

Haciendo clic sobre esta ventana, o esperando 5 segundos, la ventana de presentación desaparece dejando paso a la ventana principal mostrada en la figura 5.5.

Figura 5.5. *Ventana principal del programa FMC.*

La mayor parte de las opciones de trabajo y configuración se hallan accesibles en la ventana principal de FMC.

Dentro de la ventana principal encontramos, en la parte superior, la barra de título y, por debajo, el menú de opciones de FMC. Este último elemento, común prácticamente en todas las aplicaciones de Windows, contiene las diferentes opciones del programa agrupadas en varias opciones principales. En el programa FMC el menú de opciones es un apartado con estructura fija, es decir, no existe forma de personalizarlo. Algunas de las opciones disponibles ya las descubrimos en el capítulo anterior, y el resto las iremos analizando poco a poco a medida que las necesitemos.

5.3.2. Asignación de los puertos.

Es muy importante asignar correctamente los puertos a cada una de las diferentes máquinas de la FMC del LOIP, dado que de ello dependerá el correcto funcionamiento del programa.

Los puertos disponibles en el ordenador central.

Actualmente disponemos de 10 puertos serie para establecer una comunicación entre las máquinas y el ordenador central (host):

- √ 8 de los 10 puertos los proporciona la tarjeta de comunicaciones *National Instrument PCI-232/8* que se haya instalada en dicho ordenador. En concreto, estaríamos hablando de los puertos *COM5, COM6, COM7, COM8, COM9, COM10, COM11 y COM12*.
- √ Los dos puertos serie propios de todo ordenador personal: *COM1 y COM2*.

Cada uno de los puertos serie de la tarjeta *National Instrument PCI-232/8* están numerados físicamente del 1 al 8 para poder distinguirlos. Con el fin de evitar cualquier confusión, el programa FMC emplea este formato a la hora de visualizar o configurar los puertos. Así, el puerto serie COM5 se corresponde con el valor numérico 1, el puerto serie COM6 con el valor 2, etc. En la tabla 5.3. se puede apreciar la correspondencia numérica de cada uno de los puertos serie de que dispone el ordenador central de la célula del LOIP.

Puerto Serie	Valor Numérico
COM5	1
COM6	2
COM7	3
COM8	4
COM9	5
COM10	6
COM11	7
COM12	8
COM1	9
COM2	10

Tabla 5.3. *Correspondencia numérica de los puertos serie.*

Tal y como se puede observar en la tabla superior, los puertos serie COM1 y COM2 tienen asociados los valores numéricos 9 y 10.

Opción *Nueva Asignación de Puertos*.

Una de las suposiciones planteadas inicialmente hacía referencia al hecho de que toda la maquinaria de la Célula era de reciente adquisición, lo que conlleva que todavía no hemos asignado los puertos en el programa FMC, y más aún, que no hemos creado ninguna configuración. Por este motivo, el primer paso que vamos a dar se encaminará hacia la creación de una nueva asignación de puertos.

Para conseguir dar este primer paso disponemos de la opción *Nueva Asignación de Puertos*, que se encuentra dentro de *Asignación de Puertos* del menú *Herramientas*.

Seleccionando dicha opción, o pulsando la combinación de teclas <Ctrl+N>, nos aparecerá el módulo de la figura 5.6.

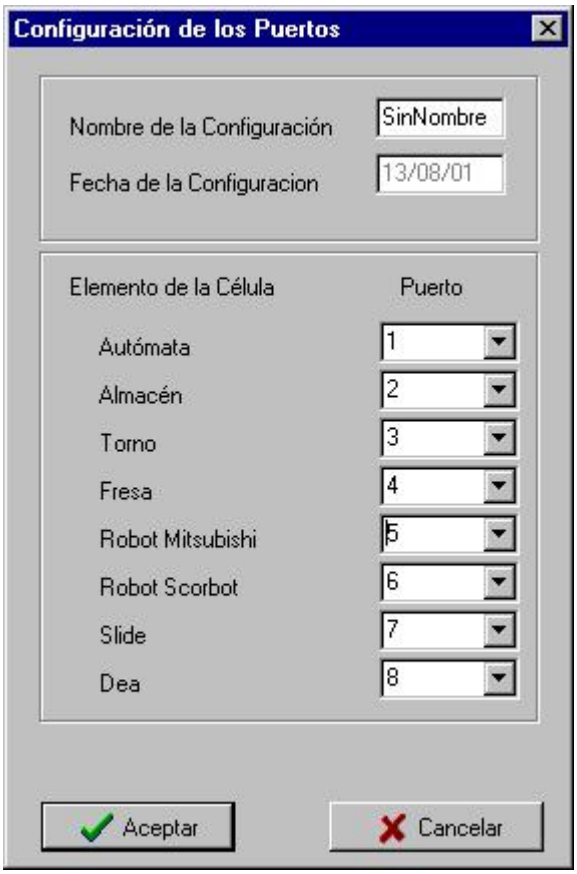


Figura 5.6. *Ventana Nueva Asignación de Puertos.*

En la parte superior de dicha ventana disponemos de dos controles de edición:

- √ En el primero de ellos insertaremos el nombre de la nueva configuración que queremos crear; el valor por defecto es SinNombre.
- √ En el segundo campo de edición se muestra, tal y como se podrá apreciar, la fecha actual. No vamos a poder modificar el valor de este cuadro.

Tras haber introducido el nombre de la nueva configuración, toca el turno a la asignación de puertos. Como ya sabemos, existen 10 posibles puertos serie entre los que poder elegir y sólo 8 máquinas (incluido el Centro de Calidad), por lo que nos quedarán dos puertos libres.

La configuración que aparece al acceder a esta ventana es la que actualmente se encuentra en uso, es decir, la que utilizaremos por 'defecto' durante el proyecto. Dicha asignación se muestra a continuación:

Máquina	Puerto
Autómata	1
Almacén	2
Torno	3
Fresadora	4
Robot Mitsubishi	5
Robot Scorbot	6
Eje de Deslizamiento (Slide)	7
Centro de Calidad (DEA)	8

Tabla 5.4. *Asignación de Puertos.*

Podemos observar que en esta configuración hemos empleado todos y cada uno de los puertos de la tarjeta de comunicaciones, quedando libres los puertos COM1 (9) y COM2 (10).

Tras la asignación correcta de los puertos (un puerto por cada máquina) tan solo debemos presionar el botón *Aceptar*, que se encuentra situado en la parte inferior izquierda de la ventana. En ese momento, el programa se encarga de guardar la nueva asignación en la tabla *Puertos.db* ubicada en el directorio "C:\Celula00_01\Tablas\AsignarPuertos", y tras ello, la ventana *Nueva Asignación de Puertos* desaparece devolviendo el control a la ventana principal.

Si en cualquier momento deseamos salir de la ventana *Nueva Asignación de Puertos* sin guardar la configuración, tan sólo debemos presionar el botón *Cancelar* que se encuentra a la derecha del botón *Aceptar.*

Ventana *Ver las Diferentes Asignaciones de los Puertos*.

Tras crear nuestra configuración debemos copiarla en la configuración denominada *Actual*, que es la que utiliza el programa *FMC*. Para ello, accederemos a la opción *Herramientas* |*Asignación de Puertos* | *Ver las Diferentes Asignaciones de los Puertos*, o bien pulsaremos la combinación de teclas <Ctrl+V>. Tras unos segundos surge la ventana de la figura 5.7., de apariencia muy similar a la ventana mostrada en la figura 5.6.

Figura 5.7. *Ventana Ver las Diferentes Asignaciones de los Puertos.*

Gracias a esta ventana podemos acceder a cada una de las asignaciones de puertos existentes, podemos eliminar aquellas que no nos interesen y podemos establecer una configuración dada como Actual.

El nombre de las configuraciones de los puertos existentes se muestra en la parte superior de la ventana, así como la fecha en la que fue creada o modificada por última vez. En la parte central se muestra la asignación máquina↔puerto de cada uno de las configuraciones existentes.

Para poder navegar entre las diferentes configuraciones disponemos de un control *TDBNavigator* situado debajo de los campos de edición. Este control es bastante sencillo puesto que sólo contiene 4 botones: primer registro, registro anterior, registro siguiente y último registro. Se ha optado por no utilizar el resto de los botones que puede albergar un control *TDBNavigator*, y en especial los de edición, debido a que el objetivo de esta ventana no es la modificación manual de los valores que se muestran en pantalla; el programa se encargará de hacer las modificaciones pertinentes según se lo indiquemos.

En este sentido, en la parte inferior del cuadro de diálogo hallamos un control *TRadioGroup* con dos botones de radio. Estos dos botones representan las dos posibles opciones de edición sobre la tabla que contiene los datos de las asignaciones de puertos.

- **Establecer esta Configuración como Actual**. Tal y como hemos comentado anteriormente, *Actual* es la configuración que necesita el programa para su ejecución. Seleccionando este botón de radio nos aseguraremos que los datos de la asignación que se muestra en ese momento se copiarán en la configuración **Actual**.
- **Borrar esta Configuración**. Activando este botón de radio estaremos indicando al programa que la configuración que se muestra actualmente en pantalla debe desaparecer de la tabla *Puertos.db*.

En un principio ninguno de estos dos botones de radio está activado, por lo que si queremos llevar a cabo alguna de las operaciones anteriores deberemos seleccionar previamente uno de ellos. Estas dos operaciones son mutuamente excluyentes, por lo que sólo podremos seleccionar una de ellas.

Tras esto, sólo nos resta presionar el botón *Aceptar* y el programa se encargará de realizar la operación que le hemos ordenado, devolviendo el control a la ventana principal.

Si en cualquier momento deseamos abandonar la ventana *Ver Asignación de los Diferentes Puertos* sin realizar ninguna operación, tan sólo debemos presionar el botón *Cancelar*.

Para finalizar este apartado comentaremos que el programa nos impide eliminar la configuración *Actual*. Por esta razón, si hemos seleccionado esta configuración y el botón de radio *Borrar Configuración* se encuentra activo, al presionar el botón *Aceptar* surge el cuadro de diálogo de la figura 5.8., que nos informa que la configuración *Actual* no puede ser borrada por ser necesaria en el programa *FMC*.

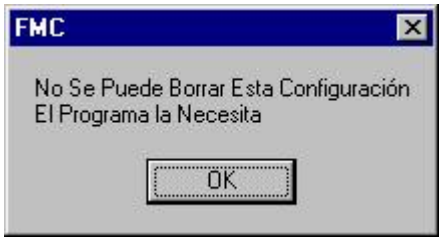


Figura 5.8. *Mensaje de advertencia.*

El mismo mensaje nos aparece si intentamos eliminar la configuración *Defecto*, que es, evidentemente, la asignación que se utiliza por defecto. Esta es la configuración que se muestra nada más acceder a la ventana *Nueva Asignación de Puertos*, que es la que actualmente está presente en la célula.

5.4. PROGRAMAS RESIDENTES EN LOS CONTROLADORES. GESTIÓN DEL STOCK.

Tras definir la configuración de puertos que vamos a emplear, es el turno de enviar los programas de ejecución a la memoria de cada controlador. En concreto enviaremos[4]:

- Al autómata el programa *Residente.cim*.
- Al robot Mitsubishi los programas *R-4pos.pgm y R-4pos.cdr*.
- Al eje de deslizamiento el programa *4pos.prg*.

Por otro lado, una vez hayamos enviado estos archivos quedarán guardados en la memoria del controlador correspondiente. Por lo tanto, si sabemos que estos programas ya están en la memoria de sus sistemas de control no será necesario llevar a cabo este proceso nuevamente.

Otra tarea importante que debemos llevar a cabo antes de comenzar la ejecución de un proceso es la gestión inicial del stock. Debemos comprobar que los datos del almacén están correctamente introducidos, es decir, debemos verificar que el estado real de cada celda se corresponde con el estado que se refleja en la tabla *MaterialActualCelda.db*, a la que podemos acceder, tal y como sabemos, mediante el módulo de control remoto del almacén.

Una vez hecha esta verificación tenemos que diseñar la distribución en planta, o disposición de las máquinas, de la FMC.

[4] No vamos a explicar cómo transmitir estos archivos a los sistemas correspondientes, dado que dicha tarea ya se realizó en el capítulo anterior.

5.5. DISEÑO DEL LAYOUT DE LA CÉLULA. LAS ESTACIONES.

Para poder comenzar el diseño de la distribución de nuestra Célula tan solo debemos acceder a la opción *Editar el Diseño de un Célula* del menú principal *Célula de Fabricación Flexible* del programa *FMC*, o bien presionar la combinación de teclas <Ctrl+C>. Nos aparecerá una ventana como la representada en la figura 5.9.

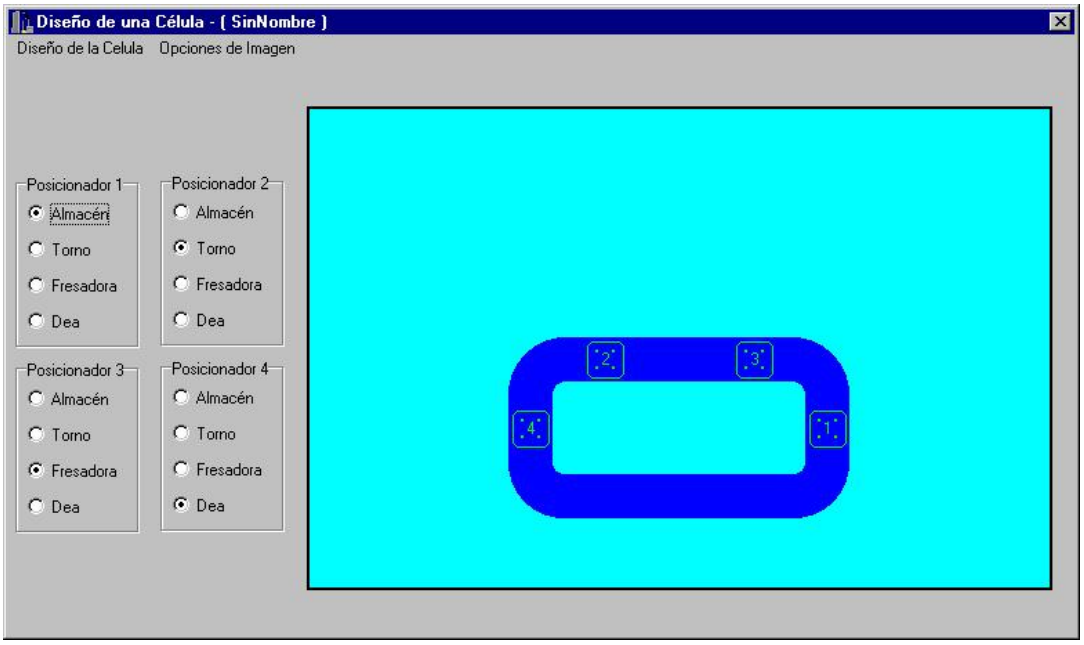


Figura 5.9. *Ventana Diseño de una Célula.*

5.5.1. Descripción de la ventana *Diseño de una Célula*.

En esta ventana descubrimos un menú principal con las opciones relativas tanto a la célula (*Diseño de la Célula*) como a las imágenes que posteriormente insertemos (*Opciones de Imagen*). Cada una de estas opciones las iremos analizando poco a poco.

En la parte central izquierda se encuentra la zona en la que esbozaremos la distribución en planta de la célula, el *Área de Diseño*. Inicialmente sólo aparece el dibujo de la cinta transportadora sobre un fondo azul. Sobre el conveyor se aprecian unos pequeños iconos cuadrados de color verde que representan cada una de las cuatro estaciones de trabajo (workstations) que existen en la cinta. Estos iconos, tal y como veremos más adelante, guardan una estrecha relación con los controles *TRadioGroup* situados en la parte central izquierda de la ventana.

Para saber en todo momento sobre qué estación estamos actuando, éstas están perfectamente numeradas del 1 al 4.

5.5.2. Creación de una distribución en planta de la célula.

Nuestra intención en este apartado es diseñar el **Layout** de la célula. El primer paso consistirá, por lo tanto, en dibujarla. Para ello, tendremos que insertar correctamente las imágenes que representan a cada una de las diferentes máquinas que van a componer nuestra célula.

Insertar imágenes en el área de diseño.

Con este propósito accedemos a la opción *Insertar Imagen* del menú *Opciones de Imagen.* Una vez seleccionada, surge una nueva ventana que nos servirá para seleccionar la imagen a insertar (figura 5.10.). Podemos lograr el mismo resultado de forma mucho más rápida si hacemos clic con el botón derecho del ratón sobre el área de diseño; esto ocasiona la aparición de un menú emergente con la opción *Insertar Imagen.*

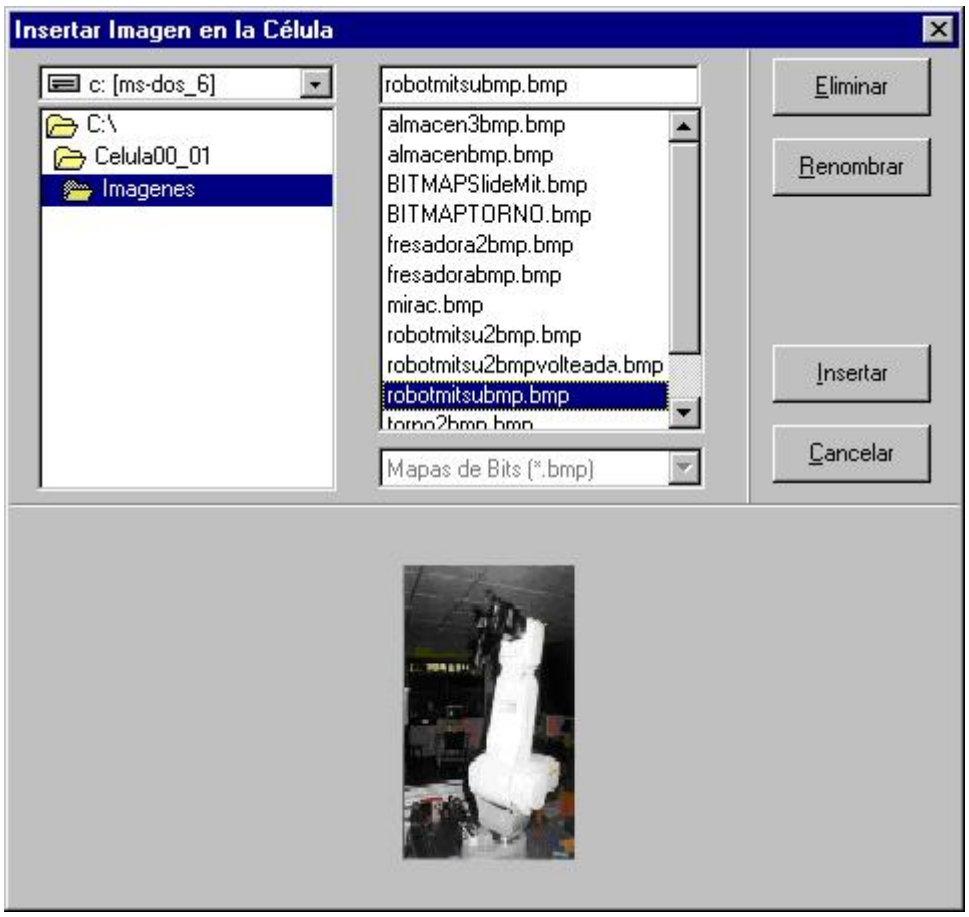


Figura 5.10. *Ventana encargada de la selección de imágenes.*

Inicialmente, la unidad activa en este cuadro de diálogo es la "C" y el directorio de trabajo es "C:\Celula00_01\Imagenes", que es el directorio en el que, por defecto, se encuentran las imágenes que debemos insertar. Aunque es posible cambiar el directorio actual en busca de nuevas imágenes que insertar, es recomendable que toda imagen que podamos haber generado previamente con el editor de imágenes, se guarde en este directorio.

De cualquier forma, el control *TFileListBox* nos muestra, en todo momento, los archivos de *mapas de bits* ubicados en el directorio seleccionado actualmente. Este hecho nos obliga a generar únicamente archivos con extensión *BMP* para representar las diferentes máquinas de la FMC del LOIP.

Tras elegir un archivo de mapa de bits del control *TFileListBox*, la imagen en él guardada se muestra en la parte inferior de la ventana con su tamaño real. Si deseamos contemplar la imagen con mayor detalle, tan solo debemos hacer clic con el botón derecho del ratón sobre la parte inferior del formulario y automáticamente emergerá un menú en el que podremos seleccionar la opción *Ajustar a la Ventana*. Activando esta opción, la imagen se muestra ocupando toda la parte inferior del cuadro de diálogo. Para volver a visualizar la representación en su tamaño real debemos seguir el mismo proceso que antes, solo que ahora nos aparecerá la opción *Tamaño Real* en el menú emergente.

En la sección derecha de esta ventana encontramos los siguientes botones:

- *Botón* ***Eliminar.*** Este botón nos permite suprimir del directorio actual el archivo elegido en el control *TFileListBox*. El programa nos pide confirmar dicha acción antes de proceder con la eliminación del archivo.
- *Botón* ***Renombrar.*** Tal y como su nombre nos indica, este botón nos permite renombrar el archivo que hemos seleccionado. Tras pulsar este botón nos aparece un cuadro de diálogo en el que podremos introducir, sin extensión, el nuevo nombre del archivo.
- *Botón* ***Cancelar.*** Si en cualquier momento deseamos regresar a la ventana de diseño sin introducir una imagen, tan solo deberemos hacer clic sobre esta opción y automáticamente la ventana de selección desaparecerá de nuestra pantalla.
- *Botón* ***Insertar.*** Este es el botón que debemos pulsar para introducir la imagen seleccionada en el layout de la Célula.

En cualquier caso, a excepción del botón *Cancelar*, el resto de los botones mostrados anteriormente no se ejecutan si no se ha seleccionado previamente un archivo.

Siguiendo con el proceso de diseño, y tal como se ha comentado, cuando tenemos la imagen que queremos incluir en el área de diseño presionamos el botón *Insertar*. En ese momento, la ventana de selección desaparece y la imagen elegida se muestra en la esquina superior izquierda de la zona de dibujo. Para poder desplazar la imagen sobre cualquier punto del área de diseño tan sólo nos situaremos encima de ella, haremos clic con el botón izquierdo del ratón y, manteniéndolo pulsado, nos desplazaremos hasta la posición adecuada, momento en el que dejaremos de presionar el ratón. Este proceso de desplazamiento lo podremos realizar siempre que deseemos, lo que nos proporciona una gran flexibilidad a la hora de diseñar la célula (figura 5.11.).

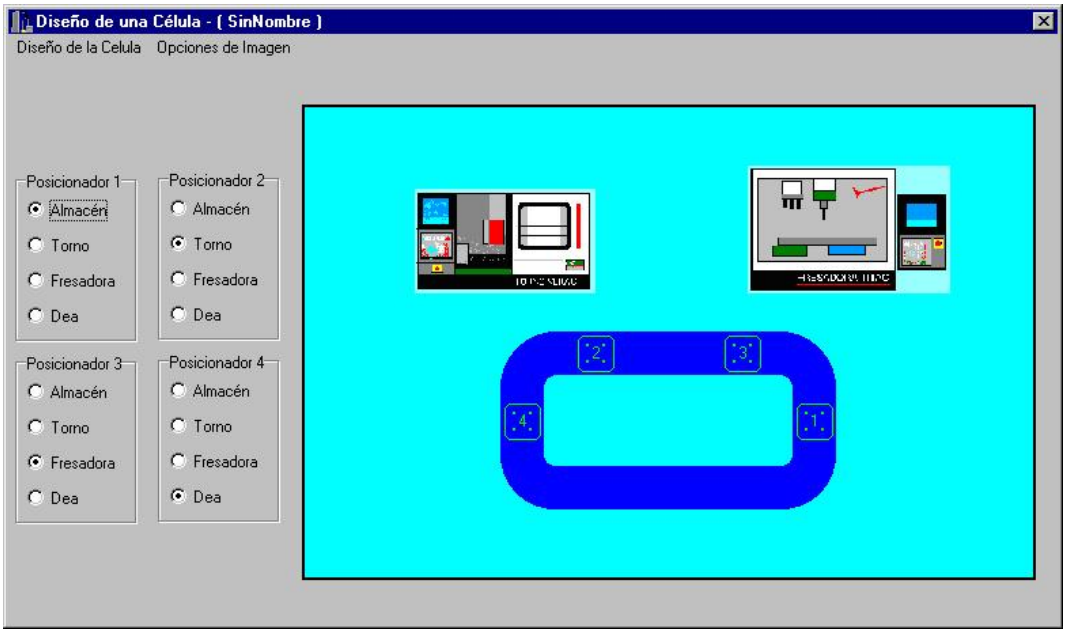


Figura 5.11.*Ejemplo de un diseño.*

Repitiendo esta misma actuación podremos incluir hasta un total de ocho imágenes. No se da la opción de insertar un mayor número para evitar que el diseño quede excesivamente recargado.

Si en cualquier momento del proceso de diseño una imagen no nos resulta adecuada, podemos eliminarla fácilmente. Para ello, como primer paso, debemos situarnos sobre la imagen que deseamos eliminar. Tras presionar el botón derecho del ratón emerge un menú con la opción *Eliminar Imagen*. Activando esta opción conseguiremos eliminar la imagen del layout.

Elección de las estaciones activas. Asignación de las máquinas a las estaciones.

Una vez esbozado el dibujo de la célula del LOIP, el siguiente paso será determinar aquellas estaciones que durante el proceso de ejecución no van a estar activas. Por defecto, todas las estaciones aparecen con un color verde, lo que nos indica que todas están inicialmente activas.

- Para poder *Desactivar* una estación, tan sólo debemos hacer clic con el botón izquierdo del ratón sobre el icono correspondiente, y acto seguido la estación aparecerá representada con color rojo y el control *TRadioGroup* asociado se inhabilitará.

- *Activar* cualquier estación inactiva es tan sencillo como desactivarla; lo único diferente es que la estación pasa de representarse con un color rojo a representarse con el color verde y que el control *TRadioGroup* correspondiente vuelve a habilitarse, lo que nos permite acceder a sus diferentes botones de radio.

Pero, ¿qué significa que una estación esté activa o inactiva?. Este calificativo tiene que ver con el proceso de ejecución. Una estación obligatoriamente estará activa cuando la máquina asociada a dicho terminal tenga que realizar algún proceso (mecanizado, inspección,...) durante la ejecución de la tarea. Si por el contrario el equipo asociado no debe realizar proceso alguno, o bien, no existe máquina asociada a la estación, ésta deberá estar inactiva.

Tras establecer el estado de las estaciones, el siguiente paso es asociarlas a una determinada máquina de la célula. Por defecto, la estación 1 está asociada al Almacén, la estación 2 al centro de torneado, la estación 3 al centro de fresado y, por último, la estación 4 al centro de calidad. Para cambiar dicha asignación, tan solo debemos hacer clic con el botón izquierdo del ratón sobre el botón de radio correspondiente. Obviamente, en una estación inactiva no podremos modificar la máquina asociada.

Para la correcta elección de las máquinas asociadas tan sólo hay que respetar dos reglas:

1) El almacén siempre debe estar asignado a una estación.
2) No vamos a poder asignar la misma máquina a dos o más estaciones.

En caso de incumplirse alguno de estos requisitos, el programa nos avisa en el momento de *Guardar el Diseño.*

Antes de pasar a la siguiente fase debemos comentar que aunque hemos seguido un proceso determinado a la hora de diseñar la célula, nada nos impide variarlo. Es decir, podemos, por ejemplo, asignar inicialmente las máquinas a sus estaciones, desactivar las estaciones correspondientes y finalmente insertar las imágenes en el área de diseño. Lo único que debemos tener claro son las respuestas a las preguntas ¿cómo es la célula? y ¿qué máquinas trabajarán durante la ejecución de la tarea?.

Guardar el diseño.

Una vez que hemos definido el layout de la célula, qué estaciones estarán activas durante la ejecución y cuáles son las máquinas asociadas a las estaciones activas, sólo nos resta guardar el diseño de la célula. Presionando sobre la opción *Guardar Diseño de Célula* del menú principal *Diseño de la Célula* surge un nuevo cuadro de diálogo (véase figura 5.12.):

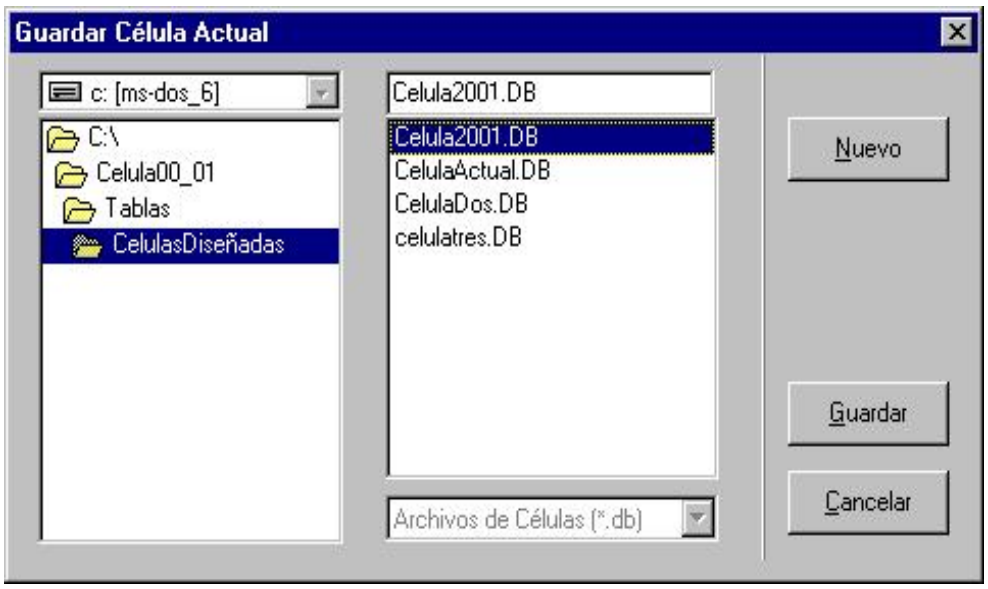


Figura 5.12. *Cuadro de Diálogo Guardar.*

Este cuadro de diálogo presenta algunas curiosidades:

- No vamos a poder elegir la dirección donde guardar el diseño; la ruta que aparece por defecto es "C:\Celula00_01\Tablas\DiseñoCelulas", que es el directorio donde se almacenan todas las TABLAS con información relativa a una célula.

- Del Componente *TFileListBox*, donde se muestran todas las tablas que actualmente guardan información sobre células, debemos seleccionar una tabla. Si deseamos guardar los datos en un nuevo archivo presionaremos el botón *Nuevo*, tras lo cual nos aparecerá un cuadro de diálogo en el que introduciremos el nuevo nombre pero, eso sí, sin extensión alguna (el programa se encarga de asignarle la extensión DB que es la que empleada por las tablas de tipo paradox que venimos utilizando).

- Después de seleccionar una tabla, o crearla, debemos pulsar el botón *Guardar*, tras lo cual, el programa se encarga de guardar todos los datos necesarios en la tabla escogida. Si no deseamos guardar el diseño deberemos presionar el botón *Cancelar* de la misma ventana.

Al pulsar la opción *Guardar*, el cuadro de diálogo desaparece, el nombre del diseño se muestra en el título de la ventana y los datos necesarios son guardados. Pero, ¿qué datos se guardan?. En la tabla elegida se guardan los datos relativos a las imágenes insertadas en el diseño, mientras que la información relativa a las estaciones se guarda en una tabla aparte, *posicionadores.db*, ubicada en el diretorio "C:\Celula00_01\Tablas\Posicionadores". En *posicionadores.db* se inserta el nombre de la tabla que se utiliza para almacenar los datos de las imágenes que conforman el layout, así como los datos relativos a las estaciones (si están activas o no, su máquina asociada, etc.). Se puede ver una descripción más detallada de estas tablas en el capítulo sexto.

En la imagen de la figura 5.13. se puede apreciar tanto la disposición de las máquinas de la célula como la asignación de estaciones que emplearemos durante el proceso que sigue al diseño de la célula, y que es *CelulaActual.db*.

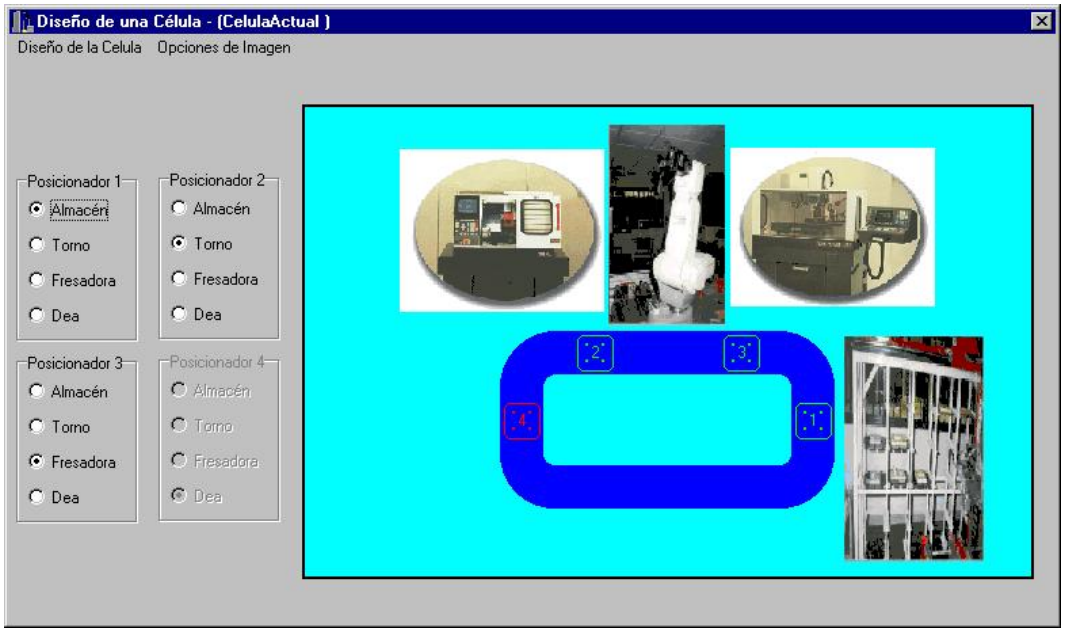


Figura 5.13. *Diseño de la Célula CelulaActual.db.*

5.5.3. Otras opciones relacionadas con el diseño de la célula.

Para finalizar el punto 5 de este capítulo, sólo nos queda comentar tres opciones del menú principal *Diseño de la Célula*:

- **Opción *Editar un Diseño de Célula Existente.*** Seleccionando esta opción surge un cuadro de diálogo muy similar la de la figura 5.12., pero en el que no aparece la opción Nuevo y el botón Guardar se ha transformado en un botón Editar. Con este cuadro de diálogo podemos seleccionar la célula que queremos modificar. Tras seleccionarla y presionar el botón Editar, el layout de la célula elegida se mostrará en la ventana inicial y el nombre de la nueva Célula aparecerá en el título de la ventana. De esta forma podremos modificarla fácilmente siguiendo todos los pasos vistos anteriormente.

- **Opción *Inicializar el Diseño de la Célula.*** Esta opción elimina cualquier imagen o alteración que hayamos podido introducir en el diseño. En definitiva, tras pulsar esta opción, la ventana se muestra tal y como surgió al iniciar la sesión de diseño.

- **Opción *Salir.*** Este control provoca la desaparición de la ventana de diseño devolviendo el control a la ventana principal del programa *FMC*.

5.6. DISEÑO DE LAS TAREAS.

Para comenzar el diseño de una tarea tan solo hay que hacer clic en la opción *Editar el Diseño de una Tarea* del menú principal *Célula de Fabricación Flexible* del programa *FMC*, o bien presionar la combinación de teclas <Ctrl+T>.

Tras realizar alguna de estas operaciones, en la pantalla del ordenador central aparecerá la ventana de la figura 5.14.

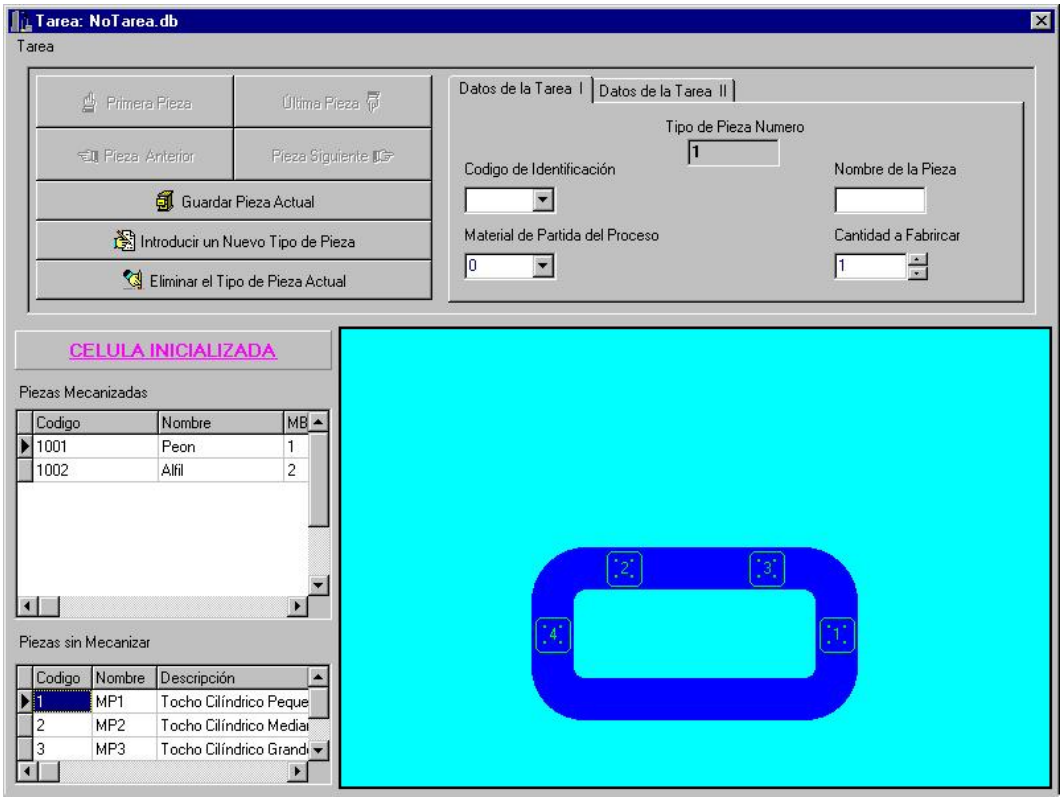


Figura 5.14. *Ventana Diseño de una Tarea.*

5.6.1. Descripción de la ventana *Diseño de una Tarea*.

En esta ventana descubrimos un menú principal con las opciones relativas a las tareas. Iremos analizando dichas opciones a medida que las necesitemos.

En la parte inferior derecha se encuentra la zona en la que se mostrará la distribución en planta de la célula. Inicialmente sólo aparece el dibujo de la cinta transportadora sobre un fondo azul. Sobre el conveyor o cinta transportadora se aprecian unos pequeños iconos cuadrados de color verde que representan cada una de las cuatro estaciones de trabajo que existen en la cinta. Como ya conocemos, cada una de estas estaciones están numeradas del 1 al 4.

En la parte inferior izquierda observamos unos controles *TDBGrid* que nos permiten conocer las características importantes tanto de las piezas que podemos fabricar, *Piezas Mecanizadas*, como de las materias primas, *Piezas sin Mecanizar*.

Finalmente, en la parte superior descubrimos los diferentes controles que nos permitirán tanto navegar por una tarea como editarla. En el proceso de diseño se describirán cada uno de estos elementos detenidamente.

5.6.2. Creación de una tarea.

Como se recordará, el objetivo planteado al inicio de este capítulo fue la creación de las 32 piezas de un juego de ajedrez. En principio, podríamos intentar incluir todas las piezas en una sola tarea. Sin embargo, dado que el almacén limita la producción a 24 piezas, deberemos crear dos tareas. En la primera de ellas definiremos las piezas de color negro y en la segunda las de color blanco.

Como primer paso en el diseño de estas tareas debemos hacer clic en la opción *Crear Nueva Tarea* del menú *Tarea*. Tras esto, surge un cuadro de diálogo donde vamos a poder seleccionar el archivo de una célula. Seleccionamos el diseño *CelulaActual.db*, que fue, como se recordará, la distribución creada en el apartado anterior.

Inmediatamente después de presionar el botón *Aceptar*, la ventana de selección desaparece y en la zona de diseño surge el layout de la célula seleccionada. Debemos observar que aquellas estaciones que se definieron como inactivas en el proceso de diseño no aparecen ahora en el layout de la célula.

Al mismo tiempo, los controles superiores se habilitan para su uso, ya que hasta este momento, y como se habrá comprobado si se ha intentado acceder a ellos, han permanecido inhabilitados. Tras habilitarse los controles superiores, podemos proceder a introducir los datos correspondientes al primer tipo de pieza a mecanizar.

En la página *Datos de la Tarea I* seleccionamos el código de identificación de la pieza que queremos fabricar, entre los códigos disponibles del control *TComboBox* correspondiente. Estos códigos se relacionan con los mostrados en el control *TDBGrid* superior (*Piezas Mecanizadas*). Tras esta selección los campos destinados al nombre de la pieza y al tipo de material requerido se rellenarán automáticamente.

Inicialmente, el código de la pieza de partida es el que aparece en la tercera columna del componente *TDBGrid* superior, que se corresponde, como sabemos, con la columna *Códigos* del control *TDBGrid Piezas sin Mecanizar*. Sin embargo, si observamos la lista asociada a este campo, podemos apreciar que tiene dos elementos: por un lado el código de la pieza sin mecanizar y, por otro, el código de la propia pieza mecanizada. Pero, ¿qué sentido tiene partir de una pieza ya terminada?. La explicación la tiene el proceso de inspección. En ciertas ocasiones, puede ocurrir que necesitemos inspeccionar en el centro de calidad una pieza ya fabricada de antemano. En este caso, la pieza no sufriría modificación alguna y, por lo tanto, la pieza final e inicial serían la misma.

En esta misma página también estableceremos el número de veces que la pieza, que estamos definiendo, debe fabricarse durante la ejecución de la tarea.

En la hoja *Datos de la Tarea II,* tenemos las opciones que relacionan la pieza a fabricar con las diferentes máquinas de mecanizado e inspección. En concreto indicaremos si la pieza requiere una operación de torneado, fresado y/o calidad, así como el archivo que deberá ejecutarse en caso afirmativo.

Por ejemplo, el primer tipo de peón, código de identificación 1001, sólo requiere la ejecución del programa *Peon.mir* en el centro de torneado. Para indicarle esto al programa, activaremos el campo de selección del torno, lo que provocará que el control *TComboBox* anexo se active. Este control mostrará los diferentes archivos de control numérico disponibles y, de entre todos ellos, seleccionaremos el archivo *Peon.mir*. Un proceso similar se sigue cuando la pieza requiere una operación de fresado o inspección. En este último caso, también tendremos que introducir el número de piezas que requerirán ser inspeccionadas.

Este número estará comprendido entre cero y el número total de piezas que debemos fabricar.

Tras haber introducido los datos de la primera pieza, toca el turno de guardarlos presionando el botón *Guardar Pieza Actual*. Si los datos no se han introducido correctamente el programa nos avisa para que realicemos las modificaciones oportunas. Pero, ¿qué puede provocar un error a la hora de guardar los datos?.

- No hemos introducido el código de la pieza a fabricar (o inspeccionar).
- El número de piezas a fabricar excede de 24, que es el número máximo de celdas de que disponemos en el almacén.
- No hemos seleccionado un archivo para un proceso que sí lo requiere. Por ejemplo, hemos activado el control de selección de la fresadora, pero no hemos elegido el programa de control numérico que se ejecutará.

Una vez guardados los datos de la primera pieza a fabricar, podemos introducir los datos de las piezas restantes. Para ello, tan sólo debemos presionar el botón *Introducir un Nuevo Tipo de Pieza*, y repetir el proceso descrito anteriormente.

Para poder navegar entre los distintos tipos de piezas disponemos de 4 botones: primera pieza, pieza anterior, pieza siguiente y última pieza. Hemos preferido que la navegación por las piezas se realizase con estos botones, y no con un control *TDBNavigator,* que es lo normal, con el objetivo de hacer la navegación más intuitiva y más sencilla.

El botón *Eliminar el Tipo de Pieza Actual* nos permite suprimir piezas de la tarea. Para ello, deberemos situarnos previamente en la pieza que queremos eliminar.

Tras introducir y guardar los datos de todas las piezas, es el momento de guardar la tarea. La opción *Guardar Tarea* nos permite realizar dicha operación. Presionando dicha opción daremos paso a un cuadro de diálogo (véase figura 5.15.) que nos permitirá guardar la tarea en el directorio "C:\Celula00_01\Tablas\ TareaDiseñada".

Figura 5.15. *Cuadro de diálogo Guardar Tarea.*

Esta ventana nos ofrece la posibilidad de guardar la tarea tanto en un archivo ya existente como crear uno nuevo. En el primer caso sólo tendremos que seleccionar el archivo de la lista y presionar el botón *Aceptar*. En el segundo, pulsaremos el botón *Nuevo* que dará paso a la ventana de la figura 5.16.

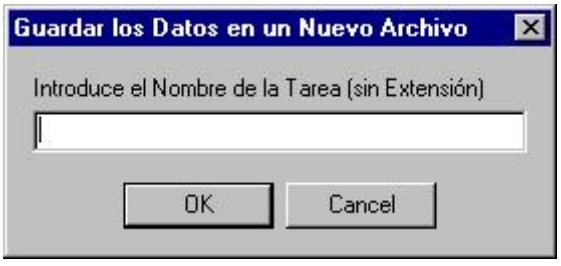


Figura 5.16. *Introducir un nuevo nombre para la tarea.*

En esta ventana introduciremos el nombre de la tarea sin extensión (el programa le da la extensión DB de las bases de datos tipo Paradox). Al presionar *Aceptar* regresaremos a la ventana de la figura 5.15. Tan sólo nos restará presionar el botón Aceptar de esta ventana y la tabla se guardará con el nombre elegido.

Tras guardar la tarea, el nombre de la misma aparece en la barra de título de la ventana *Diseño de una Tarea*.

5.6.3. Otras opciones relacionadas con el diseño de una tarea.

Para finalizar el punto 6 de este capítulo, sólo nos queda comentar tres opciones del menú principal *Diseño de la Célula*:

- ***Editar una Tarea Existente.*** Presionando esta opción daremos paso a un cuadro de diálogo muy similar la de la figura 5.15., pero en el que no aparece el botón Nuevo. Con este cuadro de diálogo podemos seleccionar la tarea que queremos editar. Tras seleccionarla y presionar el botón *Aceptar*, los datos de la tarea elegida se muestran en los controles correspondientes y su nombre aparece en el título de la ventana. De esta forma, podremos modificarla fácilmente siguiendo todos los pasos analizados anteriormente.

- ***Inicializar los Datos.*** Esta opción elimina cualquier imagen o alteración que hayamos podido introducir en el diseño. En definitiva, tras pulsar esta opción, la ventana se muestra tal y como surgió al iniciar la sesión de diseño.

- ***Salir.*** Este control provoca la desaparición de la ventana de diseño devolviendo el control a la ventana principal del programa *FMC*.

5.7. SECUENCIACIÓN.

Después de diseñar la tarea que debe ejecutarse llega el momento de secuenciarla. La secuenciación supone establecer el orden en el que debe procesarse cada pieza dentro del sistema productivo.

Sin embargo, esta herramienta es parte del estudio de un proyecto de secuenciación, por lo que aún no está diseñada. En su lugar, se ha creado una aplicación que descompone la tarea, anteriormente creada, para que pueda ser utilizada por el programa encargado de la ejecución. Esta aplicación es *DesCo*.

5.7.1. DesCo. Desglose de una tarea.

A este programa se accede haciendo clic sobre la opción *Desglose de Tarea* del menú principal *Herramientas.*

Figura 5.17. *Programa de descomposición de tarea DesCo.*

DesCo es un programa sencillo tanto de interfaz como de manejo. Está compuesto por un control *TPageControl* con tres páginas y dos botones.

En la primera de las páginas del control *TPageControl*, *Tarea a Desglosar*, encontramos un control *TFileListBox* que nos permite seleccionar la tarea a desglosar. Estas tareas están, como ya sabemos, en el directorio C:\Celula00_01\Tablas\TareasDiseñadas.

La segunda página, *Tipo de Inspección*, nos permite seleccionar cómo serán inspeccionadas las piezas de la tarea que requieran dicha operación. Las opciones posibles son:

- **Aleatoria**. Las piezas se inspeccionan aleatoriamente.
- **Inicial**. Las primeras piezas en fabricarse son las que se inspeccionan.
- **Intermedia**. En este caso son las piezas que se fabrican a la mitad del proceso las que van a ser inspeccionadas.
- **Final**. Se inspeccionarán las últimas piezas fabricadas.
- **Alternada**. Las piezas a inspeccionar se intercalarán entre las piezas que no deban ser inspeccionadas.

La tercera página, *Nombre de la Tarea Desglosada*, nos permite introducir el nombre con el que se guardará la tarea desglosada en el directorio C:\Celula00_01\Tablas\TareasDiseñadas. Inicialmente, el nombre de esta tarea será el de la tarea de partida+"_D". Se ha elegido este formato para diferenciar las tareas secuenciadas de las simplemente Desglosadas. En cualquier caso, este nombre puede ser modificado por el usuario.

Tras haber elegido la tarea a desglosar y haber modificado el resto de las opciones convenientemente, es el momento de pulsar el botón *Desglosar Tarea*. Este botón se encargará de ejecutar el código de programa destinado a disgregar adecuadamente la tarea. Tras esto, el control es devuelto al programa principal.

Si en cualquier momento deseamos abandonar el programa sin desglosar ninguna tarea tan sólo tendremos que hacer clic sobre el botón *Regresar al Programa Principal*.

5.8. EJECUCIÓN DE LA TAREA.

Tras definir el diseño de la célula, establecer las tareas y secuenciarlas, es el momento de comenzar el proceso de elaboración de las piezas de ajedrez.

El módulo *Ejecución de la Tarea*, al que se accede a través del menú principal *Célula de Fabricación Flexible*, se va a encargar de controlar todas las máquinas, y sus señales, con el fin de fabricar las 32 piezas del juego de ajedrez.

El módulo *Ejecución de la Tarea* aparece representado en la figura 5.18. que se muestra a continuación.

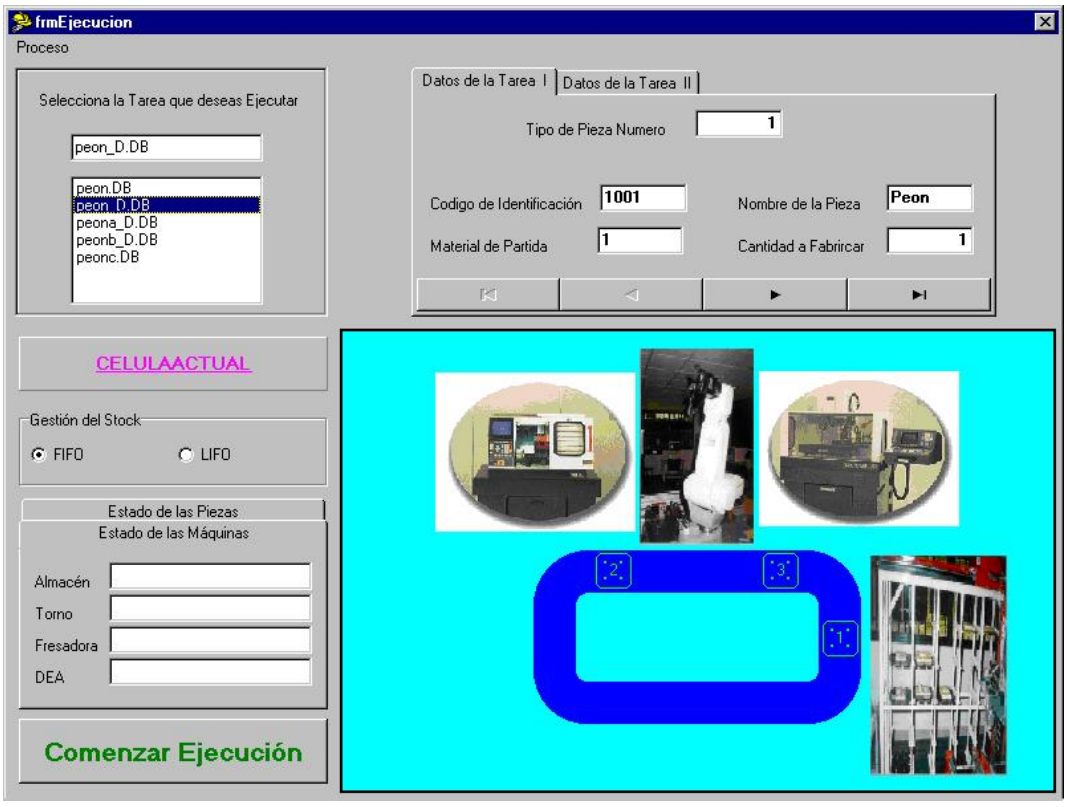


Figura 5.18. *Apariencia inicial de la ventana Ejecutar Tarea.*

5.8.1. Breve descripción de la ventana *Ejecución de una Tarea*.

Se trata de una ventana muy similar, en apariencia, a la ventana destinada al diseño de tareas. De hecho, tenemos un área de representación del layout de la célula, y un control *TPageControl* desde el que vamos a poder visualizar los datos de las piezas que contenga la tarea que seleccionemos (esta vez disponemos de controles *TDBNavigator* para la navegación entre las piezas).

Además, disponemos de los siguientes elementos.

- En la parte superior izquierda tenemos un componente *TFileList* que me permite escoger la tarea a procesar. Sólo se pueden elegir aquellas tareas que estén secuenciadas o desglosadas.
- Por debajo del elemento anterior disponemos de un control *TRadioGroup* con dos opciones, *FIFO y LIFO*, que se corresponden a los modos de gestión del almacén durante el proceso de fabricación. En concreto, si optamos por una gestión *FIFO*, la que viene establecida por defecto, aquellas piezas que entraron primero en el almacén serán las que se mecanicen en primer lugar (First Input First Output). Por el contrario, si la opción elegida es *LIFO*, serán las últimas piezas introducidas las primeras en ser mecanizadas.
- Un control *TPageControl* formado por dos páginas que nos permitirán conocer, en todo momento, el estado en el que se encuentra el proceso de fabricación.
- La ventana también posee un menú. En el sólo encontraremos las opciones *Comenzar Ejecución* y *Detener Ejecución*, cuya función resulta evidente. Estos elementos están estrechamente relacionados con el botón situado en la parte inferior, el cual, también nos permite comenzar la ejecución o detenerla según se encuentre el proceso: comienzo la ejecución cuando esté detenida, y la detengo cuando esté en activo.

5.8.2. Ejecución.

El primer paso encaminado hacia el comienzo de la ejecución consiste en seleccionar la tarea a ejecutar. Una vez seleccionada, tanto el layout como los datos de la tarea se muestran en sus controles correspondientes. Después, toca el turno de elegir el modo de gestión del almacén, el cual puede ser, como ya sabemos, de tipo *FIFO* o *LIFO*.

Tras estos breves pasos ya puedo comenzar la ejecución de la tarea presionando la opción comenzar Ejecución del menú, o bien haciendo clic sobre el botón de la parte inferior de la ventana. Sin embargo, resulta aconsejable comprobar previamente que no existen palés sobre la cinta, dado que su presencia podría provocar fallos durante la ejecución.

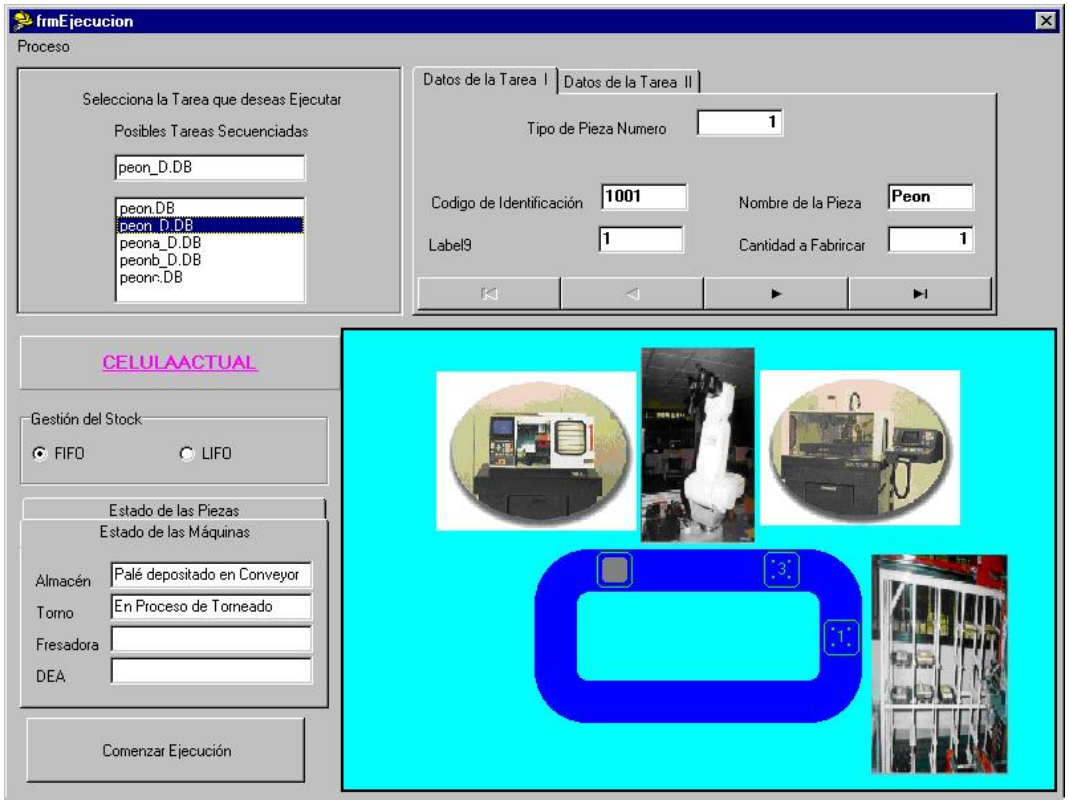


Figura 5.19. *Apariencia de la ventana Ejecutar Tarea durante la ejecución.*

5.9. OTRAS HERRAMIENTAS.

A continuación se describen el resto de herramientas a las que nos proporciona acceso el programa *FMC*.

5.9.1. MEditor. Interfaz de múltiples documentos.

Esta aplicación es capaz de trabajar con múltiples documentos de texto simultáneamente, mediante el uso del MDI (Multiple Document Interface), nombre con el que se conoce a un conjunto de servicios Windows que facilitan esta técnica.

Para comenzar una sesión de trabajo en este MultiEditor tan solo hay que hacer clic en la opción *Editor MDI* del menú principal *Herramientas* del programa *FMC*. Unos instantes después aparece la ventana principal de la aplicación (figura 5.20.), que servirá como marco de trabajo para todas las demás ventanas.

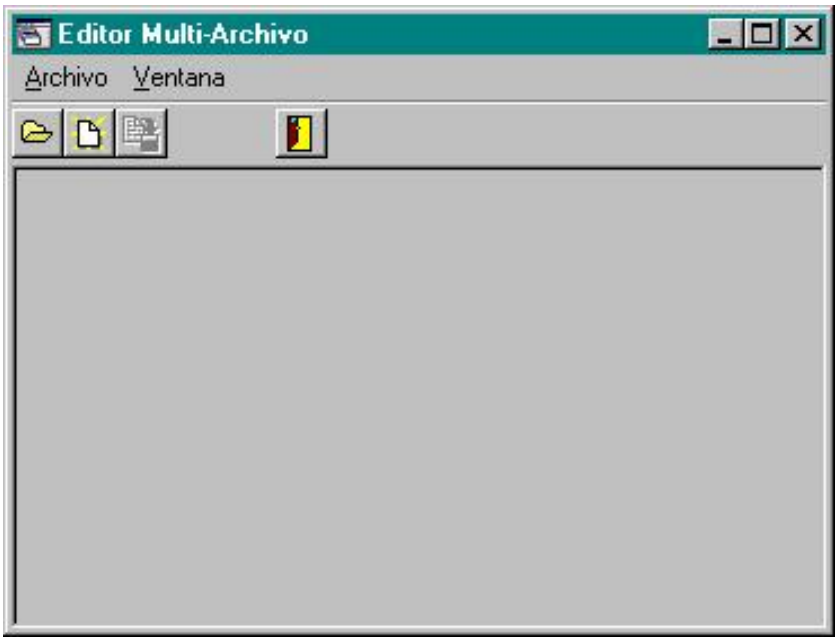


Figura 5.20. *Apariencia inicial de la ventana principal de la aplicación Meditor.*

Tal y como hemos comentado, dentro de esta *Ventana Principal,* o ventana marco, aparecerán las ventanas en las que visualizarán los documentos; estas ventanas se conocen con el nombre de *Ventanas Hija*. Estas fichas pueden ser redimensionadas, minimizadas y maximizadas, ajustándose siempre a los límites de la ventana padre (véase figura 5.21.). Por ejemplo, al minimizar un documento el icono de la ventana no aparece en el escritorio de Windows, sino en el interior de la ventana padre.

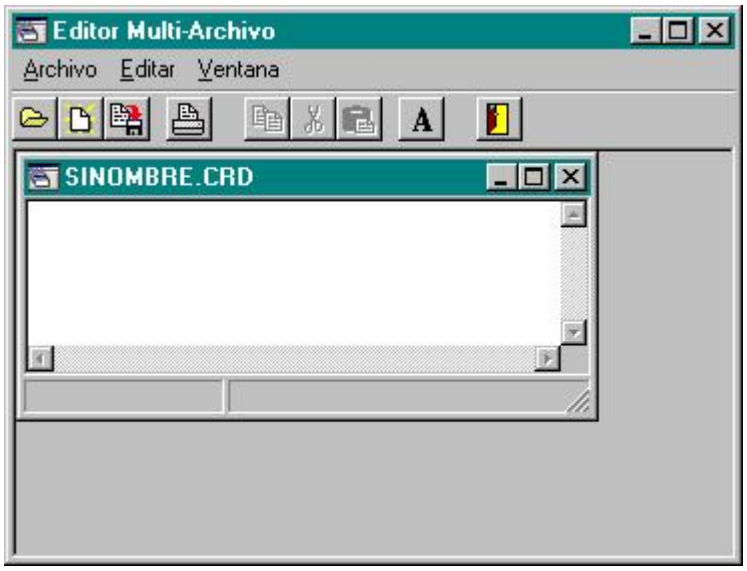


Figura 5.21 *Ejemplo de una ventana hija.*

Antes de proceder con la descripción de la interfaz y los controles del programa, así como de su utilización, debemos comentar que esta aplicación está creada para editar los distintos archivos de texto utilizados a lo largo del proyecto. En concreto, nos estamos refiriendo a:

- Archivos de programa y archivos de puntos del robot mitsubishi.
- Programas residentes del autómata.
- Programas de control numérico y archivos con los offsets de torno y fresadora.
- Programas, archivos de puntos y configuraciones del robot scorbot.

La ventana principal.

El aspecto inicial del programa MEditor es el que se ha mostrado en la figura 5.20. Está formada por:

- Un *menú de opciones* formado por dos elementos: Archivo y Ventana.
- Una *barra de botones*. Inicialmente, esta barra sólo contiene los botones relacionados con las opciones del menú Archivo.
- El *área de edición*. En esta zona se insertarán las diferentes ventanas hijas.

Sin embargo, al introducir una ficha hija la apariencia de la ventana cambia y tanto el menú de opciones como la barra de botones ven incrementados sus elementos (figura 5.21.). A continuación se procederá a describir brevemente los controles de los que disponemos.

Opciones.

- **Archivo |** *Abrir (Ctrl+A).* Esta opción nos permite editar archivos ya creados. Presionando dicho botón daremos paso al cuadro de diálogo mostrado en la figura 5.22.

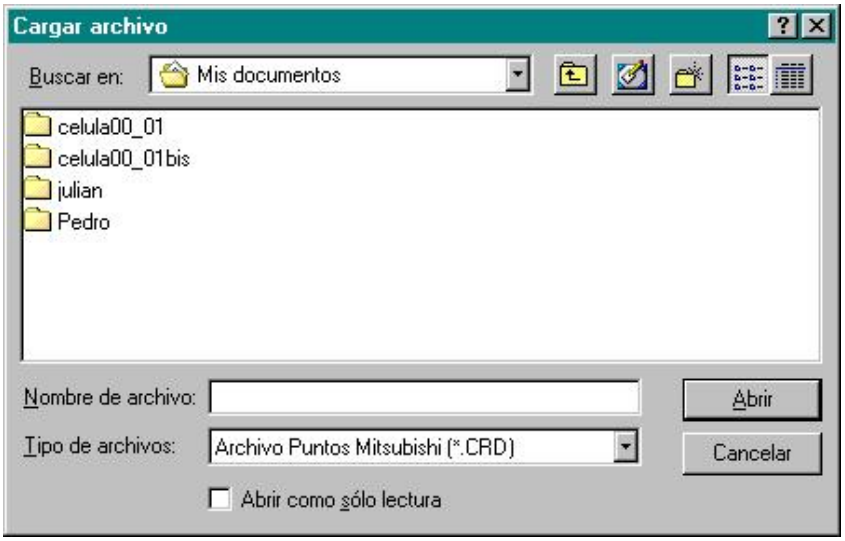


Figura 5.22. *Cuadro de diálogo Cargar Archivo.*

En esta ventana, el primer paso que debemos dar consiste en seleccionar el tipo de archivo que queremos abrir. Una vez seleccionado modificaremos la carpeta actual hasta encontrar el archivo que queremos abrir, haciendo doble clic sobre él. También podemos introducir directamente el nombre del archivo tecleándolo y presionando seguidamente el botón Abrir. Inmediatamente después, el archivo elegido aparecerá en la zona de edición de la ventana principal dentro de una ventana hija. Como ya sabemos, esta ventana hija puede ser redimensionada como mejor nos convenga.

- **Archivo** | *Nuevo (Ctrl+N).* Gracias a este elemento vamos a poder crear un nuevo archivo. El cuadro de diálogo que aparece a continuación nos permite elegir entre los distintos tipos de archivo existentes en el proyecto.

- **Archivo** | *Guardar como (Ctrl+S)*. Es el elemento diseñado para realizar la operación complementaria a la recuperación de un archivo, es decir, nos permite la grabación de los archivos creados o modificados por el usuario. El aspecto del cuadro de diálogo Guardar Como es muy similar al correspondiente a la opción Abrir.

- **Archivo** | *Imprimir (Ctrl+I).* Esta opción nos permite imprimir los archivos a través de la impresora que determinemos. El usuario puede elegir entre imprimir la selección actual, un determinado número de páginas o bien todo el documento.

- **Archivo** | *Salir*. Este es el elemento encargado de finalizar la sesión y, por lo tanto, de devolver el control al programa principal FMC. Antes de salir definitivamente, el programa nos pide confirmación para guardar los archivos que no hayan sido grabados previamente.

En el menú **Editar** encontramos varios elementos destinados a la edición de archivos: *Copiar (Ctrl+C), Cortar (Ctrl+X), Pegar (Ctrl+P) y Fuente (Ctrl+F)*. Este último elemento nos permite seleccionar el tipo y estilo de letra, tal y como se muestra en la figura 5.23.

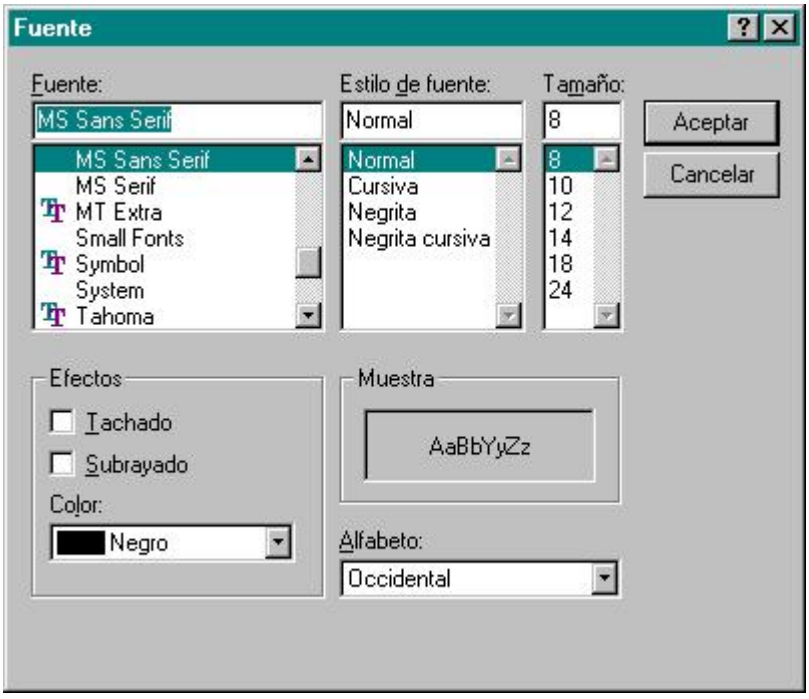


Figura 5.23. *Opciones del cuadro de diálogo Fuente.*

Por último tenemos las opciones relativas al menú **Ventana**:

- *Cascada y Mosaico*. Permiten modificar la posición y dimensiones de todas las ventanas hija que en ese momento se encuentren abiertas, disponiéndolas en forma de cascada, unas sobre otras con un ligero desplazamiento horizontal y vertical, o en forma de mosaico, de tal forma que se ajustan los tamaños para que todas las ventana se visualicen simultáneamente.
- Las ventanas hijas que se encuentran minimizadas en el espacio de la ventana principal pueden ser ordenadas mediante la opción *Ordenar Iconos*.
- Las opciones *Minimizar Todas, Maximizar Todas* y *Cerrar Todas* nos permiten, respectivamente, minimizar, maximizar y cerrar todas las ventanas existentes en el espacio de la ventana principal.

El menú Ventana se completa con una lista de todas las ventanas hija existentes en ese momento, facilitando así la activación de una cierta ventana de forma inmediata.

5.9.2. Paint. Edición de imágenes.

Para la creación y manipulación de imágenes de Mapas de Bits no se ha creado un software específico, sino que se ha optado por la utilización de un programa conocido por todos: Paint de Microsoft. Estamos hablando de un programa sencillo y muy común, pues ha aparecido en todas las versiones de Windows. Por este motivo no vamos a explicar su funcionamiento.

El aspecto que presenta este editor de imágenes es el que aparece reflejado en la figura 5.24.

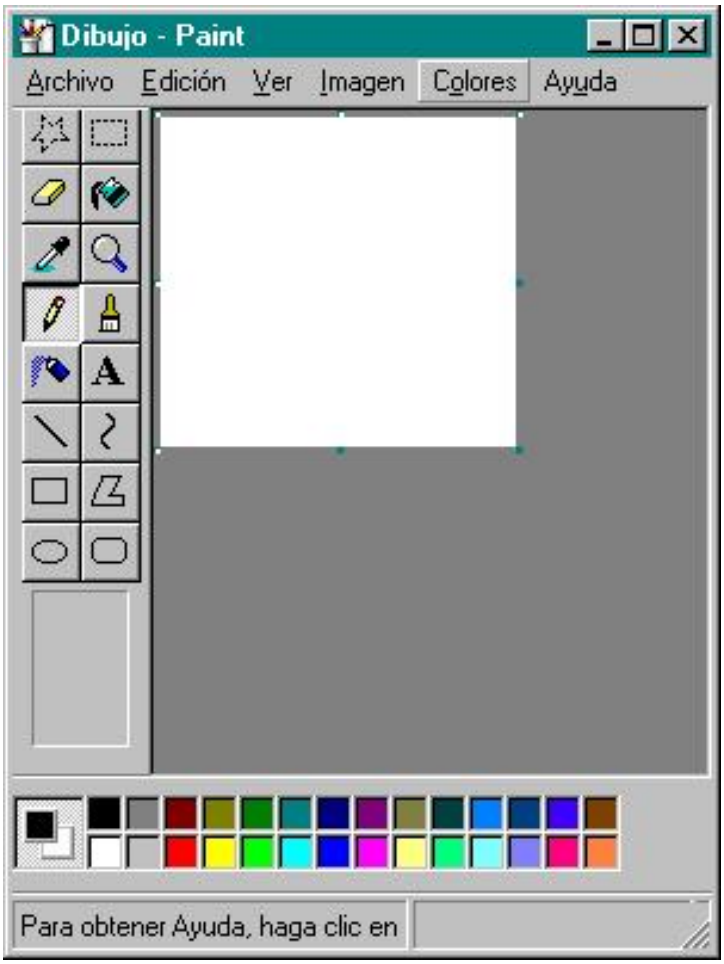


Figura 5.24 *Imagen del programa Paint de Microsoft.*

5.9.3. Database Desktop Manager. El gestor local de bases de datos.

Está claro que la gestión de las bases de datos ha jugado un papel muy importante en la ejecución de este proyecto. En este sentido, la mayor parte de las tablas utilizadas han sido creadas o editadas mediante el programa *Database Desktop Manager*, o *Gestor Local de Bases de Datos*, al que nos referiremos en adelante como *DBD*.

DBD es uno de los programas que acompañan a C++ Builder. Por este motivo, tan sólo analizaremos algunos aspectos relevantes del mismo, ya que cualquier manual de programación de C++ analiza esta aplicación con mayor detenimiento.

La ventana principal.

En la figura 5.25. puede verse el aspecto inicial del *DBD*, en cuya ventana principal podemos destacar los tres botones que nos permiten abrir una tabla, una consulta *QBE* o una sentencia *SQL*, así como el menú principal, del cual analizaremos algunas de sus opciones.

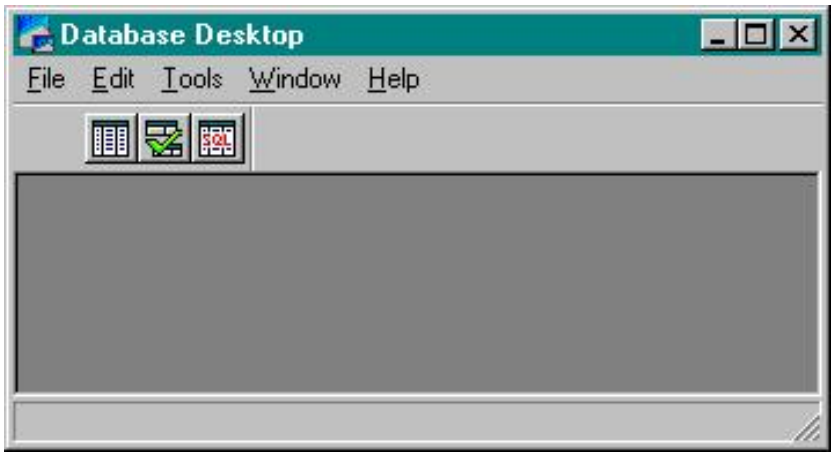


Figura 5.25. *Aspecto inicial del Gestor Local de Bases de Datos.*

Gestión de alias.

Tanto si vamos a proceder con la creación de una base de datos como si lo que pretendemos es trabaja con algunas ya existentes, habitualmente el primer paso que daremos será fijar los directorios de trabajo, con el fin de evitar tener que estar especificando una y otra vez los caminos completos en los que se encuentran o donde se van a alojar las mencionadas bases de datos.

En este sentido, *DBD* nos permite utilizar *alias* o *motes* cuya finalidad es asignar un nombre al camino en el que se encuentra la base de datos, de tal forma que sea posible referirse a ella mediante este alias. Esto, además de facilitar la referencia a una base de datos, ya que tan sólo es necesario recordar su alias, nos permite en cualquier momento mover la base de un sitio a otro, siendo necesario tan sólo redefinir el alias.

En el menú *Tools* del DBD encontramos una opción, llamada *Alias Manager*, que da paso a la ventana mostrada en la figura 5.26., denominada *Gestor de Alias*.

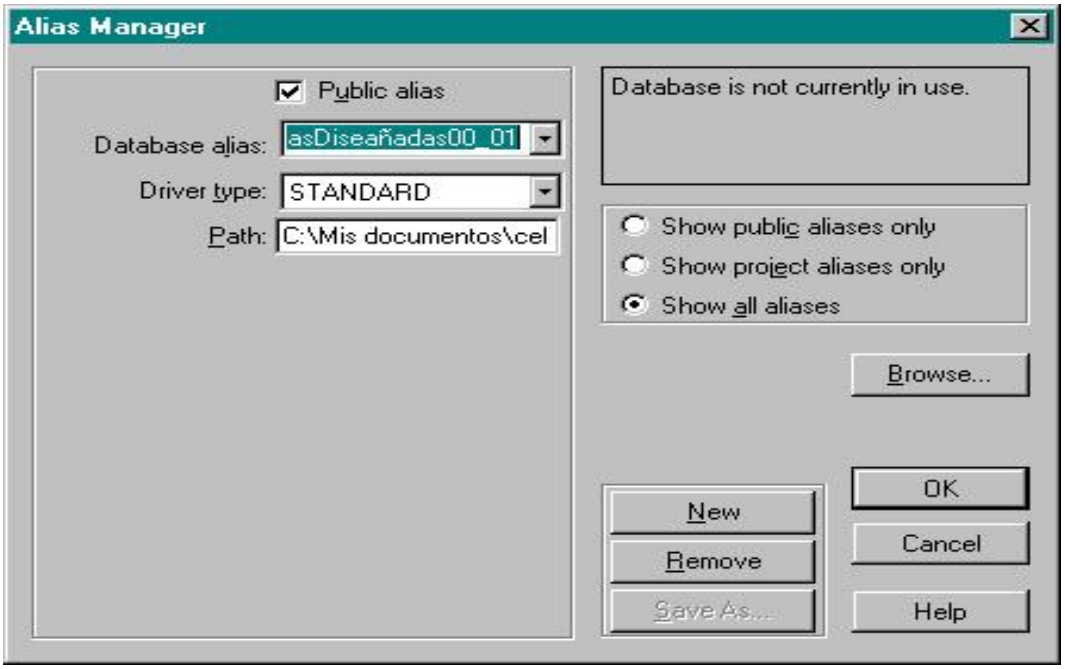


Figura 5.26. *Ventana del Gestor de Alias.*

En ella podemos crear nuevos alias, modificar datos de los ya existentes y borrar alias que no nos sean útiles.

En la tabla 5.5. se muestran todos los alias empleados durante el desarrollo del software de control de la célula.

Alias	Ruta de Trabajo.
AsignarPuertos00_01	C:\Celula00_01\Tablas\AsignarPuertos
Tareas00_01	C:\Celula00_01\Tablas\TareasDiseñadas
TareasSecuenciadas00_01	C:\Celula00_01\Tablas\TareasSecuenciadas
TareaIntermiedia00_01	C:\Celula00_01\Tablas\TareaIntermedia
Posicionadores00_01	C:\Celula00_01\Tablas\Posicionadores
MaterialesFabricados00_01	C:\Celula00_01\Tablas\MaterialesFabricados
MaterialesBase00_01	C:\Celula00_01\Tablas\MaterialesBase
MaterialActualCelda00_01	C:\Celula00_01\Tablas\MaterialActualCelda
Celulas00_01	C:\Celula00_01\Tablas\CelulasDiseñadas

Tabla 5.5. *Alias usados en el proyecto.*

Creación de una tabla.

En el menú *File* del *DBD* encontramos una opción, llamada New, que da paso a tres opciones más que nos permiten crear una tabla, una consulta QBE o una sentencia SQL.

Si seleccionamos la opción *Table* aparece un cuadro de diálogo similar al de la figura 5.27. en el que indicaremos el tipo de la tabla a crear. Aunque *DBD* tiene varios tipos de tablas disponibles, nosotros elegiremos la opción *Paradox7*, ya que todas las tablas desarrolladas en este proyecto son de este tipo.

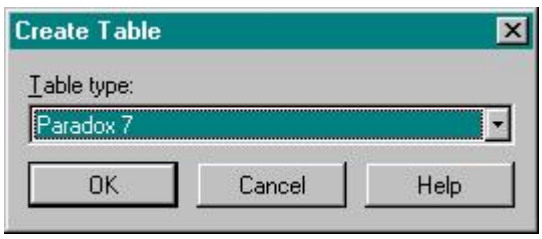


Figura 5.27. *Definición de la estructura de la tabla.*

Una vez hayamos indicado el tipo *Paradox* y pulsado el botón *OK*, en pantalla aparecerá una ventana similar a la mostrada en la figura 5.28., en la que debemos indicar cada uno de los campos o columnas de la tabla, indicando su nombre, tipo, longitud y cualquier dato adicional.

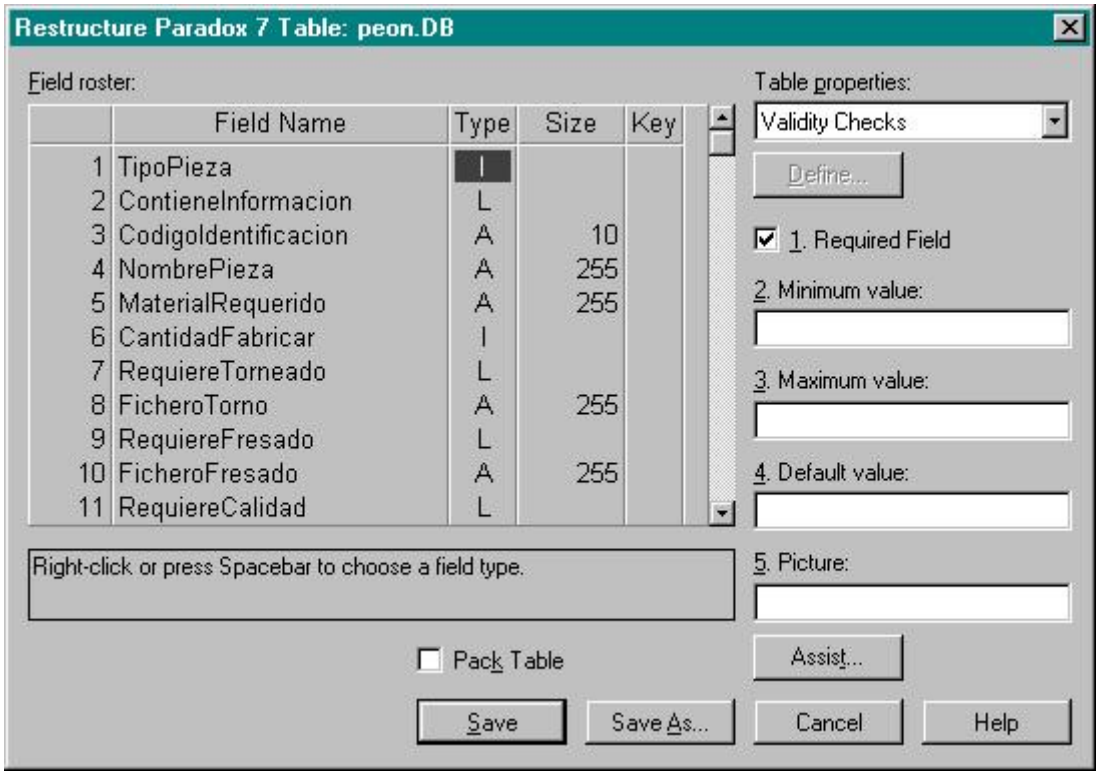


Figura 5.28. *Definición de la estructura de la tabla.*

La forma más fácil de indicar el tipo de un campo consiste en pulsar la barra espaciadora o el botón derecho del ratón, provocando así la apertura de una lista en la cual podremos seleccionar uno de entre todos los tipos disponibles.

Cuando hayamos finalizado la definición de nuestra tabla tendremos que guardarla en el disco. Tras pulsar el botón *Save as* se mostrará una ventana en la que indicaremos el camino de la base (o el alias) y nombre de la tabla.

Si hemos activado la opción *Display Table* en la ventana de grabación, la nueva tabla se abrirá de forma inmediata, permitiéndonos así trabajar ya sobre ella, por ejemplo añadiendo datos.

Editar datos.

Teniendo abierta una tabla, una de las operaciones más simples y habituales que podemos llevar a cabo es la edición de datos, añadiendo nuevas filas o bien modificando el contenido de las ya existentes. Para ello lo primero que tendremos que hacer será activar el modo de edición, pulsando sobre el botón correspondiente, seleccionando la opción *Edit Data* del menú *Table*, o simplemente pulsando la tecla F9.

La navegación por los datos que contiene la tabla está facilitada por el conjunto de botones existentes en la barra. Obviamente también podemos utilizar el teclado o el ratón para realizar estos desplazamientos.

Durante la edición de los datos, basta con desplazarnos al final de la tabla para entrar en el modo inserción, añadiendo nuevas filas a la tabla. También podemos insertar una nueva fila entre dos ya existentes pulsando la tecla *Insert* o usando la opción *Insert* del menú *Record*. La eliminación de un registro, seleccionado en ese momento, puede hacerse mediante la opción *Delete* de ese mismo menú, o bien pulsando la combinación *Control+Supr*.

CAPÍTULO SEXTO:

ESTRUCTURA DEL PROGRAMA 'FMC'. MANUAL DE PROGRAMACIÓN.

6.1. INTRODUCCIÓN.

El siguiente capítulo pretende ser una descripción de los aspectos más relevantes del código desarrollado en este proyecto. El objetivo es servir de guía a posteriores mejoras o actualizaciones.

La herramienta de desarrollo empleada ha sido C++ Builder 4 de Borland. El lenguaje empleado por C++ Builder es, precisamente, C++, que se caracteriza por ser un lenguaje sencillo, flexible y, ante todo, potente.

El siguiente capítulo está estructurado de la siguiente forma:

- Primero hablaremos sobre Windows y las ventajas de su entorno gráfico. Windows es la primera evolución hacia el desarrollo más sencillo de aplicaciones.
- Analizaremos brevemente C++ Builder mostrando sus principales ventajas frente a otras herramientas de desarrollo rápido y visual, que son, a fin de cuentas, las razones de su elección para este proyecto.
- Mostraremos la estructura de directorios empleada en este programa: es necesario conocer dónde están los diferentes archivos empleados en este proyecto.
- Y, finalmente, analizaremos aquellas partes o funciones del código creado que más importancia o relevancia posean. Nos centraremos particularmente en aquellas partes relacionadas con la comunicación.

6.2. PROGRAMACIÓN EN WINDOWS.

La informática en los últimos años a evolucionado a un ritmo trepidante, en una carrera cuya meta es la aceptación del ordenador en el hogar como un elemento más. En esta carrera el software representa un papel fundamental.

Si en los primeros años de los ordenadores personales el usuario debía ser prácticamente un experto, adaptándose a los programas y salvando las limitaciones propias de ellos, en la actualidad la situación se ha invertido, siendo los programas los que se adaptan a los usuarios, llegando a una facilidad de uso que hace posible que una persona sin experiencia los utilice.

En las aplicaciones actuales ha cobrado una especial importancia la interfaz de usuario, elemento que en programas de hace unos años no era el aspecto de más cuidado. Es esta interfaz la que facilita el trabajo del usuario con el programa haciéndolo no solamente más fácil y accesible sino también más cómodo y agradable. En sistemas operativos basados en texto, como DOS, la creación de esas complejas interfaces de usuario, con múltiples ventanas redimensionables, uso de ratón, botones de diferentes tipos, etc., supone para el programador una cantidad de trabajo muy importante, en muchas ocasiones superior a la de la programación del motor del programa. La aparición de entornos y sistemas operativos con interfaz gráfica, como Windows, facilitan en gran medida el diseño de las aplicaciones, ya que muchos de los elementos mencionados antes son gestionados por el propio sistema.

El hecho de que un sistema cuente con una interfaz gráfica, como es el caso de Windows 98 y Windows NT, conlleva que el programador sepa trabajar con una serie de objetos que no existen en los sistemas basados en texto. No cabe duda de que mostrar un botón gráfico, sobre el que el usuario pueda pulsar con el ratón, es un trabajo mucho más simple en Windows que en DOS, siempre y cuando conozcamos las funciones precisas de la API[5] de Windows.

[5] Application Programming Interface / Interfaz de Programación de Aplicaciones.

Inicialmente el desarrollo de aplicaciones para Windows se basaba en el conocimiento, por parte del programador, de las funciones y mensajes de la API de Windows, que se cuentan por cientos. El lenguaje prácticamente obligado era C, en gran medida porque el propio Windows estaba programado en este lenguaje, siendo el más apropiado y flexible a la hora de realizar llamadas a las funciones.

Windows 9x y NT son sistemas que van ganando en complejidad a medida que aparecen nuevas versiones, ya que cada vez cuentan con un mayor numero de posibilidades. Esto se traduce en el aumento de objetos, funciones, interfaces y mensajes que el programador debe conocer, provocando que el programador deba estar en continuo proceso de aprendizaje y recodificación de sus aplicaciones si quiere estar siempre aprovechando las últimas novedades. Este aumento de la complejidad en el desarrollo de las aplicaciones ha provocado que muchos abandonasen el lenguaje C para utilizar C++, ya que éste, gracias a sus características de orientación a objetos, facilita la reutilización de objetos y código, consiguiendo un ahorro de trabajo.

Todos los compiladores C++ existentes para Windows incorporan alguna jerarquía de clases, como OWL[6] o MFC[7], que facilitan en gran medida la creación de nuevas aplicaciones. Estas jerarquías o marcos de aplicación son decenas de miles de líneas de código ya escritas y probadas, preparadas tan sólo para que nosotros las usemos según nuestras necesidades. Este código se estructura en múltiples clases de objetos, mediante los cuales se simplifica el uso de elementos como los botones o las ventanas. Al disponer de un marco de aplicación prefabricado, el programador no tiene necesariamente que conocer toda la API de Windows, ya que la mayoría de los elementos que necesita los encuentra en forma de objetos que tan sólo tiene que crear y utilizar.

[6] Object Windows Library, de Borland.
[7] Microsoft Foundation Classes, de Microsoft.

6.3. HERRAMIENTAS RAD. C++ BUILDER.

6.3.1. Herramientas RAD.

El siguiente paso en el avance hacia un desarrollo más fácil de aplicaciones Windows lo constituye la aparición de las herramientas de desarrollo rápido o visual, conocidas como herramientas RAD, cuyos exponentes más conocidos son Borland Delphi, de Inprise, y Visual Basic, de Microsoft. La escritura de aplicaciones mediante este tipo de entornos se basa en el uso de componentes o controles prefabricados, que son dispuestos en ventanas y personalizados mediante propiedades. De esta forma, la creación de la interfaz de usuario pasa de ser un tedioso trabajo exclusivamente de escritura de código a unas simples operaciones de ratón y poco más. El programador puede volver a centrarse en el núcleo del programa, en el tratamiento de la información, y no necesita preocuparse de la gestión de la ventana en la que se visualizan los datos, ni de la forma en que el usuario se puede desplazar de un lugar a otro con el ratón, por poner un ejemplo.

A pesar de la simplicidad que denotan este tipo de entornos, hay que tener en cuenta que siempre será necesario escribir algo de código para que una aplicación realice una función útil. El código necesario para realizar determinadas funciones de la aplicación estará escrito en un determinado lenguaje, que será la base del entorno de desarrollo visual que estemos usando. El lenguaje base de Visual Basic es BASIC, el de C++ Builder es C++,... El lenguaje es precisamente lo que diferencia de manera fundamental a un entorno RAD de los demás, haciéndolo más o menos potente y flexible. El lenguaje BASIC es de gran simplicidad, ideal para el programador novel, y por ello el elegido por muchos usuarios, pero a cambio carece de la potencia y flexibilidad de C++.

C++ es actualmente el lenguaje orientado a objetos más usado, tanto en Windows como en otros sistemas operativos y otras plataformas, siendo flexible, potente, transportable y conocido por muchos programadores.

6.3.2. C++ Builder.

C++ Builder surgió de la fusión de dos tecnologías en un único entorno de desarrollo. Por una parte estaba el compilador de Borland C++, que soporta el lenguaje C++ estándar con todas sus últimas novedades, y por otra el entorno RAD de Delphi, de conocido éxito. De esta forma, C++ Builder nació siendo una herramienta de desarrollo de última generación sumamente avanzada que se ha ido mejorando en cada versión.

El entorno de desarrollo de C++ Builder es simple, flexible y potente al mismo tiempo, contando con un gran número de componentes prefabricados que facilitan de forma notable la creación de cualquier aplicación. El compilador, por su parte, está altamente optimizado, procesando el código a una velocidad vertiginosa y generando ejecutables de un tamaño razonable.

A diferencia de otros entornos RAD, C++ Builder incorpora un compilador de C++, lo que quiere decir que el código objeto generado es directamente ejecutable, sin necesidad de distribuir archivos adicionales para la interpretación de pseudo-código. Esto tiene básicamente dos consecuencias: el ejecutable es más rápido, al no tener que ser interpretado en tiempo de ejecución, y el tamaño final total de la aplicación suele ser inferior, ya que no es necesaria la distribución adicional del programa que interpreta y ejecuta el código, al que se denomina *run time*. No obstante, C++ Builder permite el uso de paquetes para reducir el tamaño de los ejecutables y permitir que varias aplicaciones compartan librerías.

6.4. ESTRUCTURA DE DIRECTORIOS.

Todo el software generado, así como los archivos, tablas, imágenes, etc., empleados en este proyecto se han distribuido siguiendo la estructura de directorios mostrada en la figura 6.1.

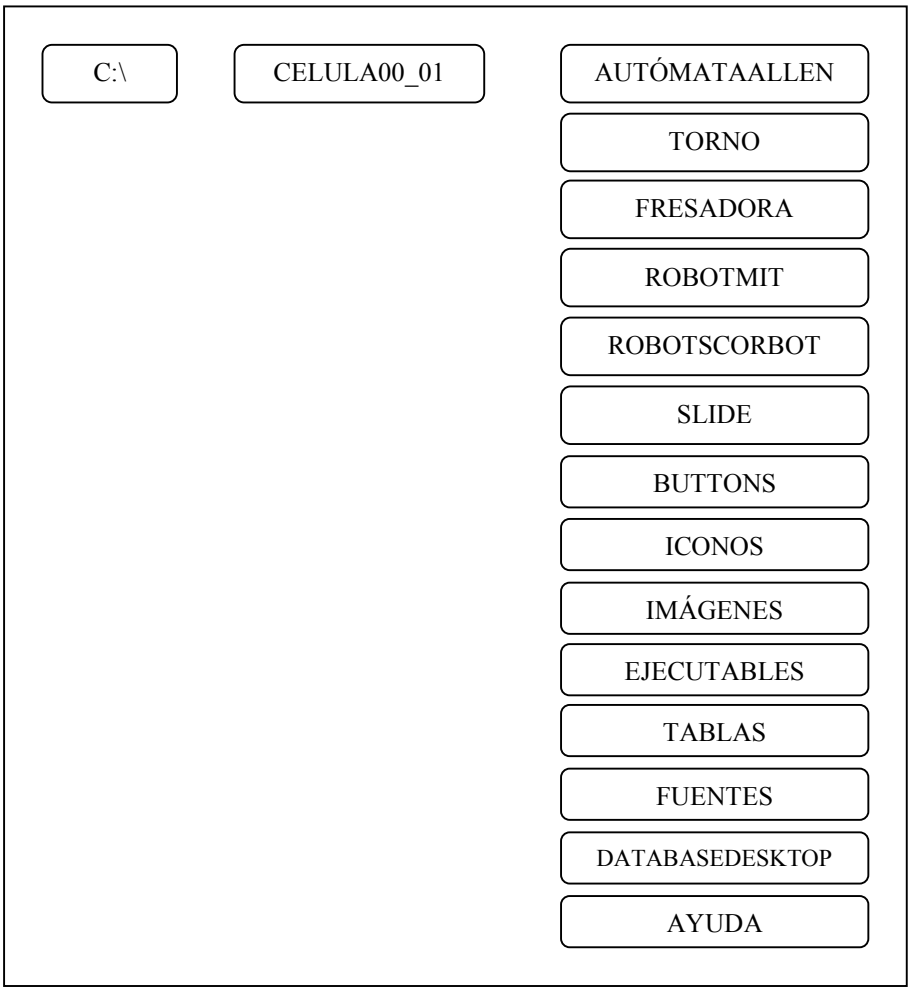


Figura 6.1. *Estructura de directorios de la aplicación FMC.*

A continuación se analiza el contenido de cada uno de estos directorios.

C:\Celula00_01\AutomaAllen.

En este directorio se guardan los ficheros relacionados con el autómata programable "Allen-Bradley" que forma parte de la FMC del LOIP. Contiene ficheros con extensión "*.CIM", es decir, contiene los programas que debemos enviar al autómata para que éste funcione correctamente.

Dentro de este directorio encontramos la carpeta "Buenos", en la que se almacena el programa de funcionamiento actual del autómata, "Residente.cim". Aunque este archivo también se encuentra en el directorio "C:\Celula00_01\ AutomaAllen", hemos optado por guardarlo por duplicado en un archivo diferente. De esta forma, si durante su edición alteramos erróneamente su contenido, o lo eliminamos, siempre sabremos que existe una copia de donde poder extraer el programa[8].

C:\Celula00_01\Torno.

En este archivo se guardan los programas de control numérico empleados por el torno para la mecanización de las diferentes piezas que la célula debe ser capaz de fabricar. Estos archivos se guardan con la extensión "*.MIR". Así mismo, esta carpeta contiene los archivos con los offsets empleados por los programas anteriores (extensión "*.MIO").

C:\Celula00_01\Fresadora.

En este directorio, al igual que ocurría con el directorio correspondiente al torno, se guardan los programas de control numérico que la fresadora emplea durante el proceso de mecanizado. Estos ficheros se guardan con extensión "*.FNC". También podremos encontrar los archivos que contienen los datos relativos a diferentes offsets. Estos archivos se corresponden con aquellos que tienen una extensión "*.FAO".

[8] . Esta lógica de actuación también se ha seguido con aquellas máquinas que necesitan de un programa de funcionamiento, como es el caso del robot Mitsubishi o del eje de deslizamiento.

C:\Celula00_01\RobotMit.

Este directorio es el encargado de almacenar los ficheros que requiere el robot Mitsubishi. En concreto, podemos encontrar dos tipos de archivos:

- *Archivos con extensión **PGM.*** Son los programas que el controlador del robot almacena en su memoria para su posterior ejecución, es decir. Son los programas que se ejecutarán cada vez que presionemos el interruptor *START* del controlador.
- *Archivos con extensión **CRD.*** Estos archivos contienen las coordenadas de puntos. Estas definiciones son las que emplean los archivos de programa (*.PGM) para su correcto funcionamiento.

Para finalizar con el directorio asociado al robot Mitsubishi, sólo nos resta comentar que existe una copia de los archivos necesarios durante el proceso de ejecución integral en el directorio "C:\Celula00_01\RobotMit\Buenos". En concreto, estaríamos hablando de los archivos *R-4pos.pgm* y *R-4pos.crd*.

C:\Celula00_01\RobotScorbot.

Este directorio es el encargado de almacenar los ficheros que requiere el robot Scorbot. En concreto, podemos encontrar tres tipos de archivos:

- *Archivos con extensión **CBU.*** Se trata de los programas que el controlador del robot almacena en su memoria para su posterior ejecución.
- *Archivos con extensión **POS.*** Son los archivos encargados de almacenar las coordenadas de los puntos a los que queremos que llege el elemento terminal del robot. Esta serie de puntos debe definirse para ejecutar correctamente los programas del robot almacenados en los archivos de programa (*.CBU).
- *Archivos con Extensión **CNF**.* Estos archivos almacenan la configuración del robot Scorbot.

C:\Celula00_01\Slide.

Aquí encontramos los archivos de funcionamiento del eje de deslizamiento o slide. En concreto, distinguimos los siguientes archivos:

- ***ControlRemoto.prg***. Este archivo es el que enviamos a la caja negra o controlador del slide cada vez que iniciamos una sesión de control remoto. En él se guardan las instrucciones precisas que nos permitirán controlar remotamente al slide.

- ***Ejecución.prg***. Si al iniciar una sesión de control remoto del eje de deslizamiento enviábamos el archivo *ControlRemoto.prg*, al finalizar dicha sesión el programa *FMC* se encarga de enviar el programa *Ejecución.prg*. Este programa es el que se ejecutará en el proceso de fabricación global.

Por lo tanto, mientras estemos controlando el eje de deslizamiento mediante el programa de control remoto generado en este proyecto, el programa que se ejecutará será el almacenado en *ControlRemoto.prg*, mientras que el resto del tiempo se ejecutará *Ejecucion.prg*.

Actualmente, estos archivos son los mismos que *Mover.prg* y *4pos.prg* respectivamente, es decir, la información almacenada en *ControlRemoto.prg* y *Ejecución.prg* es la misma que contienen *Mover.prg* y *4pos.prg* respectivamente. Este último archivo debe su nombre a que define sobre el slide cuatro posiciones que necesita el programa *R-4pos.pgm* del robot Mitsubishi.

Si en cualquier momento queremos modificar el programa residente en la memoria del controlador tan sólo deberemos copiar en *Ejecución.prg* las nuevas líneas de código, acceder al bloque de programa de control remoto del slide y finalizar la sesión.

C:\Celula00_01\Buttons.

En este directorio disponemos de las diferentes imágenes, en formato *mapas de bits (*.BMP)*, empleadas en los botones de nuestra aplicación.

C:\Celula00_01\Iconos.

Reúne los iconos, en formato *"*.ICO"*, que hemos empleado para representar las diferentes aplicaciones que integran nuestro programa.

C:\Celula00_01\Imagenes.

En este directorio se recopilan las imágenes de los distintos elementos de la célula. Estas imágenes las emplearemos para desarrollar el diseño de la célula, su layout. Todas están en formato de mapas de bits.

C:\Celula00_01\Ejecutables.

Contiene los diferentes archivos ejecutables usados en este proyecto. Todos ellos tienen extensión EXE:

- √ ***FMC.exe***. Se trata del ejecutable principal. Nos permite integrar y controlar todas y cada una de las máquinas de la FMC del LOIP. Desde este bloque de programa podremos acceder fácilmente al resto de aplicaciones ejecutables desarrolladas.
- √ ***MEditor.exe***. Meditor.exe es una aplicación MDI desarrollada para editar de forma rápida los múltiples archivos de texto utilizados (*.pgm, *.cdr, *.cim,...). Se analizó este programa en el capítulo 5.
- √ ***PBrush.exe***. Esta es la aplicación encargada de la edición de archivos de mapas de bits: *Paint* de Microsoft.
- √ ***PDisCelula.exe***. Aplicación destinada a generar el layout de la célula.
- √ ***PDisTarea.exe***. Este archivo ejecutable es el encargado del diseño de las tareas.

- √ ***PDesglosar.exe***. Su utilidad es la de "desglosar" las tareas definidas en el programa PDisTarea.exe para que puedan ser utilizadas por la parte del programa FMC.exe encargada de la ejecución de las tareas.

C:\Celula00_01\Tablas.

En este directorio encontramos las diferentes tablas utilizadas en el programa. No comentaremos nada acerca de ellas, puesto que ya las hemos ido analizando en los capítulos anteriores y porque su gestión la realiza íntegramente el programa FMC. Por lo tanto, el usuario de FMC no necesita conocer en profundidad su estructura y comportamiento.

C:\Celula00_01\Fuentes.

En este directorio se encuentran todos los programas fuente, es decir, todo el código desarrollado en el lenguaje de programación C++ y que hace posible el control de cada una de las máquinas de la célula tanto de forma individual como de forma colectiva. Además de la explicación existente en el propio código de las diferentes rutinas, podremos acceder a una visión global de este código en el apartado 5 de este mismo capítulo.

C:\Celula00_01\DatabaseDesktop.

En este directorio están todos los archivos relacionados con el gestor de bases de datos de Borland que incorpora el programa C++ Builder.

C:\Celula00_01\Ayuda.

Como su propio nombre indica, en este directorio encontramos los archivos de ayuda del proyecto.

6.5. COMENTARIOS ACERCA DEL CÓDIGO DESARROLLADO EN ESTE PROYECTO.

Comenzaremos examinando la estructura que posee el directorio "C:\Celula00_01\ Fuentes", que es el directorio que contiene todo el código fuente del programa.

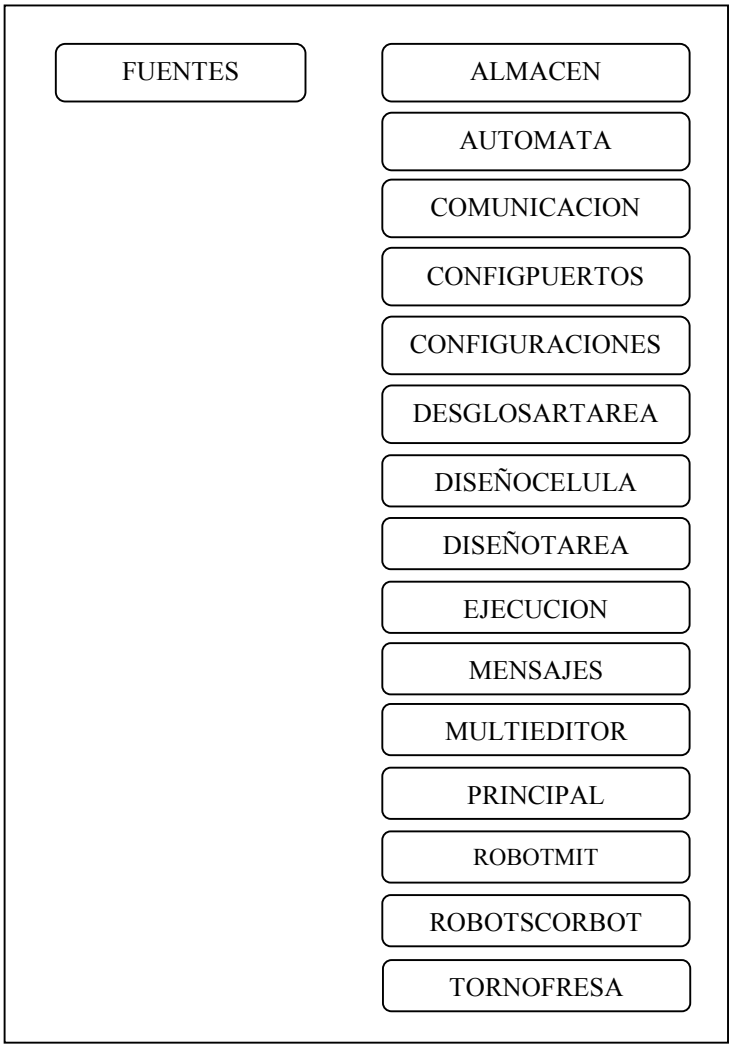


Figura 6.2. *Estructura del directorio"C:\Celula00_01\Fuentes".*

Aunque esta estructura no es realmente necesaria, por cuestiones de limpieza de código y accesibilidad se ha considerado provechoso unir los diferentes programas en bloques relacionados entre sí. En cada uno de estos

directorios encontraremos principalmente uno o varios archivos de proyecto, con extensión BPR, uno o más formularios, cada uno de los cuales tiene asociado un archivo DFM con la definición de la interfaz, un módulo de código con extensión CPP, y un archivo de cabecera con extensión HPP por cada módulo de código. Para distinguir correctamente los archivos de proyecto todos ellos comienzan por la letra P mayúscula, mientras que los módulos de código comienzan con la letra U mayúscula. Por otra parte, los formularios empleados comienzan con las letras frm minúsculas.

En los párrafos siguientes analizaremos los módulos de código más importantes.

6.5.1. El código.

En el directorio "C:\Celula00_01\Fuentes\Principal" encontramos los archivos relacionados con la ventana principal del programa. Tal y como se describió, desde esta ventana vamos a poder acceder a las diferentes opciones de trabajo y configuración. El módulo de código principal a analizar es "UPrincipal.cpp", del que a continuación pasamos a describir algunos aspectos.

Entre las definiciones de variables globales podemos destacar las siguientes

```
HANDLE hcomm[8];
LPCSTR commname[8];
bool PuertoAbierto[8],
```

donde *hcomm* es una matriz que se utiliza para almacenar las direcciones de memoria de los puertos abiertos. *commname*, es una variable utilizada en la asignación de los puertos existentes a las diferentes máquinas de la FMC del LOIP. *PuertoAbierto* tan sólo es una matriz booleana que se utiliza para saber si los puertos asignados están abiertos o no.

ControRemotoExecute es la función encargada de mostrarnos las ventanas desde donde poder controlar individualmente cada una de las máquinas de la FMC. A esta función acceden todos los miembros del menú principal *Control Remoto*, lo que nos proporciona una idea del ahorro de código que se ha conseguido. Cada vez que accedamos a una de las opciones del menú *Control Remoto* se ejecutan las mismas funciones, cuyo funcionamiento depende de una constante denominada Tag, que todo control de C++ posee.

```
void __fastcall TfrmPrincipal :: ControlRemotoExecute ( TObject  * Sender )
{
AnsiString Mensaje;
int Elemento, NumPuerto;
TMsgDlgButtons botones;
Botones << mbYes << mbNo;
Elemento= (( TMenuItem  * ) Sender ) -> Tag;
//Si los puertos ya han sido configurados sólo debo mostrar la ventana correspondiente
switch( Elemento ){
   case 0 : if( PuertoAbierto[0] ) { frmCRAutomataCinta -> Show ( ); return; } break;
   case 1 : if( PuertoAbierto[1] ) { frmCRAlmacen -> Show ( ); return; } break;
   case 2 : if( PuertoAbierto[2] ) { frmCRTorno -> Show ( ); return; } break;
   case 3 : if( PuertoAbierto[3] ) { frmCRFresadora -> Show ( ); return; } break;
   case 4 : if( PuertoAbierto[4] ) { frmCRRobotMit -> Show ( ); return; } break;
   case 5 : if( PuertoAbierto[5] ) { frmCRScorbot -> Show ( ); return; } break;
   case 6 : if( PuertoAbierto[6] ) { frmCRSlide -> Show ( ); return; } break;
   case 7 : if( PuertoAbierto[7] ) { frmCRDea -> Show ( ); return; } break;
}
// si el puerto no está abierto se abre y se configura. Para ello se abre la tabla
// en la que se almacena la información sobre la asignación de los puertos Actual
Screen -> Cursor = crHourGlass;
Tabla1 -> Close ( );
Tabla1 -> Open ( );
Tabla1 -> First();
while ( ! Tabla1 -> Eof ) {  //recupero los datos almacenados en la tabla
      if( Tabla1Configuracion -> Value == "Actual" ) break;
      Tabla1 -> Next ( );
}
```

```
// realizo la asignación de puertos
NumPuerto = Tabla1 -> Fields -> Fields[ Elemento+2] -> Value;
commname[ Elemento] = asignar( NumPuerto );
Tabla1 -> Close ( );
Hcomm [ Elemento] = ConfigurarPuerto( commname [ Elemento], Elemento);//abre el puerto y lo configura
PuertoAbierto[ Elemento] =true;
Screen -> Cursor = crDefault;
// Compruebo si puedo abrir el puerto en cuestión
// si lo puedo abrir inicializo el elemento y configuro el puerto
// si no lo puedo abrir cierro el puerto abierto y genero un mensaje de error
if( ComprobarElemento ( Elemento)){
  InicioElemento( Elemento);
} else {
  CerrarPuerto( hcomm[ Elemento] );
  PuertoAbierto[ Elemento] = false;
  //mostrar mensaje con las posibles causas de la falta de comunicación
  MensajeErrorApertura(Elemento);
}
}
```

A continuación se analizarán alguna de las funciones utilizadas en esta porción de código:

- *ConfigurarPuerto* es una función que se encuentra en el módulo UConfiguracion.cpp en el directorio "C:\Celula00_01\Fuentes\ Configuracion". Se trata del código encargado de abrir y configurar los diferentes puertos serie utilizados.

```
HANDLE ConfigurarPuerto(LPCSTR commname,int puerto){
// Se abre el puerto de comunicaciones, si se produce  algún fallo se avisa de este al usuario ,
// si la operación se efectúa con éxito se configuran los parámetros necesarios para la comunicación
// y se devuelve  la dirección al puerto
HANDLE hcomm;
COMMCONFIG cc;
COMMPROP cp;
char cadena[100];
DCB dcb;
```

```
if((hcomm=CreateFile(commname,GENERIC_READ|GENERIC_WRITE,
  0,NULL,OPEN_EXISTING,0,NULL))==INVALID_HANDLE_VALUE){
  // operación inválida: error en la apertura del puerto
  ShowMessage("ERROR AL ABRIR EL PUERTO\t"+AnsiString(commname));
  return 0;
  }else{
    //obtenemos el tamaño de la estructura Commprop
    cp.wPacketLength=sizeof(COMMPROP);
    // y la información sobre el puerto.
    GetCommProperties(hcomm,&cp);
    //Se llama a la función GetCommState para obtener los DCB settings del puerto.
    GetCommState(hcomm,&dcb);
    //habilitamos el chequeo de paridad
    cc.dcb.fParity=1;
    switch(puerto){
      case 4:  //Procedimiento que configura el puerto de comunicaciones con el robot Mitsubishi :
               // 4.800 baudios, paridad par, 7 bits de datos y 1 bit de parada.
        cc.dcb.fOutX=false;
        cc.dcb.fInX=false;
        //DTR y RTS a ON
        cc.dcb.fDtrControl=DTR_CONTROL_ENABLE;
        cc.dcb.fRtsControl=RTS_CONTROL_ENABLE;
        wsprintf(cadena,"48,e,7,1");
        break;
      case 6:  // Procedimiento que configura el puerto de comunicaciones con el Slide :
               // 9.600 baudios, no paridad, 8 bits de datos y 2 bit de parada.
        cc.dcb.fOutxCtsFlow=false;
        cc.dcb.fOutxDsrFlow=false;
        cc.dcb.fDsrSensitivity=false;
        cc.dcb.fNull=false;
        cc.dcb.fOutX=false;
        cc.dcb.fInX=false;
        cc.dcb.fDtrControl=DTR_CONTROL_ENABLE;
        cc.dcb.fRtsControl=RTS_CONTROL_ENABLE;
        wsprintf(cadena,"96,n,8,2");
        break;
```

```
case 5: // Procedimiento que configura el puerto de comunicaciones con el robot Scorbot :
        // 9.600 baudios, no paridad, 8 bits de datos y 1 bit de parada.
  cc.dcb.fOutxCtsFlow=false;
  cc.dcb.fOutxDsrFlow=false;
  cc.dcb.fDsrSensitivity=false;
  cc.dcb.fNull=false;
  cc.dcb.fOutX=false;
  cc.dcb.fInX=false;

  cc.dcb.fDtrControl=DTR_CONTROL_ENABLE;
  cc.dcb.fRtsControl=RTS_CONTROL_ENABLE;
  wsprintf(cadena,"96,n,8,1");
  break;
case2:
case 3:  // Procedimiento que configura el puerto de comunicaciones del torno y fresadora:
        // 2.400 baudios, paridad par, 7 bits de datos y 1 bit de parada.
  wsprintf(cadena,"24,e,7,1");
  break;
case 0:   // Procedimiento que configura el puerto de comunicaciones con el autómata:
         // 9.600 baudios, paridad par, 7 bits de datos y 1 bit de parada.
  cc.dcb.fOutxCtsFlow=false;
  cc.dcb.fOutxDsrFlow=false;
  cc.dcb.fDsrSensitivity=false;
  cc.dcb.fNull=false;
  cc.dcb.fOutX=false;
  cc.dcb.fInX=false;
  cc.dcb.fDtrControl=DTR_CONTROL_ENABLE;
  cc.dcb.fRtsControl=RTS_CONTROL_DISABLE;
  // Especificamos los nuevos parámetros a introducir
  wsprintf(cadena,"96,e,7,1");
  break;
case 1:  // Procedimiento que configura el puerto de comunicaciones con el almacén:
        // 2.400 baudios ,no paridad, 8 bits de datos y 1 bit de parada.
  cc.dcb.fOutxCtsFlow=false;
  cc.dcb.fOutxDsrFlow=false;
  cc.dcb.fDsrSensitivity=false;
  cc.dcb.fNull=false;
  cc.dcb.fOutX=false;
  cc.dcb.fInX=false;
```

```
        cc.dcb.fDtrControl=DTR_CONTROL_ENABLE;
        cc.dcb.fRtsControl=RTS_CONTROL_DISABLE;
        wsprintf(cadena,"24,n,8,1");
        break;
    case 7:  // Procedimiento que configura el puerto de comunicaciones del centro de calidad:
             // 9600 baudios ,no paridad, 8 bits de datos y 2 bit de parada.
        cc.dcb.fOutxCtsFlow=false;
        cc.dcb.fOutxDsrFlow=false;
        cc.dcb.fDsrSensitivity=false;
        cc.dcb.fNull=false;
        cc.dcb.fOutX=false;
        cc.dcb.fInX=false;
        cc.dcb.fDtrControl=DTR_CONTROL_ENABLE;
        cc.dcb.fRtsControl=RTS_CONTROL_ENABLE;//DISABLE;
        wsprintf(cadena,"96,n,8,2");
        break;
}
    // introducimos los nuevos parámetros mediante una llamada
    // a la función BuildCommState que cambia los settings en DCB
    // mediante una cadena de caracteres que se le pasa como parámetro.
    BuildCommDCB(cadena,&cc.dcb);
    //SetCommState escribe el contenido de la estructura DCB en port settings
    SetCommState(hcomm,&cc.dcb);
    return hcomm;
        }
}
```

- *ComprobarElemento* es una función que nos permite conocer si la máquina asociada al puerto configurado está encendida y en modo comunicación. En caso afirmativo devuelve el valor *true* y genera la llamada a la función que inicializa la máquina.

- *InicioElemento*. Esta función es la encargada de realizar aquellas acciones encaminadas a inicializar las diferentes máquinas. Principalmente se trata de enviar a las máquinas a sus orígenes de coordenadas.

- En caso de que la función *ComprobarElemento* devuelva el valor *false*, la función que se ejecuta es *CerrarPuerto*, que se encarga, como su propio nombre indica, de cerrar el puerto que anteriormente abrimos y configuramos. Después de cerrar el puerto, el programa emite un mensaje de error mediante la función *MensajeErrorApertura*.

Tras conocer la función encargada de configurar los distintos puertos, necesitamos comprender las funciones destinadas a la transmisión y recepción de datos.

Inicialmente se optó por una entrada/salida de datos síncrona, dado que este tipo de transmisión mejora considerablemente la técnica de transmisión polling. Sin embargo, el resultado no fue muy satisfactorio debido, principalmente, a que las diferentes máquinas de la FMC utilizan flujos de control diferentes.

Por este motivo, la comunicación entre el ordenador central y el resto de las máquinas se lleva a cabo mediante unas funciones de transmisión y recepción específicas para cada una de los elementos.

Dado que la transmisión de datos se realiza mayoritariamente mediante el método polling, debemos buscar una solución que nos permita transmitir y recibir datos con independencia de la trama a enviar o recibir. Con este propósito se recurre a las funciones *EnviaCaracter* y *RecibeCaracter* que transmiten y reciben, uno a uno, los caracteres de cada trama.

```
bool EnviaCaracter(char c,HANDLE puerto)
// Devuelve true si ha enviado correctamente
{
DWORD actual_escrito;
char buffer_salida;
bool envia=true;
COMMTIMEOUTS to;
// cuando es necesario conocer si existe comunicación se ejecutan las siguientes líneas
```

```
if(comprueba){
  memset(&to,0,sizeof(to));
  to.WriteTotalTimeoutMultiplier=5;
  to.WriteTotalTimeoutConstant=500;
  SetCommTimeouts(puerto,&to);
  comprueba=false;
}
buffer_salida=c;
// intento escribir en el puerto un único carácter y si no puedo proceso el error y devuelvo el control.
if(!WriteFile(puerto,&buffer_salida,1,&actual_escrito,NULL)){
      envia=false;
      locProcessCommError(GetLastError(),puerto);
}
return envia;
}
//---------------------------------------------------------------------------
char RecibeCaracter(HANDLE puerto)
{
DWORD actual_leido;
char c,buffer_entrada[1];
bool error=false;
COMMTIMEOUTS to;
memset(&to,0,sizeof(to));
to.ReadIntervalTimeout=MAXDWORD;
SetCommTimeouts(puerto,&to);
// intentamos leer un único carácter del puerto
if(!ReadFile(puerto,buffer_entrada,1,&actual_leido,NULL)){
      //si no se puede leer gestionamos el error y devolvemos el carácter \0
      locProcessCommError(GetLastError(),puerto);
      error=true;
      return '\0';
      }else{//si hemos logrado leer devolvemos el carácter leído.
            if(actual_leido){
                  c=buffer_entrada[0];
                  return c;
                  }
           }
if(error)ShowMessage("No Hemos Podido Leer");
return '\0';
```

Respecto a las funciones de comunicación de las diferentes máquinas, cabe resaltar la función de transmisión del robot Mitsubishi, en la que se emplea el método de lectura mediante eventos.

```
bool EnviaRobot(HANDLE puerto,bool mensaje,char Comando[],int Num)
//Devuelve true si se ha enviado correctamente al robot y false en caso contrario
{
DWORD dwEvent;
int flag;
bool envia=true;
for(int i=0;i<Num;++i)
      if(!EnviaCaracter(Comando[i],puerto))envia=false;
if(!EnviaCaracter('\r',puerto))envia=false;
if(!envia){//si se produce un error en la transmisión esta se interrumpe y se envía un mensaje indicándolo
      ShowMessage("error al intentar escribir");
      return envia;
      }


if(mensaje){ // si se requiere lectura mediante eventos. Se emplea para conocer cuando
            // ha terminado de ejecutarse una determinada orden.
  Screen->Cursor=crHourGlass;
  SetCommMask(puerto,EV_DSR);
  SetCommMask(puerto,EV_CTS);
  while(true){
   if(WaitCommEvent(puerto,&dwEvent,NULL))flag=1;
    if(flag==1)
     if(WaitCommEvent(puerto,&dwEvent,NULL)&&
           WaitCommEvent(puerto,&dwEvent,NULL))break;
  }
PosicionRobotMit(puerto);//acutaliza la posición que ocupa el robot en la imagen simulada
Screen->Cursor=crDefault;
}
return envia;
}
```

Para finalizar el capítulo destinado al código, vamos a tratar el código asociado al módulo *UEjecucion.cpp*. En concreto nos centraremos en las funciones más relevantes que controlan el proceso de fabricación.

Cuando presiono el botón *Comenzar Ejecución*, o la opción del menú principal que tiene el mismo nombre, se ejecuta la siguiente función.

```
void __fastcall TfrmEjecucion::EjecutarTarea1Click(TObject *Sender)
{
int i,j;
AnsiString Cadena;
// compruebo que en el almacén tengo las piezas necesarias para poder llevar a cabo
// la fabricación de las piezas de la tarea. En caso negativo detengo el proceso.
if(!ComprobarStock()){
  ShowMessage("El almacén contiene menos piezas de las requeridas");
  return;
}
//inicializar las variables (menos NumTotalPiezas)
InicializaVariables();
// configuro los  puertos que van a ser utilizados
for(i=0;i<7/*8*/;++i)
  if(!ConfigurarLosPuertos(i)){
    //  si se a producido un error en la configuración muestro un mensaje indicándonoslo
    ShowMessage("error al configurar los puertos");
    return;
  }
//Inicializo al autómata. No pongo en run al automata porque ya lo he hecho antes
for(i=1;i<5;++i){
  Estacion[i].Coil=0;
  for(j=1;j<9;++j){
    Estacion[i].EstadoActual[j]=0;
    Estacion[i].EstadoAnterior[j]=0;
    Estacion[i].Coil+=Estacion[i].EstadoActual[j]*pow(2,j-1);
  }
EscribirAutomata(hcomm[0]);
//habilito el reloj que iniciará la ejecución
Reloj->Enabled=true;
}
```

Si el almacén tiene las piezas necesarias y los puertos requeridos han sido abiertos y configurados correctamente, es el momento de comenzar realmente el proceso de fabricación. Para ello disponemos de un control *TTimer* que se ejecuta cada dos segundos.

```
void __fastcall TfrmEjecucion::RelojTimer(TObject *Sender)
{
int i,j;
if(Ejecutar){
 frmCRAutomataCinta->LeeEstadoCinta();
 //actualizo las posiciones de los palés
 ActualizarCinta();
 // actualizo el estado anterior de los pernos de restricción, pernos posicionadores y límites de entrevía.
 for(i=1;i<5;++i)
  for(j=1;j<9;++j)
    Estacion[i].EstadoAnterior[j]=Estacion[i].EstadoActual[j];
 // Si he puesto un palé en la cinta, espero 4 segundos a poner el siguiente.
 // Esta medida se encamina a evitar un choque del palé con los límites de entrevía,
 // puesto que el palé tarda 3 segundos en salir de la estación.
 if(espera)TiempoEsperaAlmacen++;
 if(TiempoEsperaAlmacen>=2){TiempoEsperaAlmacen=0;espera=false;}
 //evalúo el estado de las señales de cada estación.
 Host();
 //escribo en el automata para modificar el estado de las estaciones en función de las señales recibidas
 EscribirAutomata(hcomm[0]);
 //compruebo la condición de parada
 if(PiezasQuitadas==NumTotalPiezas)fin=true;
}
if(fin){//si se ha finalizado el proceso deshabilito el reloj
 Reloj->Enabled=false;
 DetenerEjecucion1->Enabled=true;
}
}
```

Sin duda alguna, el elemento más importante al que se llama desde la función anterior es *Host*. El código desarrollado en *Host* se encarga de gestionar las llamadas a otras funciones, y lo hace de acuerdo al estado en que se encuentren los relés de cada una de las cuatro estaciones del conveyor.

```
void TfrmEjecucion::Host(void)
{
//analiza lo que ocurre en las cuatro estaciones
for(int i=1;i<=4;++i){
  // Si la estación está inactiva dejo pasar el palé cuando éste llega a la misma
  if(!PActivo[i+1]){
    if(est[i]){
        DibujarPallet(Pos[i]+1,Pos[i]);
        est[i]=false;
    }else{
      if(Estacion[i].EstadoAnterior[4]&&Estacion[i].EstadoAnterior[6]&&
        !Estacion[i].EstadoAnterior[3]){
        Estacion[i].EstadoActual[6]=0;
        Estacion[i].EstadoActual[1]=0;
        Estacion[i].EstadoActual[2]=0;
        Estacion[i].EstadoActual[3]=0;
        DibujarPallet(Pos[i],Pos[i]-1);
        est[i]=true;
      }
      break;
    }
  }
//Por el contrario, si la estación está activa, compruebo cómo están sus señales asociadas
//y en función de estas y de la máquina asociada a las mismas, actúo de una manera u otra.

 if(Inicio||(Poner&&Estacion[i].EstadoAnterior[5]&&!Estacion[i].EstadoAnterior[3])){
    //si el almacén está asociado a la estación y si:
    //        a._ debemos poner el primer palé en la cinta,
    //        b._ o el almacén ha de poner otro palé,
    // llamamos al primer bloque de la función Almacén.
    if(Almacen==i)HAlmacen(1,i);
}
  //si el almacén acaba de poner una pieza en la cinta
  if(Estacion[i].EstadoAnterior[5]&&Estacion[i].EstadoAnterior[3]&&AlmacenPoniendo)
    if(Almacen==i)HAlmacen(2,i);
  //si acaba de llegar un palé a la estación asociada al almacén
  if(Estacion[i].EstadoAnterior[5]&&Estacion[i].EstadoAnterior[4]&&!Estacion[i].EstadoAnterior[3])
    if(Almacen==i)HAlmacen(3,i);
```

```
 //si acabo de poner el palé en el almacén
 if(Estacion[i].EstadoAnterior[5]&&Estacion[i].EstadoAnterior[3]&&AlmacenQuitando)
   if(Almacen==i)HAlmacen(4,i);
 // si acaba de llegar un palé a alguna de las estaciones asociadas
 // a los centros de mecanizado e inspección.
 if(Estacion[i].EstadoAnterior[6]&&Estacion[i].EstadoAnterior[4]&&
   !Estacion[i].EstadoAnterior[3]){
   if(Torno==i)if(!Torneando)HTorno(1,i);
   if(Fresadora==i)if(!Fresando)HFresadora(1,i);
   if(Dea==i)HDea(1,i);
 }
 //si  hemos finalizado la mecanización o inspección.
 if(Estacion[i].EstadoAnterior[5]&&Estacion[i].EstadoAnterior[3]){
   if(Torno==i)HTorno(2,i);
   if(Fresadora==i)HFresadora(2,i);
   if(Dea==i)HDea(2,i);
 }
}
```

Las funciones *HAlmacen, HTorno, HFresadora* y *HDea* son las funciones en cargadas de actuar de acuerdo al estado en el que se encuentren las señales asociadas a cada uno de los relés. Nos van a decir, por ejemplo, si debo mecanizar una pieza en el torno o si debo dejar pasar el palé, si debo recoger el palé y depositarlo en el almacén o dejarlo pasar para ser inspeccionado en el centro de calidad,... Sirva como ejemplo de este comportamiento una parte de la función *HAlmacen* que se muestra a continuación.

```
void TfrmEjecucion::HAlmacen(int caso,int estacion)
{
switch(caso){
 case 1://poner pieza del almacén a la cinta
   Inicio=false;
   Estacion[estacion].EstadoActual[1]=1;//levanta límites de entrevía
   Estacion[estacion].EstadoActual[2]=1;//lenvanta perno de restricción
   Estacion[estacion].EstadoActual[3]=1;//Habilita posicionadores
   Cinta[0]=numeropieza;
   P0->Visible=true;
   //ponemos la pieza correcta en la cinta transportadora
```

```
    PonerPiezaCinta(hcomm[1],SeleccionaCelda(true));
    AlmacenPoniendo=true;
    Poner=false;
    break;
  case 2://hemos puesto el palé en la cinta
    Estacion[estacion].EstadoActual[1]=0;//bajo los límites de entrevía
    Estacion[estacion].EstadoActual[3]=0;//deshabilito los posicionadores
    PiezasPuestas++;
    AlmacenPoniendo=false;
    DibujarPallet(1,0);
    // si ya hay 3 palés en la cinta o no se debe poner ninguno más doy la orden de no poner
    if((PiezasPuestas<3)&&(PiezasPuestas<NumTotalPiezas)){
      Poner=true;
      espera=true;
    }else Estacion[estacion].EstadoActual[2]=0;//bajo el perno de restriccion
    break;
.....
}
```

CAPÍTULO SÉPTIMO.

ESTUDIO ECONÓMICO.

7.1. INTRODUCCIÓN.

La finalidad de este proyecto ha sido la creación y desarrollo de una aplicación informática capaz de integrar y controlar eficientemente las diferentes máquinas que constituyen la FMC del LOIP de la Escuela Técnica Superior de Ingeniería Industrial de Valladolid.

Dicha célula fue adquirida antes de realizar este proyecto, por lo que en el estudio económico no incluiremos el coste del equipo físico que se debe integrar y controlar, es decir, no tendremos en cuenta el coste de la FMC.

Por otro lado, la gestión de un proyecto de software presenta unas características diferenciadoras que es necesario tener en cuenta. En este sentido, a diferencia de otros proyectos industriales de tipo mecánico, eléctrico o electrónico, este proyecto no implica un aporte sustancial de material sino que se encamina a la concepción, realización y ejecución de un producto informático que realice el control y gobierno de las máquinas que forman la célula.

Durante el desarrollo de este proyecto se ha puesto especial interés en cuestiones relativas a la ingeniería de proyectos, con las características especiales que presenta el desarrollo de un sistema de *software.* Algunos de estos aspectos se pueden observar a la hora de realizar la gestión del proyecto y el presupuesto del mismo.

7.2. JERARQUÍA EN LA GESTIÓN DE UN PROYECTO DE SOFTWARE.

Las personas que intervienen en un proyecto de este tipo pueden ser clasificadas de acuerdo con alguno de estos cometidos.

- **Organizador.**
- **Analista funcional.**
- **Analista orgánico.**
- **Programador.**

Estas personas establecen unas relaciones entre ellas de acuerdo a una determinada jerarquía de gestión de software, tal y como se puede apreciar en la figura 7.1.

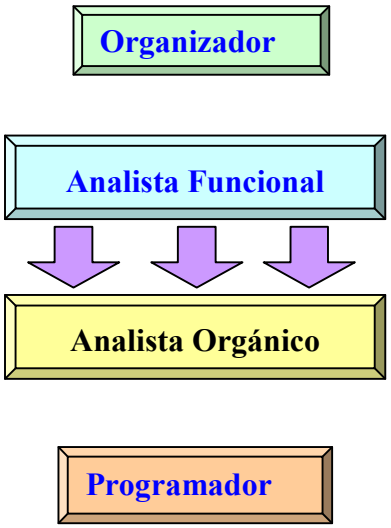


Figura 7.1. *Jerarquía organizativa del proyecto.*

El **organizador**, o **director de proyecto**, es el responsable de la idea del proyecto y el encargado de llevar a cabo los contactos iniciales con los clientes. También realiza la planificación del proyecto, al igual que su presupuesto económico. Por otra parte, es el encargado de dirigir y coordinar a las diferentes personas que intervienen en su realización.

El **analista funcional** es el que define los detalles que deberá cumplir el proyecto de *software* a desarrollar. El contacto con los clientes, así como el buen conocimiento del campo en el que se inserta el contenido del proyecto, resulta fundamental para que el producto final cumpla las necesidades especificas de éstos y sea de aplicación en la empresa. El analista funcional, junto con el analista orgánico, realiza la elección de los medios que deberán utilizarse en el desarrollo del proyecto. Esta elección también incluye el lenguaje de programación.

Además, en aquellos proyectos en los que la elaboración del software no implica su finalización sino que éste es utilizado como una herramienta para el desarrollo del resto del proyecto, el analista funcional es el encargado de la parte restante, si bien puede ser ayudado en esta tarea por otros colaboradores.

El **analista orgánico** se encarga de la realización del pseudocódigo, es decir, analiza desde el punto de vista de la programación las características del proyecto a realizar.

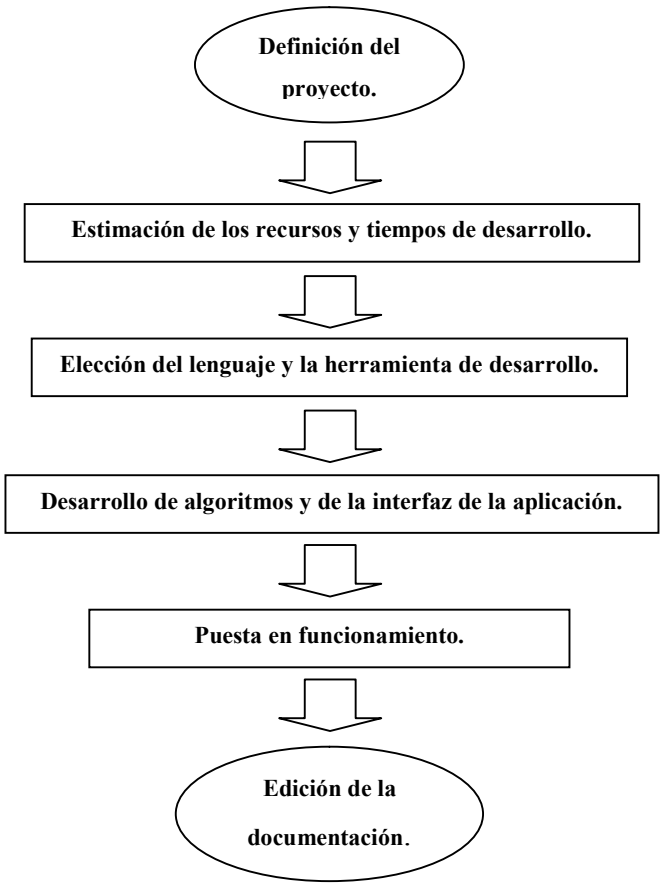
Por último, el **programador** es el encargado de traducir lo recibido por el analista orgánico, el pseudocódigo, al lenguaje de programación elegido (en nuestro caso C++).

7.3. FASES DE UN PROYECTO DE DESARROLLO DE SOFTWARE.

Las características de los proyectos de software hacen que su gestión presente una orientación diferente. Una primera particularidad es que **el software se desarrolla**, es decir, no se fabrica en un sentido clásico. Los costes del *software* vienen dados por las horas de ingeniería empleadas y no por la fabricación física del producto. Como muestra, considérese el bajo coste de los medios de almacenamiento magnético en los que se graban los programas, comparándolo con el coste global del sistema.

La segunda diferencia está en la vida útil del producto. **El software no se degrada**. En teoría, una vez que se han detectado y corregido los errores iniciales, su vida útil es ilimitada. Sin embargo, la realidad es que en el sistema se irán introduciendo correcciones y modificaciones, que hacen que los fallos tiendan a aumentar a lo largo del tiempo, debido al aumento de la complejidad y a la pérdida de homogeneidad.

Otra particularidad es el mantenimiento. Cuando una pieza falla se sustituye por el repuesto adecuado, y el sistema vuelve a tener un funcionamiento correcto. Pero en nuestro caso, los fallos suelen traer consecuencias más graves. Un fallo en el programa de software puede indicar un error de diseño. Por tanto, **el mantenimiento del software tiene una complejidad** considerablemente **mayor** que el mantenimiento de otros sistemas.



La determinación de las fases que conlleva el desarrollo de un proyecto de este tipo puede variar según el punto de vista de la persona que lo esté analizando. Sin embargo, estas etapas pueden ajustarse a la división mostrada en la figura 7.2.

Figura 7.2. *Fases en el Desarrollo del proyecto.*

La explicación de cada etapa se expone a continuación:

1. **Definición del proyecto.** En esta etapa se lleva a cabo un análisis general del problema. Se procede a la recopilación de la información necesaria referente al tema a tratar así como a la consulta de bibliografía. También se buscan en el mercado programas que estén relacionados. Basándonos en los datos obtenidos se formula el problema, se determina el alcance y las líneas generales del proyecto, se establece lo que hay que hacer (planificación de tareas) y quién tiene que hacerlo (asignación de recursos).

2. **Estimación de los recursos y tiempos de desarrollo.** Una vez abordado el problema en general, se procede al análisis detallado del mismo. Se planifica su desarrollo en cuanto a tiempos, así como en cuanto a recursos necesarios. Es en este momento cuando debe analizarse la viabilidad del proyecto; detectar la inviabilidad del proyecto en las etapas posteriores conduce a un aumento considerable de los costes.

3. **Elección del lenguaje y la herramienta de desarrollo visual.** Dada la gran variedad de lenguajes de programación y herramientas de desarrollo visual existentes hoy en día en el mercado, hay que estudiar de forma exhaustiva cuál de ellos es el más adecuado a nuestras necesidades. La elección de un lenguaje frente a otros vendrá determinada por aspectos tales como la complejidad, su potencia, la importancia en el manejo de bases de datos,...

4. **Desarrollo de algoritmos y de la interfaz de la aplicación.** Consiste en la determinación de un organigrama tanto general, como de cada una de las partes que puedan considerarse, especificando los datos de entrada y salida que tienen lugar en las mismas. Se elaboran las diferentes pantallas y menús que organizan la entrada y salida de datos.

5. **Puesta en funcionamiento.** Se realizan pruebas de aplicación para detectar posibles errores y poder proceder a su corrección.
6. **Edición de la documentación.** Una vez desarrollado el programa se redactará un estudio del mismo, así como un manual que facilite al usuario el empleo del programa desarrollado.

7.4. EL PROCESO DE GESTIÓN DEL PROYECTO.

Antes de comenzar a planificar un proyecto, deben establecerse el ámbito y los objetivos de éste, deben considerarse soluciones alternativas y deben identificarse las restricciones técnicas y de financiación.

Una de las actividades cruciales del proceso de gestión de proyectos de *software* es la planificación. Cuando se planifica un proyecto de software se tienen que obtener estimaciones del esfuerzo humano requerido, de la duración cronológica del proyecto y del coste. Generalmente las estimaciones se hacen valiéndose de la experiencia pasada, utilizando datos de otros proyectos análogos ya finalizados.

Es importante poder identificar los riesgos y su variabilidad antes de que aparezcan, para combatirlos enérgicamente, pues en caso contrario ellos nos combatirán a nosotros. El análisis de riesgos es algo vital para una buena gestión de proyectos de *software.*

La planificación temporal de un proyecto de *software* no difiere de la planificación de cualquier proyecto de ingeniería. Se identifican un conjunto de tareas del proyecto, se establecen interdependencias entre las tareas, se estima el esfuerzo asociado con cada tarea, se hace la asignación del personal y de otros recursos, se crea una red de tareas y una agenda de fechas.

Es fundamental realizar un seguimiento y control del proceso. Se deben detectar los retrasos con anticipación y reasignar los recursos, reordenar las tareas o modificar los compromisos de entrega, para resolver el problema detectado.

7.5. ESTUDIO ECONÓMICO DEL PROYECTO.

En este apartado se va a desarrollar el estudio económico propiamente dicho, relacionándolo con las diferentes etapas de desarrollo del proyecto. Se sigue una **contabilidad por actividades**, en la que se valoran los costes de cada una de las actividades que se realizan hasta la obtención del producto final. De esta forma, es posible analizar la influencia total en el coste del producto de cada uno de los procesos que intervienen con relación al coste total del producto.

Para llevar a cabo el estudio, se efectúan los siguientes cálculos:

1. Cálculo de las horas efectivas anuales, para determinar las tasas por hora de los salarios.
2. Cálculo de las amortizaciones del equipo.
3. Coste por hora y por persona de los materiales calificados como consumibles.
4. Coste por hora y persona de los costes indirectos.
5. Horas de personal dedicadas a cada una de las etapas.

7.5.1. Horas efectivas anuales y tasas horarias de personal.

Debemos obtener las tasas por hora y por semana de los salarios y de las amortizaciones de material para poder efectuar las valoraciones presupuestarias, tanto en la fase de planificación como en la de desarrollo. Las tasas por semana se utilizan en la contabilización de los costes del tiempo de desarrollo estimado; resultaría absurdo realizar un planteamiento para obtener el resultado en horas de trabajo.

En primer lugar, se calculan los días efectivos de trabajo anual basándonos en datos históricos. Todos estos valores se recogen en las tablas 7.1 (días efectivos) y 7.2 (semanas efectivas).

Año medio: (365.25 días)	365,25
Sábados y domingos: (365 × 2/7)	-104,36
Días efectivos de vacaciones:	-20,00
Días festivos reconocidos:	-12,00
Media de días perdidos por enfermedad:	-15,00
Cursillos de formación, etc.:	4,00
Total estimado días efectivos:	**210**
TOTAL HORAS/AÑO EFECTIVAS (8 hora/día):	**1.680**

Tabla 7.1. *Días efectivos.*

Año medio (semanas):	52
Vacaciones y festivos:	-5
Enfermedad:	-2
Cursos de formación:	-1
TOTAL SEMANAS:	**44**

Tabla 7.2. *Semanas efectivas.*

Para el desarrollo del proyecto se contará con:

- Un **ingeniero de sistemas**, que actúa como director del proyecto.
- Un **ingeniero industrial**, que es el encargado de llevar a cabo la gestión de los diferentes elementos del programa de software.
- Un **informático**, encargado de la programación.
- Un **auxiliar administrativo**, encargado de generar los informes correspondientes y los documentos de utilización del sistema.

El coste horario y semanal de cada uno de estos profesionales es el que se muestra en la tabla 7.3.

Concepto	Director	Ingeniero	Informático	Auxiliar Administrativo
Sueldo (neto +incentivos)	54.091,09 ∈ 9.000.000 Pts	23.794,07 ∈ 3.959.000 Pts	17.789,96 ∈ 2.960.000 Pts	11.419,23 ∈ 1.900.000 Pts
Seguridad Social (35%)	18.931,88 ∈ 3.150.000 Pts	8.329,76 ∈ 1.385.956 Pts	6.229,54 ∈ 1.036.509 Pts	3.996,73 ∈ 665.000 Pts
Total:	**73.022,97 ∈ 12.150.000 Pts**	**32.129,09 ∈ 5.345.831 Pts**	**23.839,15 ∈ 3.966.500 Pts**	**15.415,96 ∈ 2.565.000 Pts**
COSTE HORARIO:	**43,47 ∈ 7.232 Pts**	**19,12 ∈ 3.182 Pts**	**14,30 ∈ 2.380 Pts**	**9,18 ∈ 1.527 Pts**
COSTE SEMANAL:	**1.869,04 ∈ 310.982 Pts**	**822,34 ∈ 136.826 Pts**	**545,83 ∈ 90.818 Pts**	**394,63 ∈ 65.661 Pts**

Tabla 7.3. *Coste del equipo de profesionales (en euros y pesetas).*

7.5.2. Cálculo de las amortizaciones del equipo.

Para el equipo informático instalado se considera un período de amortización de 3 años, con cuota lineal.

El equipo se puede separar en dos grupos diferentes: el equipo destinado a realizar las tareas de control y elaboración del software propiamente dicho, y que se denomina **sistema de control** o **sistema de desarrollo**, cuyos costes (indicados en euros) vemos en la tabla 7.4. y, por otra parte, el **sistema de gestión de documentos** o **sistema de edición** (tabla 7.5.), empleado en el desarrollo de todos los documentos e informes relacionados con el proyecto.

Concepto		Coste	Cantidad	Coste total
Pentium III 600 Mhz, 20 Gb HD, 128 Mb RAM		1.292,18	2	2.584,35
Tarjeta de Comunicaciones NI RS232		721,21	1	721,21
Software de desarrollo				
Microsoft Windows 98 [9]		90,26	1	90,26
Borland C++ Builder Enterprise 4.0.		116,42	1	116,42
	Total a amortizar:			**3.512,24**
	Tipo		**Número**	**Amortización**
	Diaria (365 días)		9,62	3,21
	Semanal (52 semanas)		67,54	22,51
	Horaria (40 horas/semana)		1,69	0,56

Tabla 7.4. *Coste del equipo de desarrollo (en euros).*

Concepto		Coste	Cantidad	Coste total
Pentium III 500 Mhz, 10 Gb HD, 64 Mb RAM		1.081,82	1	1.081,82
Impresora HP DeskJet 895C		240,40	1	240,40
Software de edición				
Microsoft Windows 98		17,92	1	17,92
Office 97		420,11	1	420,11
	Total a amortizar:			**1.760,25**
	Tipo		**Número**	**Amortización**
	Diaria (365 días)		4,82	1,61
	Semanal (52 semanas)		33,85	11,28
	Horaria (40 horas/semana)		0,85	0,28

Tabla 7.5. *Costes del equipo de edición (en euros).*

[9] Microsoft Windows 98 tiene un valor de 108,18 ∈uros (18.000 Pts). Sin embargo, este sistema operativo forma parte tanto del software de edición como el de desarrollo. Por este motivo hemos descompuesto su coste en dos partes proporcionales al número de horas personales dedicadas a cada proceso.

7.5.3. Costes del material consumible.

Tanto el equipo de desarrollo como el equipo de edición necesitarán medios materiales para la realización de su trabajo: imprimir listados e informes, almacenar programas y documentos, etc. Estos medios se denominan consumibles e incluyen: papel de impresora, disquetes, documentación, etc. Se ha desarrollado su consumo medio, por persona y hora de trabajo, dando los siguientes resultados:

Concepto	Coste
Papeles de impresora	90,15
Suministros para impresora	270,46
Disquetes y CDs	120,20
Otros	360,61
Coste total anual por persona:	**841,42**
Coste horario por persona:	**0,50**

Tabla 7.6. *Costes del material consumible (en euros).*

7.5.4. Costes indirectos.

Aquí se considerarán los gastos propios del consumo de electricidad, teléfono, calefacción, alquileres, etc. Las tasas de coste calculadas por persona y hora para cada uno de estos conceptos se muestran en la tabla 7.7.

Concepto	Coste
Teléfono	120,20
Alquileres	390,66
Electricidad	120,20
Otros	240,40
Coste total anual por persona:	**871,46**
Coste horario por persona:	**0,52**

Tabla 7.7. *Costes del material consumible (en euros).*

7.5.5. Horas de personal dedicadas a cada fase.

Mediante la realización de un estudio de tiempos y la revisión de otros proyectos con características similares al presente, se determinó que la dedicación del personal del personal en cada una de las etapas fue como se reseña a continuación en la tabla 7.8.

Personal	Etapas					
	1	**2**	**3**	**4**	**5**	**6**
Director de proyecto	75	15	5			
Ingeniero		60	15	150	10	30
Informático			10	300	75	55
Auxiliar Administrativo	5	10				120
Total	**80**	**85**	**30**	**450**	**85**	**205**

Tabla 7.8. *Horas dedicadas por persona al proyecto.*

7.6. COSTES ASIGNADOS A CADA FASE DEL PROYECTO.

Para asignar los costes calculados para los recursos a cada fase del proyecto, se tendrán en cuenta las horas que cada persona dedica a cada etapa y las tasas horarias de salarios y amortización, así como los costes estimados para el material consumible y los costes indirectos.

7.6.1. Fase 1: Definición del proyecto.

En esta etapa intervienen tanto el director de proyecto como el auxiliar administrativo. El director del proyecto, a través de entrevistas personales, concreta con los responsables de la empresa cuáles son los objetivos que se desean alcanzar. El auxiliar administrativo se encarga de las tareas de redacción de documentos y mecanografía requeridas en esta etapa.

Este es el momento en el que se propone la realización de un programa de software para la FMC. Dada la especificidad de la célula, este programa tendrá que ser de elaboración propia.

Como se puede apreciar en la tabla 7.8., ésta es la fase en la que más interviene el director de proyecto: recopila información en forma de bibliografía y artículos publicados referentes al proyecto y, además, define las líneas de actuación y orienta a los otros integrantes del equipo.

- El ingeniero desarrollará el modelo del programa de la célula que posteriormente el informático traducirá a un lenguaje de programación. Así mismo, en caso de tener que instalar posteriormente el programa en una determinada empresa, sería necesario estudiar las características de ésta analizando el entorno sectorial de la misma y el proceso productivo que se lleva a cabo.

- El informático se encargará de desarrollar la aplicación sobre la base del modelo ideado por el ingeniero. En caso de realizarse estudios de viabilidad de la aplicación del modelo en una empresa determinada, se encargará de habilitar los mecanismos de captura de datos necesarios para alimentar al modelo.

- El auxiliar administrativo será el responsable de todas las tareas de redacción de documentos, manuales y labores de mecanografía allá donde las requieran el resto de integrantes del equipo.

Este proceso llevó en total un plazo de 75 horas al director de proyecto. La elaboración de documentos, por parte del al auxiliar administrativo, llevó un total de 5 horas. Por lo tanto, esta primera fase supuso un total de 80 horas.

Los costes de esta fase se reparten según se indica en la tabla 7.9.

Concepto		**Horas**	**C.H.**[10]	**Coste total (∈/Ptas)**
Personal				
	Director de proyecto	75	43,47	**3.260,25 ∈ / 542.460 Ptas**
	Ingeniero			
	Informático			
	Auxiliar administrativo	5	9,18	**45,9 ∈ / 7.367 Ptas**
Amortización				
	Equipo de desarrollo			
	Equipo de edición	5	0,28	**1,40 ∈ / 233 Ptas**
Material Consumible				
	Varios	80	0,50	**40 ∈ / 6.655 Ptas**
Costes indirectos		80	0,52	**41,60 ∈ / 6.922 Ptas**
			COSTE TOTAL:	**3.389,15 ∈ / 563.907 Ptas**

Tabla 7.9. *Costes asociados a la fase 1.*

7.6.2. Fase 2: Estimación de los recursos y tiempos de desarrollo.

En esta etapa intervienen tanto el director de proyecto como el ingeniero y el auxiliar administrativo, empleando un total de 85 horas (véase tabla 7.8.).

La **estimación de los recursos** requeridos para hacer frente al esfuerzo de desarrollo del sistema es una labor complicada pero muy importante. Con esta estimación se puede predecir el esfuerzo de tiempo, presupuesto y personal. Como asunto destacado en la estimación de los recursos, cabe reseñar la necesidad de comprar dos ordenadores de desarrollo. Esta exigencia se debe a que durante algunas etapas del proceso dos personas van a tener que trabajar simultáneamente con dicho equipo.

[10] C.H. Coste Horario expresado en euros.

Otro factor esencial para poder estimar el presupuesto final del proyecto, y su resultado en pérdidas y ganancias es la **estimación del tiempo de desarrollo** de cada tarea.

Como consecuencia de este análisis, se puede valorar el riesgo de continuar con el proyecto y las pérdidas en las que se puede incurrir. Ante las perspectivas favorables ofrecidas por los estudios de viabilidad, el director del proyecto da luz verde al desarrollo del mismo.

Los costes asignados en esta fase se muestran en la tabla 7.10.

Concepto		**Horas**	**C.H.**	**Coste total (∈/Ptas)**
Personal				
	Director de proyecto	15	43,47	**652,05 ∈ / 108.492 Ptas**
	Ingeniero	60	19,12	**1.147,20 ∈ / 190.878 Ptas**
	Informático			
	Auxiliar administrativo	10	9,18	**91,80 ∈ / 15.274 Ptas**
Amortización				
	Equipo de desarrollo	60	0,56	**33,60 ∈ / 5.591 Ptas**
	Equipo de edición	10	0,28	**2,80 ∈ / 466 Ptas**
Material Consumible				
	Varios	85	0,50	**42,5 ∈ / 7.071 Ptas**
Costes indirectos		85	0,52	**44,2 ∈ / 7.354 Ptas**
			COSTE TOTAL:	**2.014,15 ∈ / 335.126 Ptas**

Tabla 7.10. *Costes asociados a la fase 2.*

7.6.3. Fase 3: Elección del lenguaje y la herramienta RAD.

Tal y como se comentó en el capítulo anterior, en este proyecto se ha optado por la utilización de Borland C++ Builder versión 4.0 como herramienta de desarrollo rápido y visual. Esta herramienta utiliza como lenguaje de programación C++. Las razones de su empleo se analizan a continuación.

C++ es actualmente el lenguaje orientado a objetos más usado, tanto en Windows como en otros sistemas operativos y plataformas, siendo flexible, potente, transportable y conocido por muchos programadores.

El entorno de desarrollo de C++ Builder es simple, flexible y potente, contando con un gran número de componentes prefabricados que facilitan de forma notable la creación de cualquier aplicación.

A diferencia de otros entornos RAD, C++ Builder incorpora un compilador de C++, lo que quiere decir que el código objeto generado es directamente ejecutable, sin necesidad de distribuir archivos adicionales para la interpretación de pseudocódigo. Esto tiene básicamente dos consecuencias: el ejecutable es más rápido, al no tener que ser interpretado en tiempo de ejecución, y el tamaño final de la aplicación suele ser inferior

Los costes correspondientes a esta fase del proyecto se muestran en la tabla 7.11.

Concepto		**Horas**	**C.H.**	**Coste total (∈/Ptas)**
Personal				
	Director de proyecto	5	43,47	**217,35 ∈ / 36.164 Ptas**
	Ingeniero	15	19,12	**286,8 ∈ / 47.720 Ptas**
	Informático	10	14,3	**143 ∈ / 23.793 Ptas**
	Auxiliar administrativo			
Amortización				
	Equipo de desarrollo	25	0,56	**14 ∈ / 2.329 Ptas**
	Equipo de edición			
Material Consumible				
	Varios	30	0,50	**15 ∈ / 2.496 Ptas**
Costes indirectos		30	0,52	**15,60 ∈ / 2.596 Ptas**
			COSTE TOTAL:	**691,75 ∈ / 115.098 Ptas**

Tabla 7.11. *Costes asociados a la fase 3.*

7.6.4. Fase 4: Desarrollo de algoritmos y de la interfaz de usuario.

En esta etapa han intervenido el ingeniero y el informático, empleando un total de 450 horas, repartidas tal y como se indicó en la tabla 7.8. El ingeniero es el encargado de, entre otros aspectos:

- Determinar el organigrama, tanto general como de cada una de las partes que puedan considerarse.
- Desarrollar las necesidades de comunicación con las máquinas.
- Especificar los datos de entrada y salida que tienen lugar en las mismas.
- Esbozar el diseño de las bases de datos necesarias.
- Diseñar las distintas pantallas y menús que se puedan necesitar.

Toda esta información es la que recibe el programador informático para que pueda llevar a cabo el adecuado desarrollo de la aplicación. Los costes asignados a esta fase aparecen en la tabla 7.12.

Concepto		**Horas**	**C.H.**	**Coste total (∈/Ptas)**
Personal				
	Director de Proyecto			
	Ingeniero	150	19,12	**2.868 ∈ / 477.195 Ptas**
	Informático	300	14,30	**4.290 ∈ / 713.796 Ptas**
	Auxiliar Administrativo			
Amortización				
	Equipo de desarrollo	450	0,56	**252 ∈ / 41.929 Ptas**
	Equipo de edición			
Material Consumible				
	Varios	450	0,50	**225 ∈ / 37.437 Ptas**
Costes indirectos		450	0,52	**234 ∈ / 38.934 Ptas**
			COSTE TOTAL:	**7.869 ∈ / 1.309.291 Ptas**

Tabla 7.12. *Costes asociados a la fase 4.*

7.6.5. Fase 5: Puesta en funcionamiento.

Tras desarrollar los algoritmos y la interfaz del programa, es el momento de ponerlo en marcha. En esta etapa intervendrán tanto el ingeniero (10 horas) como el informático (75 horas).

Con la puesta en funcionamiento pretendemos depurar la aplicación con el fin de evitar todos aquellos errores que provoquen el mal funcionamiento del programa. Se pondrá especial atención a aquellos relacionados con la transferencia de datos entre el ordenador central y las máquinas, y los relativos al tratamiento de datos.

También es conveniente consultar al personal del taller para conseguir un entorno más cómodo y amigable durante su utilización.

Los costes asignados a esta fase se muestran a continuación, en la tabla 7.13.

Concepto		**Horas**	**C.H.**	**Coste total (∈/Ptas)**
Personal				
	Director de Proyecto			
	Ingeniero	10	3.182	**191,24 ∈ / 31.820 Pts**
	Informático	75	2.380	**1.072,81 ∈ / 178.500 Pts**
	Auxiliar Administrativo			
Amortización				
	Equipo de desarrollo	85	107,6	**54,97 ∈ / 9.146 Pts**
	Equipo de edición			
Material Consumible				
	Varios	85	83,3	**42,56 ∈ / 7.081 Pts**
Costes indirectos		85	86,3	**44,09 ∈ / 7.336 Pts**
			COSTE TOTAL:	**1.405,67 ∈ / 233.883 Pts**

Tabla 7.13. *Costes asociados a la fase 5.*

7.6.6. Fase 6: Edición de la documentación.

En esta etapa han intervenido tanto el ingeniero como el informático y el auxiliar administrativo empleando un total de 205 horas de trabajo.

En esta fase se elabora toda la documentación referente a las mejoras realizadas en el sistema, así como un manual para explicar las variaciones en el funcionamiento que haya podido sufrir el proceso debido a las modificaciones que se hayan introducido.

Los costes asignados en esta fase son los que se detallan a continuación (tabla 7.14.).

Concepto		**Horas**	**C.H.**	**Coste total (∈/Ptas)**
Personal				
	Director de proyecto			
	Ingeniero	30	19,12	**573,6 ∈ / 95.439 Ptas**
	Informático	55	14,30	**786,5 ∈ / 130.863 Ptas**
	Auxiliar administrativo	120	9,18	**1.101,60 ∈ / 183.291 Ptas**
Amortización				
	Equipo de desarrollo	85	0,56	**47,60 ∈ / 7.920 Ptas**
	Equipo de edición	120	0,28	**33,6 ∈ / 5.591 Ptas**
Material Consumible				
	Varios	205	0,50	**102,50 ∈ / 17.055 Ptas**
Costes indirectos		205	0,52	**106,60 ∈ / 17.737 Ptas**
			COSTE TOTAL:	**2.752 ∈ / 457.894 Ptas**

Tabla 7.14. *Costes asociados a la fase 6.*

7.7. CÁLCULO DEL COSTE TOTAL.

El coste total se obtiene como suma de los costes totales de cada una de las seis fases del proyecto, que se detallaron en el anterior apartado. Los costes totales desglosados para cada una de las fases se muestran en la tabla 7.15.

Actividad	Horas	Euros/Pesetas
Definición del proyecto.	80	**3.389,15 ∈ / 563.907 Ptas**
Estimación de los recursos y tiempos de desarrollo.	85	**2.014,15 ∈ / 335.126 Ptas**
Elección del lenguaje.	30	**691,75 ∈ / 115.098 Ptas**
Desarrollo de algoritmos y la interfaz de usuario.	450	**7.869 ∈ / 1.309.291 Ptas**
Puesta en funcionamiento.	85	**1.348 ∈ / 232.608 Ptas**
Edición de la documentación.	205	**2.752 ∈ / 457.894 Ptas**
TOTAL:	**935**	**18.064,05 ∈ / 3.005.605 Ptas**

Tabla 7.15. *Costes totales de cada fase.*

En el siguiente diagrama se muestra cómo quedan distribuidos los costes de cada una de las fases del proyecto respecto al total (figura 7.3.).

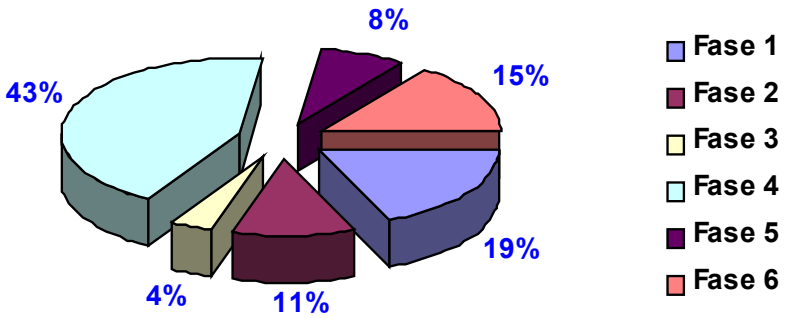


Figura 7.3. *Representación de los costes de cada fase sobre el total.*

Estimación del precio de venta unitario.

Para estimar el posible precio de venta de nuestro software se han de tener en cuenta las siguientes consideraciones:

- El software FMC se distribuirá en un CD-ROM junto con su manual de usuario.
- Se estima una venta de 15 copias para esta versión del software con un coste de edición, incluyendo el manual y el CD-ROM, de 40 Euros por unidad.
- El margen de beneficio se fija en el 20%.

En base al coste total del proyecto y a las anteriores consideraciones, se estima un precio de venta unitario de:

Concepto	**Coste**	**Cantidad**	**Total (∈/Ptas)**
Proyecto	18.064	1	**18.064 ∈ / 3.005.605 Pts**
Edición	40	15	**600 ∈ / 99.832 Ptas**
		TOTAL:	**18.664 ∈ / 3.105.436 Pts**

Tabla 7.17. *Coste total del proyecto.*

P.V.P.= (18.664 × 1,2) ÷ 15 = 1.493,12 Euros = 248.434 Pts.

Se ha estimado una demanda tan baja principalmente por el hecho de que una FMC requiere cambios importantes en la organización de la planta productiva. El CIM y la FMS implican una reforma total del modo de producir dentro de la empresa, y esta transformación no es asumida por muchos empresarios, que generalmente no son partidarios de realizar cambios en su forma de producir. Además, no todas las células están formadas por las mismas máquinas que integran la FMC del LOIP, lo que supone una limitación aún mayor.

CONCLUSIONES.

Para finalizar el presente proyecto de fin de carrera, vamos a enumerar las principales conclusiones que hemos podido extraer.

En primer lugar, se ha conseguido crear un programa de software capaz de integrar y controlar, tanto de forma individual como colectiva, todas las máquinas que componen la Célula de Fabricación Flexible del Laboratorio de Organización Industrial y Producción de la Escuela Técnica Superior de Ingenieros Industriales de Valladolid.

Se ha mejorado notablemente el entorno de trabajo del usuario encargado del ordenador central. El entorno de ventanas y botones de Windows 95 permite un uso más intuitivo del programa FMC. Además, se ha intentado que todas las operaciones necesarias se puedan realizar sin tener que soltar la mano del ratón, lo que nos permite ganar en comodidad y rapidez. En esta búsqueda de sencillez y comodidad, el programa FMC proporciona una amplia y variada ayuda sobre los temas más cruciales de la célula.

También se ha conseguido optimizar el almacenamiento de la información utilizando bases de datos. En concreto, los datos han sido almacenados en tablas del tipo Paradox. Para acceder a estas tablas se puede emplear el gestor local de bases de datos de C++ Builder, DataBase Desktop Manager (DBD). El Programa FMC incorpora una opción que nos permite acceder a este gestor DBD de forma rápida. De todas formas, aunque existe esta posibilidad de modificar manualmente los datos de las tablas, es una práctica desaconsejable (a no ser que se conozca perfectamente la estructura y el uso de la tabla que se quiere modificar), por lo que desde aquí recomendamos modificarlas mediante las diversas opciones que nos proporciona el programa FMC.

Además, este programa permite, tal y como ya se ha comentado, seleccionar el puerto de comunicaciones que se unirá a cada una de las máquinas de la célula. Esta circunstancia nos otorga una gran flexibilidad, dado que además de los 8 puertos que nos proporciona la tarjeta *National Instrument PCI-232/8* vamos a poder trabajar con los dos propios del ordenador personal, COM1 y COM2.

Otro gran logro del programa FMC, es que nos permite trabajar con distintas configuraciones, lo que significa que aunque se produjese un cambio en la ubicación física de las máquinas, el proceso seguiría ejecutándose correctamente.

En lo que se refiere al código de programa, se ha intentado que estuviese lo mejor estructurado posible, dividiendo las diferentes unidades en función de su uso principal: *unidades de control remoto y unidades de comunicación*. Se ha intentado comentar todos aquellos aspectos que sean importantes para comprender el funcionamiento tanto del programa, como de los elementos que componen la FMC. Además se ha procedido a realizar varias revisiones con el fin de minimizar los recursos que utiliza el programa.

Gracias al Editor MDI que incorpora este programa, podemos acceder de forma rápida a los diferentes archivos de texto que necesitan algunas de las máquinas que componen la FMC. Podemos modificar el programa y el archivo de puntos que enviamos al robot Mitsubishi, los programas de control numérico del torno y de la fresadora, el programa residente en la memoria NVRAM del autómata programable SLC 100,..., y muchos más. Gracias a esta aplicación vamos a conseguir un acceso rápido y cómodo a todos estos archivos.

FMC nos permite acceder al programa de Microsoft *Paint*, con el que poder crear, o modificar, de forma rápida imágenes de mapa de bits (archivos *.bmp). Así, podemos generar la imagen de un almacén, la de un robot, o la de un torno de una manera rápida y divertida. Además, *Paint* nos permite comprimir y expandir imágenes de forma fácil, lo que nos será muy útil cuando deseemos insertarlas en la distribución en planta de nuestra propia célula.

FMC también incorpora la ayuda necesaria para poder conocer el proceso de encendido y puesta en modo comunicación de las máquinas, el funcionamiento de los diferentes módulos de control remoto y el proceso a seguir para fabricar piezas en la FMC del LOIP.

Por último, aunque durante este proyecto no hemos podido incorporar a la célula el Centro de Calidad, el programa FMC acoge todas las opciones necesarias para su configuración y ejecución.

Por otra parte, no podemos olvidarnos de las posibles mejoras que pueden realizarse tanto en la FMC del LOIP como en el software que la controla. Algunas de esas **futuras mejoras** se comentan a continuación**.**

Actualmente existe un proyecto en curso para mejorar el software que controla el almacén. Sería interesante que dicho software introdujese mejoras significativas respecto al programa actual, tales como:

- Un mejor posicionamiento del brazo robotizado del almacén a la hora de coger y dejar piezas en el Almacén. En este sentido, el problema actual es el mal funcionamiento del algoritmo que calcula las posiciones de las celdas del almacén debido, en parte, a que las distancias entre celdas no son iguales.
- La posibilidad de poder modificar los puntos de recogida y espera del brazo robotizado.
- Con el software actual, las operaciones a las que tenemos acceso en modo comunicación son muy reducidas. En este sentido, el nuevo software debería permitir realizar alguna operación adicional: ejecutar el *Home* del brazo del almacén, poder transmitir los ficheros que contienen las coordenadas de las celdas, poder conocer la ubicación (en coordenadas cartesianas) del brazo robotizado,...
- Tal y como se comentó, el actual formato de transmisión inhabilita la celda número 13. Por lo tanto, el nuevo programa debe emplear un formato que permita el acceso a la totalidad de las celdas.

Además de todas estas mejoras en el software, se debe conseguir que el desplazamiento a orígenes (*Home*) se realice perfectamente. Este aspecto físico tiene una gran importancia dado que el resto de posiciones del almacén se define en función de este origen de coordenadas. Un fallo en su ubicación repercute negativamente en cualquier proceso asociado al almacén.

Sería interesante crear un programa que mejorase el proceso de producción mediante algoritmos de optimización. Este programa, partiendo de los datos introducidos por el usuario, tendría que ser capaz de secuenciarlos y descomponerlos de acuerdo a una serie de criterios de optimización.

Los datos iniciales que utilizará este programa de secuenciación serán los introducidos por el usuario en el módulo *Creación de Tareas*, los cuales se guardan en tablas de tipo Paradox ubicadas en el directorio "C:\Celula00_01\Tablas\TareasDiseñadas". El resultado del algoritmo de secuenciación será el orden de producción de las diferentes piezas de la tarea. Dicho resultado se guardará en tablas con la misma estructura que las tablas de partida, de tipo Paradox, pero en el directorio "C:\Celula00_01\Tablas\TareasSecuenciadas".

Por todo esto, el programa FMC contiene una opción denominada *Secuenciación*, en el menú *Herramientas* de la ventana principal, que está preparada para ejecutar un posible programa denominado "Secuenciacion.exe" ubicado en el directorio "C:\Celula00_01\Ejecutables".

También existe otra opción destinada a los programas de control numérico. Esta opción, denominada *Simulación CNC* y ubicada en el menú *Herramientas*, nos permitirá acceder a un programa capaz de simular la ejecución de un programa CNC. Dado que existen en el mercado numerosos simuladores que funcionan bajo entorno Windows, al presionar sobre dicha opción aparece un cuadro de diálogo que nos permite seleccionar archivos ejecutables, con el fin de que el usuario escoja el nombre del programa que más le interese.

El tamaño y peso de los equipos que constituyen la FMC, junto con la falta de espacio, hacen muy difícil cualquier modificación en su layout actual. Sin embargo, en vista a disminuir los tiempos de producción, sería necesario ubicar el almacén en la posición que actualmente ocupa el equipo de calidad y el equipo de calidad en la posición del almacén. De esta forma conseguiríamos optimizar la producción al impedir que las piezas que tuviesen que ser inspeccionadas diesen dos vueltas en la cinta. El programa FMC ya tiene en cuenta la posibilidad de un cambio en el layout de la célula, por lo que dicho cambio no supondría introducir ninguna alteración del código.

En definitiva, podemos ver que las ampliaciones que se pueden realizar son muy numerosas y con muchas posibilidades. El programa desarrollado en este proyecto, a pesar de ser bastante flexible y permitir la introducción de alguna de ellas sin tener que modificar parte del código, necesita nuevos cambios y ampliaciones, por lo que es muy probable que en el futuro se creen nuevas versiones.

REFERENCIAS BIBLIOGRÁFICAS.

BIBLIOGRAFÍA.

- [B1]: ANTÓN, L. A. *Desarrollo de un programa de software para controlar una Célula de Fabricación Flexible*. Proyecto Fin de Carrera. Departamento de Economía y Administración de Empresas de la ETSII de Valladolid. Noviembre, 1999.

- [B2]: ARNEDO, J. M. *Fabricación integrada por ordenador (CIM).* Colección Productica. vol 54. Marcombo S.A. 1992.

- [B3]: BAUMGARTNER, H.; KNISCHWSKI, K. y WIEDING, H. *CIM. Consideraciones básicas*. Marcombo S.A. 1991.

- [B4]: CHARTE, F. *Programación con C++ Builder 4.* Anaya Multimedia. 1999.

- [B5]: CHARTE, F. *Programación Avanzada con C++ Builder 4*. Anaya Multimedia. 1999.

- [B6]: FERRÉ, R. *Fabricación asistida por computador – CAM*. Colección Productica. Vol 4. Marcombo. 1987.

- [B7]: FERRÉ, R. ***La fábrica flexible***. Colección Productica. Vol 9. Marcombo. 1988.

- [B8]: FROMENT, B. y LESAGE, J.J. ***Fabricación flexible: productividad con series cortas. Centros de mecanizado flexible.*** Editorial TGP S.A. Madrid, 1988.

📖 [B9]: GUILLE, A. *Introducción a la Neumática*. Marcombo (Productica). 1990.

📖 [B10]: HERRERO, J. C. *Desarrollo de un paquete de software para el almacén de una Célula de Fabricación Flexible*. Proyecto Fin de Carrera. Departamento de Economía y Administración de Empresas de la ETSII de Valladolid. Abril, 1996.

📖 [B11]: LUGGEN, W. *Flexible manufacturing cells and systems.* Pretince Hall Inc. 1991.

📖 [B12]: MAYOL, A. *Autómatas programables.* Colección Productica. Vol 3. Marcombo, 1987.

📖 [B13]: MIRHO, C. A.; TERRISSE, A**.** ***Communications Programming for Windows 95***. Educación. Microsoft Press. 1996.

📖 [B14]: MOSHE, M**.** *Fundamentos de Robótica***.** Eshed Robotec. 1990.

📖 [B15]: SÁNCHEZ, J. L. *Desarrollo de un Paquete de Software para la Integración de una Célula de Fabricación Flexible*. Proyecto Fin de Carrera. Departamento de Economía y Administración de Empresas de la ETSII de Valladolid. Abril, 1995.

📖 [B16]: SCHILDT, H. *Turbo C/C++. Manual de Referencia*. Osborne McGraw-Hill. 1992.

📖 [B17]: WEBB, J. W**.** ***Programmable Logic Controllers. Principles and Applications***. Maxwell Macmillan. 1993.

MANUALES DE REFERENCIA.

- [M1]: *Computer integrate manufacturing laboratory manual*. Amatrol. 1992.

- [M2]: *Guía de referencia del lenguaje ALC (lenguaje de control avanzado)*. 3ª Edición. Eshed Robotec LTD. 1982.

- [M3]: *Guía de referencia Scorbase nivel 5 versión 1.2*. Eshed Robotec LTD. 1982.

- [M4]: *Industrial Micro-Robot System Model RV-M1 MoveMaster*. Mitsubishi Electric Corporation.

- [M5]: *Información sobre el controlador programable SLC™ 100***.** Fax enviado por Allen-Bradley en diciembre de 1994.

- [M6]: *Manual de usuario Scorbot-ER VII*. Eshed Robotec. 1990.

- [M7]: *Mint (Motion System Programming Guide). Users manual*. Arcom Control Systems LTD**.**

- [M8]: *Mirac instruction and maintenance manual*. Versión 2.06. Denford. 1992.

- [M9]: National Instrument. *Manual de la tarjeta para comunicaciones en serie de National Instrument*. Editorial NI. 1998.

- [M10]: *Triac VMC installation operation and maintenance manual*. Version 2.63. Denford. 1992.

REFERENCIAS WWW.

- [W1]: http://148.225.64.3/cim/robind/robind.html.
- [W2]: http://www.chi.ctesm.mx/~cim/robind/robotica.html.
- [W3]: http://www.favanet.com.ar/ftarg.com/revista/robot/robot.html.
- [W4]: http://www.eupmt.es/cra/robotica.htm.
- [W5]: http://members.aol.com/eboones/sabemos.htm.
- [W6]: http://chi.itesm.mx/~cim/peonYalfil/mill.htm.
- [W7]: http://chi.itesm.mx/~cim/peonYalfil/robotmit.htm.
- [W8]: http://www.denford.com/main.html.