TECNOLÓGICO
DE MONTERREY®

PROYECTOS DE INGENIERÍA MECATRÓNICA

DISEÑO DE UNA CELDA DE MANUFACTURA

Integrantes del equipo:
Alfredo Necoechea Hasfield [redacted]
O. Braulio Pozos López
Paola Sandoval Jiménez
Rodolfo A. Rivera Zacaula

Asesor: Israel Macías
Profesor: Jorge Eduardo Brieva R.

ÍNDICE

1 Introducción

1.1 Antecedentes

Una celda de manufactura es un equipo que simula una línea de producción automática la cual está compuesta de varias estaciones de trabajo (almacenamiento, transportación, maquinado, ensamble e inspección) que se comunican e integran entre sí.

Actualmente existen celdas de manufactura configuradas por Ethernet y que pueden ser comandadas desde una sola computadora. También existen diferentes paquetes de programación que están enfocados al manejo de robots, tal es el caso de Step7 o LabView.

1.1.1 Celdas de Manufactura

En la figura 1.1.1 a. se puede apreciar un ejemplo de una celda de manufactura. Esta celda se encuentra conectada por medio de Ethernet y cuenta con una computadora central que maneja a cada uno de los equipos. PHS (Personal Handyphone Systems) son sistemas de telefonía digital inalámbrica que dan una alta calidad y capacidad de servicios de voz y datos. El envío de datos de cada elemento a una red Local se hace a través de PHS, mientras que la red Local es controlada por medio de Ethernet. Las señales de PHS llegan a una computadora central en la que se controla a cada uno de los elementos de la red.

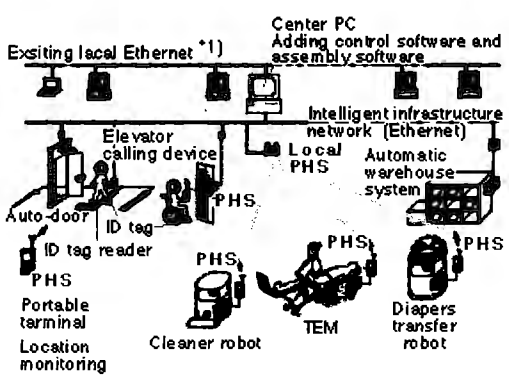


Figura 1.1.1 a. Ejemplo de Celda de Manufactura

A continuación se muestra otro ejemplo que funciona en fábricas actuales y que además de computadoras, PLC, robots y periféricos, implementa un paquete nuevo de monitoreo.

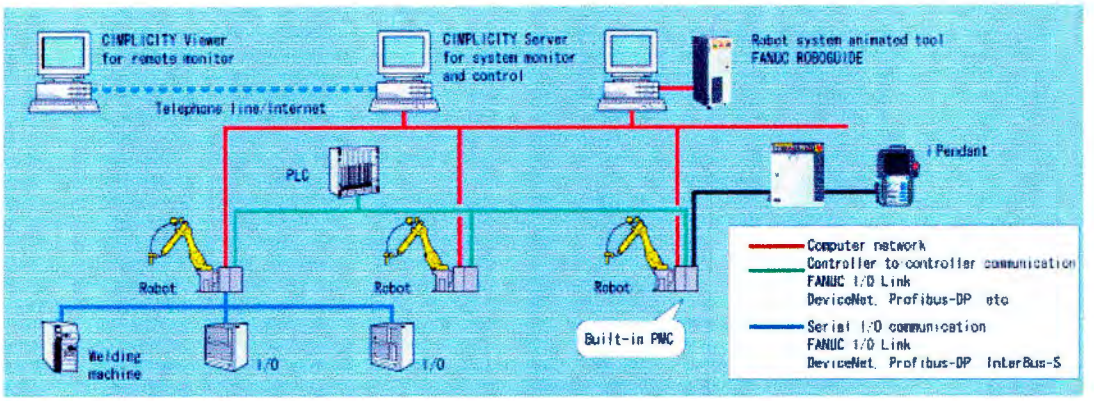


Figura 1.1.31.1.1 b. Tecnología para una Celda de Manufactura de la empresa FANUC

La funcionalidad para la comunicación en la Figura 1.1.1 b. es versátil en el controlador R-J3iB del robot y se hace posible el crear una red multi-robot con la computadora central, el PLC y periféricos para control y administración de la producción. Un procesador especializado en la comunicación asegura una comunicación de alta velocidad y estabilidad sin influencia negativa en el movimiento de los robots. La transferencia de archivos de información de entrada/salida entre el robot y el PLC así como la comunicación serial entrada/salida entre el robot y los periféricos brinda una solución de red industrial poderosa. A esto se le suma un controlador de alta velocidad y un PMC (controlador programable de máquinas) integrado al controlador del robot.

En esta celda se utiliza un paquete llamado CIMPLICITY, para el monitoreo y control administrativo en el sistema de producción. CIMPLICITY integrado a una computadora provee de un control sofisticado del sistema de robots incluso en un sitio remoto a través de una línea telefónica para satisfacer las necesidades en el piso de producción al monitorear y administrar al robot, PLC y periféricos conectados por medio de Ethernet.

Existe en este ejemplo otro dispositivo, el "iPendant", el cual nos permite realizar operaciones de enseñanza al robot. El "iPendant" puede ser conectado a la red de una fábrica y a Internet, permitiendo ver varios tipos de información en el campo. También permite crear y agregar pantallas propias al usuario.

1.1.2 Nuestra Celda de Manufactura

La antigua celda de manufactura del Instituto Tecnológico de Monterrey, Campus Ciudad de México, consta de una banda transportadora con sensores y actuadores neumáticos, un robot cartesiano (para almacenamiento), dos brazos robóticos, un sistema de visión y un PLC.

La banda transportadora es impulsada por un motor eléctrico, el cual permanece activo a lo largo del proceso (no se controla). La banda comienza y termina en el robot

cartesiano, por lo cual se puede dividir en dos secciones: una de ida y una de regreso. En el trayecto de la banda primero se encuentra un brazo robótico denominado "Puma". Para que éste pueda interactuar en el proceso, la banda cuenta con un sensor que detecta el arribo de un elemento al mismo. Asimismo, se cuenta con un actuador neumático con el fin de mantener el elemento al alcance del "Puma". Al final de la sección de ida se encuentran otro sensor y actuador neumático para cambiar al elemento de lado. Nuevamente encontramos al inicio de la banda de regreso un brazo robótico a su vez denominado "Mitsubishi", componentes como los utilizados para el "Puma" existen para el "Mitsubishi". Por último existen otro sensor y actuador neumático, además de un motor eléctrico, para introducir el elemento al sistema de diseño. Finalmente, el dispositivo cuenta con un sensor que sirve para detectar que el elemento ha llegado al final de la banda.

La comunicación entre los diferentes dispositivos de la celda se logra por medio del PLC, que a su vez es programado y monitoreado por una computadora. Todos los sensores, actuadores, y robots están conectados al PLC. Cada robot tiene una computadora que corre con el Sistema Operativo Windows 95, existen muchos problemas para arrancar las computadoras: no hay suficiente espacio en el disco duro, no hay recursos de memoria RAM suficientes y el sistema se bloquea con frecuencia, haciendo el uso de los robots complicado. Los programas que corren las computadoras de los robots se basan en el uso de comandos para programar los robots, lo cual complica la lectura y comprensión de los códigos de los robots.

La figura 1.1.2 resume el funcionamiento de la celda y muestra el equipo que se describió anteriormente.

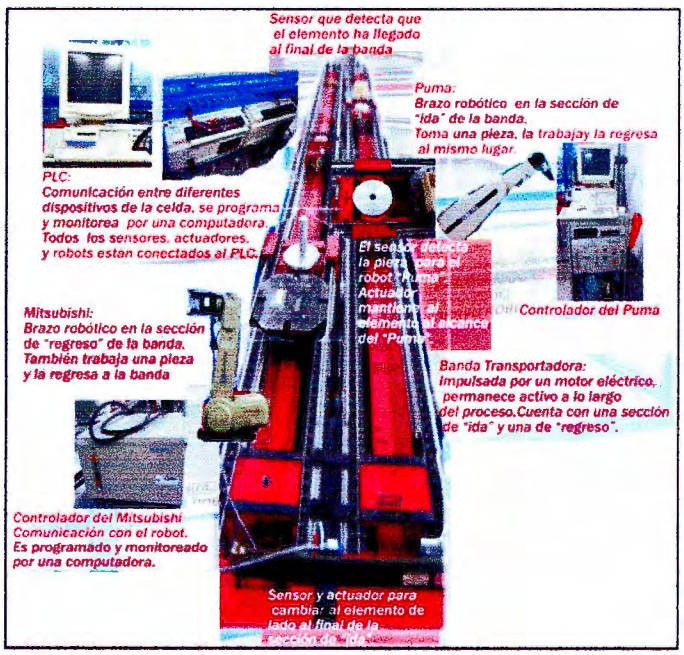


Figura 1.1.2. Componentes de la Celda de Manufactura

1.1.3 PLC

El Controlador Lógico Programable es un dispositivo usado para controlar. Este control se realiza sobre la base de una lógica, definida a través de un programa. El PLC está basado en un microprocesador con una memoria donde se pueden almacenar instrucciones para implementar funciones lógicas, secuenciales, de tiempo, de conteo y aritméticas para el control de máquinas y procesos.

La figura 1.1.3 a. muestra gráficamente las partes que pueden distinguirse en un PLC:

- ✓ *Interfaces de entradas y salidas*
- ✓ *CPU (Unidad Central de Proceso):* constituye el "cerebro" del sistema y toma decisiones en base a la aplicación programada
- ✓ *Memoria*
- ✓ *Dispositivos de Programación*

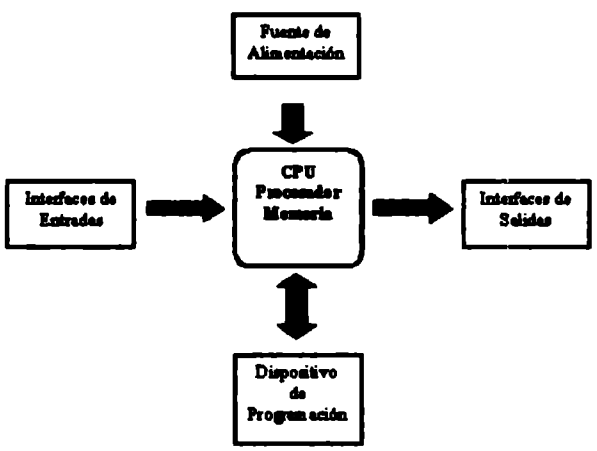


Figura 1.1.3 a. Estructura de un Controlador Lógico Programable

El PLC cuenta con varios componentes:

- Módulos para señales digitales y analógicas (I/O)
- Procesadores de comunicación (CP) para facilitar la comunicación entre el hombre y la máquina o entre máquinas. Se tiene procesadores de comunicación para conexión a redes y para conexión punto a punto.
- Módulos de función (FM) para operaciones de cálculo rápido.

Existen otros componentes que se adaptan a los requerimientos de los usuarios:

- Módulos de suministro de energía
- Módulos de interfaces para conexión de racks múltiples en configuración multi-hilera

En los módulos de entrada pueden ser conectados:

- Sensores inductivos, capacitivos, ópticos
- Interruptores

- Pulsadores
- Llaves
- Finales de carrera
- Detectores de proximidad

En los módulos de salida pueden ser conectados:
- Contactores
- Electroválvulas
- Variadores de velocidad
- Alarmas

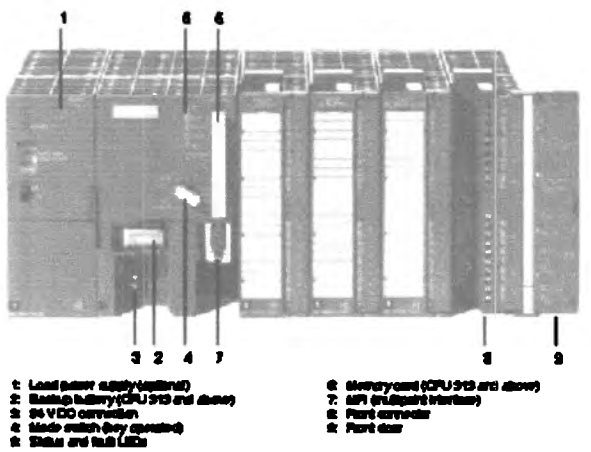


Figura 1.1.3 b. Algunos elementos del PLC

El usuario ingresa el programa a través del dispositivo adecuado (un cargador de programa o PC) y éste es almacenado en la memoria de la CPU.
La CPU procesa la información que recibe del exterior a través de la interfaz de entrada y de acuerdo con el programa, activa una salida a través de la correspondiente interfaz de salida.

Las interfaces de entrada y salida se encargan de adaptar las señales internas a niveles del la CPU. Por ejemplo, cuando la CPU ordena la activación de una salida, la interfaz adapta la señal y acciona un componente (transistor, relé, etc.)

Al comenzar el ciclo, la CPU lee el estado de las entradas. A continuación ejecuta la aplicación empleando el último estado leído. Una vez completado el programa, la CPU ejecuta tareas internas de diagnóstico y comunicación. Al final del ciclo se actualizan las salidas. El tiempo de ciclo depende del tamaño del programa, del número de E/S y de la cantidad de comunicación requerida.

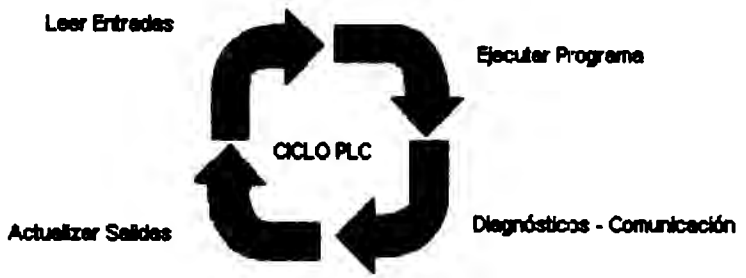


Figura 1.1.3 c. Ciclo del PLC

Las ventajas en el uso del PLC comparado con sistemas basados en relé o sistemas electromecánicos son:

- ***Flexibilidad:*** Posibilidad de reemplazar la lógica cableada de un tablero o de un circuito impreso de un sistema electrónico, mediante un programa que corre en un PLC.
- ***Tiempo:*** Ahorro de tiempo de trabajo en las conexiones a realizar, en la puesta en marcha y en el ajuste del sistema.
- ***Cambios:*** Facilidad para realizar cambios durante la operación del sistema.
- ***Confiabilidad***
- ***Espacio***
- ***Modularidad***
- ***Estandarización***

1.1.4 Ethernet

Ethernet se refiere a las redes de área local y dispositivos bajo el estándar IEEE 802.3 que define el protocolo CSMA/CD, aunque actualmente se llama Ethernet a todas las redes cableadas que usen un determinado formato de trama.

Esta tecnología fue estandarizada por la especificación IEEE 802.3, que define la forma en que los puestos de la red envían y reciben datos sobre un medio físico compartido que se comporta como un bus lógico, independientemente de su configuración física. Originalmente fue diseñada para enviar datos a 10 Mbps, aunque posteriormente ha sido perfeccionada para trabajar a 100 Mbps, 1 Gbps o 10 Gbps y se habla de versiones futuras de 40 Gbps y 100 Gbps.

En sus versiones de hasta 1 Gbps utiliza el protocolo de acceso al medio CSMA/CD (*Carrier Sense Multiple Access / Collision Detect* - Acceso múltiple con detección de portadora y detección de colisiones). Actualmente Ethernet es el estándar más utilizado en redes locales/LANs.

El diseño original funcionaba a 1 Mbps sobre cable coaxial grueso con conexiones vampiro (que "muerden" el cable) en 10Base5. Para la norma de 10 Mbps se añadieron las conexiones en coaxial fino (10Base2, también de 50 ohmios, pero más flexible), con tramos conectados entre sí mediante conectores BNC; par trenzado categoría 3 (10BaseT) con conectores RJ45, mediante el empleo de hubs y con una configuración física en estrella; e incluso una conexión de fibra óptica (10BaseF).

Los estándares sucesivos (100 Mbps o Fast Ethernet, Gigabit Ethernet, y 10 Gigabit Ethernet) abandonaron los coaxiales dejando únicamente los cables de par trenzado sin apantallar (UTP - *Unshielded Twisted Pair*), de categorías 5 y superiores y la fibra óptica. Ethernet permite un buen equilibrio entre velocidad, costo y facilidad de instalación. Estos puntos fuertes, combinados con la amplia aceptación en el mercado y la habilidad de soportar virtualmente todos los protocolos de red populares, hacen a Ethernet la tecnología ideal para la red de la mayoría de usuarios de la informática actual.

Trama de Ethernet						
Preámbulo	SOF	Destino	Origen	Tipo	Datos	FCS
7 bytes	1 byte	6 bytes	6 bytes	2 bytes	1500 bytes	4 bytes

Ethernet industrial se está convirtiendo en el estándar dominante en la tecnología de redes industriales. Por esta razón, presenta un alto valor añadido para los planificadores y operadores de instalaciones industriales. Con estándar técnico de validez universal como base, los fabricantes combinan redes y las instalaciones que las albergan.

Aunque muchas aplicaciones y los productos correspondientes se definen en gran medida a través de estándares de comunicación, soluciones de software y protocolos de aplicación, aún se necesita una estructura física para redes completamente fiable. El 80% de los errores que se producen en una red industrial tiene su origen, en el más amplio sentido, en fallos en los contactos. Conectores que no están bien sujetos, problemas de contacto, humedad, roturas de cables, problemas de compatibilidad electromagnética o fallos de conexión son sólo algunas de las causas.

Con un nuevo equipo Ethernet y componentes nuevos podremos solucionar estos fallos y problemas.

1.1.5 OPC

El bus OPC (*Object Linking and Embedding for Process Control*) es un estándar de comunicación en el campo del control y supervisión de procesos. Este estándar permite que diferentes fuentes de datos envíen datos a un mismo servidor OPC, al que a su vez podrán conectarse diferentes programas compatibles con dicho estándar. De este modo se elimina la necesidad de que todos los programas cuenten con drivers para dialogar con múltiples fuentes de datos, basta que tengan un driver OPC.

En realidad OPC es un conjunto de protocolos entre los que se pueden destacar los siguientes:

- OPC-DA (Data Access).- El original, sirve para el intercambio de datos a tiempo real entre servidores y clientes.
- OPC-AE (Alarms & Events).- Proporciona alarmas y notificaciones de eventos.
- OPC B (Batch).- Útil en procesos discontinuos.
- OPC DX (Data eXchange).- Proporciona interoperabilidad entre varios servidores.
- OPC HDA (Historical Data Access).- Acceso histórico a datos OPC.
- OPC S (Security).- Especifica cómo controlar el acceso de los clientes a los servidores.
- OPC XML-DA (XML Data Access).- Es una combinación de OPC-XML (eXtensible Markup Language) y OPC-DA.
- OPC CD (Complex Data).- Permite a los servidores exponer y describir tipos de datos más complicados en forma de estructuras binarias y documentos XML.

Entre las desventajas de este estándar se pueden mencionar el problema de la seguridad, que aún no está resuelto y su dependencia del mundo Windows.

1.1.6 SCADA (Supervisory Control And Data Acquisition)

El SCADA, adquisición de datos y control de supervisión, se trata de una aplicación software especialmente diseñada para funcionar sobre computadoras en el control de producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, etc.) y controlando el proceso de forma automática desde la pantalla de la computadora. Además, provee de toda la información que se genera en el proceso productivo a diversos usuarios, tanto del mismo nivel como de otros supervisores dentro de la empresa: control de calidad, supervisión, mantenimiento, etc.

En este tipo de sistemas usualmente existe una computadora, que efectúa tareas de supervisión y gestión de alarmas, así como tratamiento de datos y control de procesos. La comunicación se realiza mediante buses especiales o redes LAN. Todo esto se ejecuta normalmente en tiempo real, y están diseñados para dar al operador de planta la posibilidad de supervisar y controlar dichos procesos.

Los programas necesarios, y en su caso el hardware adicional que se necesite, se denomina en general sistema SCADA.

Un paquete SCADA debe estar en disposición de ofrecer las siguientes prestaciones:

- Posibilidad de crear paneles de alarma, que exigen la presencia del operador para reconocer una parada o situación de alarma, con registro de incidencias.
- Generación de históricos de señal de planta, que pueden ser volcados para su proceso sobre una hoja de cálculo.
- Ejecución de programas, que modifican la ley de control, o incluso anular o modificar las tareas asociadas al autómata, bajo ciertas condiciones.
- Posibilidad de programación numérica, que permite realizar cálculos aritméticos de elevada resolución sobre la CPU de la computadora.

Con ellas, se pueden desarrollar aplicaciones para computadoras (tipo PC, por ejemplo), con captura de datos, análisis de señales, presentaciones en pantalla, envío de resultados a disco e impresora, etc.

Además, todas estas acciones se llevan a cabo mediante un paquete de funciones que incluye zonas de programación en un lenguaje de uso general (como C, Pascal, o Basic), lo cual confiere una potencia muy elevada y una gran versatilidad. Algunos SCADA ofrecen librerías de funciones para lenguajes de uso general que permiten personalizar de manera muy amplia la aplicación que desee realizarse con dicho SCADA.

Un SCADA debe cumplir varios objetivos para que su instalación sea perfectamente aprovechada:

- Deben ser sistemas de arquitectura abierta, capaces de crecer o adaptarse según las necesidades cambiantes de la empresa.
- Deben comunicarse con total facilidad y de forma transparente al usuario con el equipo de planta y con el resto de la empresa (redes locales y de gestión).
- Deben ser programas sencillos de instalar, sin excesivas exigencias de hardware, y fáciles de utilizar, con interfaces amigables con el usuario.

Los módulos o bloques software que permiten las actividades de adquisición, supervisión y control son los siguientes:

- Configuración: permite al usuario definir el entorno de trabajo de su SCADA, adaptándolo a la aplicación particular que se desea desarrollar.
- Interfaz gráfico del operador: proporciona al operador las funciones de control y supervisión de la planta. El proceso se representa mediante sinópticos gráficos almacenados en la computadora de proceso y generados desde el editor incorporado en el SCADA o importados desde otra aplicación durante la configuración del paquete.
- Módulo de proceso: ejecuta las acciones de mando preprogramadas a partir de los valores actuales de variables leídas.
- Gestión y archivo de datos: se encarga del almacenamiento y procesado ordenado de los datos, de forma que otra aplicación o dispositivo pueda tener acceso a ellos.

- Comunicaciones: se encarga de la transferencia de información entre la planta y la arquitectura hardware que soporta el SCADA, y entre ésta y el resto de elementos informáticos de gestión.

1.2 Problemática

La antigua celda de manufactura que se utilizaba en el Tecnológico de Monterrey, Campus Ciudad de México no cumple con los requerimientos actuales para la automatización. Las computadoras que la manipulan tienen versiones de Windows 95 y el equipo es viejo, no es suficiente ni el espacio en el disco duro ni la capacidad de memoria RAM.

El PLC que maneja a la celda es antiguo y no es posible reprogramarlo por medio de una computadora, debe modificarse con un control. Los robots son programados con código de bajo nivel, lo cual hace difícil su programación para las personas que desconocen este código y no es posible monitorear éstos robots. Por último, el sistema de visión debe ser actualizado ya que la pantalla es en blanco y negro y la tecnología implementada es obsoleta.

Es necesario actualizar la celda de modo que tenga un óptimo funcionamiento ya que, de no ser así quedaría sin uso debido a que se cuenta con una nueva.

1.3 Justificación

En estos momentos la antigua celda de manufactura del Tecnológico de Monterrey, Campus Ciudad de México se encuentra fuera de servicio. Sin embargo, hay elementos de la celda que funcionan aún de forma correcta como los robots industriales, la banda transportadora y los sensores y actuadores de la banda.

Se busca rediseñar la antigua celda con el fin de que pueda seguir siendo utilizada para fines didácticos en el área de Redes Industriales y Automatización. Esto brindará a la universidad un espacio más para capacitar a los alumnos en el manejo de este tipo de maquinaria explorando los sistemas más actuales que existen en el campo de una manera sencilla.

Este proyecto puede ser aplicado en México ya que varias industrias no cuentan con la tecnología de punta, más bien tienen máquinas que no han sido reemplazadas desde hace varios años. Al proponer un rediseño de una línea de producción como el nuestro se amplían las posibilidades de mejorar la producción y contar con las ventajas de tecnología actual sin invertir grandes cantidades de dinero.

1.4 Objetivos

- Implementar un sistema en tiempo real con enfoque industrial para controlar un proceso
- Crear una aplicación que contenga una interfaz gráfica para desarrollar programas para el control de robots y monitorearlos en tiempo real
- Realizar el diseño de un diagrama electro-neumático
- Lograr el control total de las funciones de la celda mediante una sola computadora a la que se llamará computadora de monitoreo.
- Establecer una comunicación Ethernet entre la computadora de monitoreo y los robots para envío de señales de control y cambio de programas
- Cambiar el sistema de visión
- Proporcionar un enlace de comunicación con el sistema de monitoreo y control SCADA

2 Metodología

El proceso para la realización de nuestro prototipo consta de seis bloques esenciales como se muestra en la figura 2a. Para la implementación del proceso se realizó lo siguiente:

- Estudio de programación en C#
- Análisis de comunicación para los robots por medio de sus manuales
- Recableado y reacomodado de los componentes que integran la Celda de Manufactura
- Implementación de equipo nuevo (panel de monitoreo, PLC)
- Estudio de OPC.

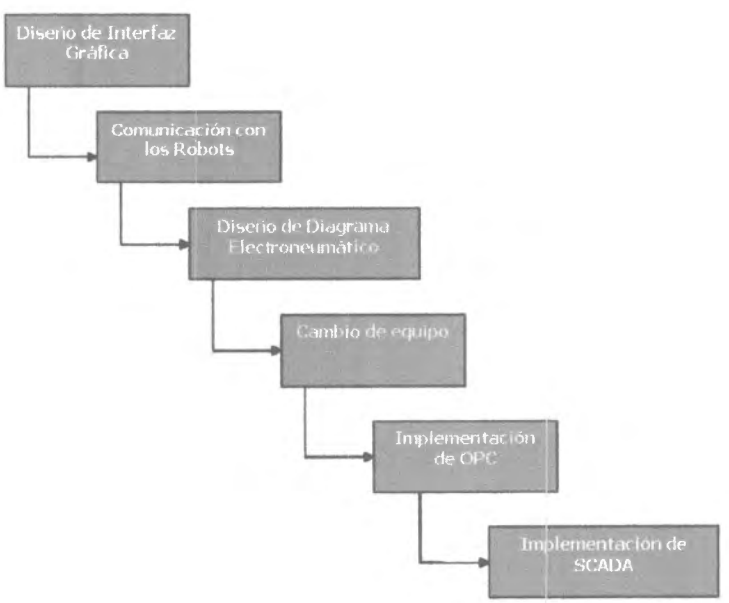


Figura 2a. Etapas en el desarrollo del sistema automatizado

2.1 Programa para controlar a los robots

El diagrama de flujo de la figura 2b describe el funcionamiento del programa internamente. Esto se pensó siguiendo una lógica para establecer el orden, los elementos que están involucrados con cada operación, la relación con cada robot y con el sistema de monitoreo vía Ethernet.

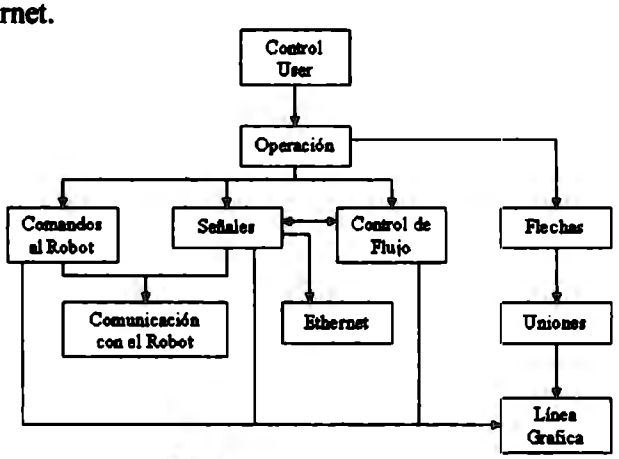


Figura 2b. Diagrama de Flujo de funcionamiento interno del programa

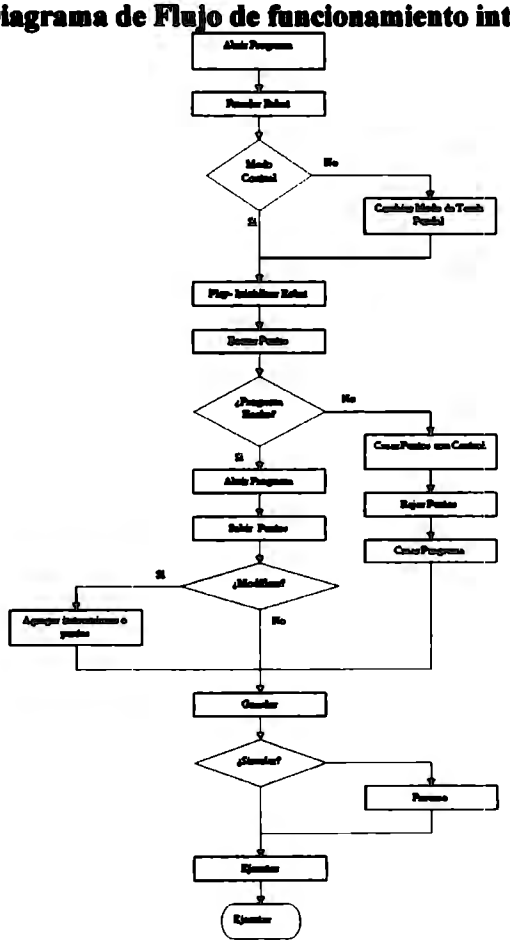


Figura 2c. Pasos a seguir para programar a los robots

El diagrama de flujo de la figura 2c explica los pasos que se deben seguir para realizar un programa para los robots desde el encendido de los mismos. Se analizan los casos en que se trabaja con un archivo existente o el archivo es nuevo y se explica el manejo de los puntos en las diferentes etapas de realización del programa.

2.1.1 Interfaz Gráfica

Para facilitar la forma de programar a los robots de la celda de manufactura se creó un programa basado en la programación orientada a objetos. Definir interfases hombre-máquina "a-la-Windows" suele ser bastante conveniente, es por eso que se utilizó C# como un programa en el cual se pudo crear una interfaz sencilla y fácil de usar para la utilización didáctica de la celda de manufactura.

2.1.1.1 Programación orientada a objetos

La orientación a objetos promete mejoras de amplio alcance en la forma de diseño, desarrollo y mantenimiento del software ofreciendo una solución a largo plazo a los problemas en el desarrollo de software: la falta de portabilidad del código y su reusabilidad, la dificultad de modificar código, la extensión de ciclos de desarrollo y las técnicas de codificación no intuitivas.

Un lenguaje orientado a objetos ataca estos problemas. Tiene tres características básicas: debe estar basado en objetos, basado en clases y capaz de tener herencia de clases.

El concepto de programación orientada a objetos (POO) se basa en la idea natural de la existencia de un mundo lleno de objetos, la resolución del problema se realiza en términos de los mismos. Se dice que un lenguaje está basado en objetos si soporta objetos como una característica fundamental del mismo.

Un objeto es un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.

Esta definición especifica varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

Un objeto puede considerarse como una especie de cápsula dividida en tres componentes que desempeñan un papel totalmente independiente:

1.- Relaciones
Las relaciones permiten que el objeto se inserte en la organización y están formadas esencialmente por punteros a otros objetos.

2.- Propiedades

Las propiedades distinguen un objeto determinado de los restantes que forman parte de la misma organización y tiene valores que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización.

3.- Métodos

Los métodos son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (código) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia.

Cada objeto es una estructura compleja en cuyo interior hay datos y programas, todos ello relacionados entre sí, como si estuvieran encerrados conjuntamente en una cápsula. Esta propiedad (encapsulamiento), es una de las características fundamentales en la POO.

Los objetos son inaccesibles, e impiden que otros objetos, los usuarios, o incluso los programadores conozcan cómo está distribuida la información o qué información hay disponible. Esta propiedad se denomina ocultación de la información.

Esto no quiere decir, sin embargo, que sea imposible conocer lo necesario respecto a un objeto y a lo que contiene. Si así fuera no se podría hacer gran cosa con él. Lo que sucede es que las peticiones de información a un objeto deben realizarse a través de mensajes dirigidos a él, con la orden de realizar la operación pertinente. La respuesta a estas órdenes será la información requerida, siempre que el objeto considere que quien envía el mensaje está autorizado para obtenerla.

El hecho de que cada objeto sea una cápsula facilita enormemente que un objeto determinado pueda ser transportado a otro punto de la organización, o incluso a otra organización totalmente diferente que precise de él. Si el objeto ha sido bien construido, sus métodos seguirán funcionando en el nuevo entorno sin problemas. Esta cualidad hace que la POO sea muy apta para la reutilización de programas.

Los objetos son portables mientras que la herencia permite la reusabilidad del código orientado a objetos, es más sencillo modificar código existente porque los objetos no interaccionan excepto a través de mensajes; en consecuencia un cambio en la codificación de un objeto no afectará la operación con otro objeto siempre que los métodos respectivos permanezcan intactos. La introducción de tecnología de objetos como una herramienta conceptual para analizar, diseñar e implementar aplicaciones permite obtener aplicaciones más modificables, fácilmente extendibles y a partir de componentes reusables. Esta reusabilidad del código disminuye el tiempo que se utiliza en el desarrollo y hace que el desarrollo del software sea mas intuitivo porque la gente piensa naturalmente en términos de objetos más que en términos de algoritmos de software.

2.1.1.2 Ambiente Gráfico

Gracias al lenguaje C# se creó una interfaz mediante la cual el usuario puede, de una manera sencilla, programar la rutina que seguirán los robots de la celda de manufactura.

Para establecer los comandos de la interfaz se pensó en las actividades principales que cada brazo robótico (Puma y Mitsubishi) realizaba y en las instrucciones que se le dan a los mismos para cumplir con sus tareas, tales como son el moverse a cierto punto, el abrir o cerrar su gripper para tomar algún objeto o incluso el cambiar su velocidad. De este modo se diseñó una ventana principal y los íconos que debían aparecer en la misma para el óptimo funcionamiento del programa.

Se utilizó la misma interfaz tanto para el Puma como para el Mitsubishi con ciertas modificaciones de acuerdo a la forma de establecer la comunicación con los mismos.

En nuestro programa principal existen 6 barras de herramientas que se mencionan a continuación con cada uno de sus botones.
NOTA: Una descripción más detallada se puede encontrar en la sección de anexos, parte 7.1.6.1.

- Controles: Esta barra consta de 6 botones: Punto, Abrir gripper, Cerrar gripper, Retardo, Velocidad y Home.

- Controles de señales: Existen dos señales, una de entrada y una de salida. Estas señales se adquieren del PLC cuando se establece una comunicación directa entre él y el robot por medio de Ethernet para controlarlas con el sistema SCADA.

- Controles de flujo: Estos controles ofrecen la posibilidad de crear condiciones en el programa, así como de permitir unir dos procesos paralelos en un punto determinado.

- Manejo de puntos: Esta barra de herramientas cuenta con 3 botones: Subir puntos, Bajar puntos y Borrar puntos.

- Manejo de archivos: Contiene 3 botones: Nuevo, Abrir y Guardar.

- Reproducción: Es con estos cuatro botones que se puede establecer la comunicación con los robots y correr el programa creado.

Existe un panel de trabajo en el cual se crea la secuencia que el robot va a seguir, eligiendo los puntos a los que irá y los procesos del mismo. Sobre este panel de trabajo siempre que se crea un nuevo archivo aparece el icono de "Inicio" ya que al correr una aplicación el robot debe ir a su posición de inicialización para después seguir una secuencia. Las operaciones se unen por medio de flechas que establecerán la secuencia de los mismos. Un panel ubicado a la derecha de la ventana de usuario es el que despliega los puntos existentes.

El panel de trabajo incluye operaciones comunes de Windows tales como el arrastrado de un objeto dentro del área de trabajo, copiar un objeto por medio del arrastrado y la tecla CTRL, eliminar objetos con la tecla Suprimir, aceptar una operación con el botón Enter y cancelar una operación con la tecla Esc.

Para el brazo robótico Puma los puntos se deben crear a partir de la ventana de usuario, por medio de un botón llamado "Crear punto", por medio de el cual se puede enumerar a los diferentes puntos que se vayan creando para después bajarlos, de forma que éstos aparezcan en pantalla para su uso. Para el brazo robótico Mitsubishi los puntos pueden crearse directamente desde el controlador del robot y después en la ventana de usuario se bajan los puntos para trabajar con los mismos.

2.1.1.3 Información general de las clases y métodos de la Interfaz Gráfica

UserControl

Las operaciones utilizadas para el control de los robots se agrupan en tres conjuntos: comandos, señales y controles de flujo. Dentro del programa cada uno cuenta con un icono propio para ser creado. Todos estos cuentan con características similares como flechas de entrada y de salida, una imagen para desplegar, un tamaño, una localización y responden a los mismos eventos, tales como arrastrar, unir, copiar, suprimir y dar doble clic.
Para implementar estas características en el programa, las operaciones heredaron (adquirieron todas las propiedades de otra clase) de la clase "UserControl". Esta clase maneja propiedades como las que se utilizan en los controles de Windows.

Operación

La clase que hereda de la clase "UserControl" en este programa es "Operación". Además de las propiedades y eventos de "UserControl", en "Operacion" se agrega la característica de las flechas. Estas flechas representan los puntos de entrada y salida en que se pueden establecer uniones entre controles. Los métodos que utiliza "Operación" son:

Operacion (ArrayList lista, Form Mitsubishi formarec, Comunicacion comrec)
Las propiedades y eventos que se utilizan y son inicializados en el constructor de "Operación" son:

- Location (localización).- Define la ubicación del control relativa a su esquina superior izquierda y la esquina superior izquierda del área de trabajo. Está definido por un punto (dos coordenadas enteras *x,y*) y cada unidad en la coordenada del punto representa un píxel. La posición en x al ser creado es al centro del área de trabajo y la posición en y es relativa al control ubicado más abajo en el área de trabajo.

- Size (tamaño).- Define el tamaño en pixeles del control. Está formado por dos propiedades: *width* y *height*. Estas dos deben ser enteras. El tamaño de los controles en esta aplicación es de 66 por 66 pixeles.
- TabStop y TabIndex.- La primera indica si este control se puede seleccionar con la tecla Tab. TabIndex determina en qué orden se selecciona. El orden que se define es conforme los controles se van creando en el programa.
- MouseDown.- Cuando ocurra el evento MouseDown en esta operación, se invoca el método EventMouseDown.
- MouseMove.- Cuando ocurra el evento MouseDown en esta operación, se invoca el método EventMouseDown.
- MouseUp.- Cuando ocurra el evento MouseUp en esta operación, se invoca el método EventMouseUp.
- DragOver.- Cuando ocurra el evento DragOver en esta operación, se invoca el método EventDragOver.
- DragDrop.- Cuando ocurra el evento DragDrop en esta operación, se invoca el método EventDragDrop.
- KeyDown.- Cuando ocurra el evento KeyDown en esta operación, se invoca el método EventKeyDown.
- Enter.- Cuando ocurra el evento Enter en esta operación, se invoca el método EventEnter.
- Leave.- Cuando ocurra el evento Leave en esta operación, se invoca el método EventLeave.
- DoubleClick.- Cuando ocurra el evento DoubleClick en esta operación, se invoca el método EventDoubleClick.

Además, el constructor recibe la referencia a tres objetos: una lista (listarec), una forma (formarec) y un objeto de comunicación (comrec). Las referencias a la forma y al objeto de comunicación son almacenadas en las variables llamadas "forma" y "com" respectivamente definidas dentro de esta clase. La operación que se crea en el constructor se agrega a la lista. Esta lista se utiliza para determinar cual de todas las operaciones se encuentra más abajo en el área de trabajo. La lista también se utiliza para tener una referencia a las operaciones que se van a guardar en un archivo. La referencia a forma sirve para agregar el control al área de trabajo.

EventMouseDown (object sender, MouseEventArgs e)
Este método se activa al momento de ocurrir el evento de presionar un botón del ratón sobre el control. En éste evento se almacena el punto en que se dio clic sobre el control y se crea un rectángulo alrededor de este punto que determina a partir de que lugar se considera como un arrastre. El tamaño del rectángulo de arrastre se obtiene del sistema. El parámetro "sender" no se utiliza, pero es requerido por el sistema al tratarse de un evento. En el parámetro "e" están las coordenadas hasta las cuales fue movido el puntero del ratón. Estas coordenadas son relativas a la pantalla, por lo que se convierten a coordenadas relativas al control por medio de la función "PointToClient(Point(e.X, e.Y))".

EventMouseMove(object sender, MouseEventArgs e)
Este método se invoca cuando (por medio de un evento) se mueve el ratón mientras alguno de sus botones esté presionado. En este método se verifica si el puntero del ratón salió del rectángulo de arrastre y si esto se hizo con el botón izquierdo del ratón presionado. En caso de haber salido del rectángulo, se determina si se empezó a arrastrar desde una flecha del control o en otra región. Si se empezó a arrastrar en una flecha se utiliza el método "DoDragDrop" para indicarle al sistema que se está arrastrando para crear la unión entre dos flechas. Si el arrastre no fue sobre una flecha se utiliza "DoDragDrop" para indicarle al sistema que se está arrastrando el control. Debido a que existen varios tipos de operaciones, la operación se encapsula en otra clase llamada "Caja". El parámetro "sender" no se utiliza, pero es requerido por el sistema al tratarse de un evento. En el parámetro "e" están las coordenadas hasta las cuales fue movido el puntero del ratón. Estas coordenadas son relativas a la pantalla, por lo que se convierten a coordenadas relativas al control por medio de la función "PointToClient(Point(e.X, e.Y))".

EventMouseUp(object sender, MouseEventArgs e)
Evento en el que se borra el rectángulo de arrastre para poder ser utilizado en otro arrastre. Los 2 parametros recibidos son requeridos por el sistema, pero no son implementados.

EventDragOver (object sender, DragEventArgs e)
Este método es llamado cuando se ejecuta el evento DragOver, el cual ocurre cuando se e arrastra un objeto sobre el control sin ser soltado. El objeto de este evento es indicarle al sistema que objetos pueden ser arrastrados a este control. En caso de ser una flecha se verifica si el ratón se encuentra sobre otra flecha, y de ser así, se le indica al sistema que es un arrastre válido entre la flecha arrastrada y la flecha de este control si se cumplen las condiciones siguientes: que una flecha sea de entrada y la otra de salida, que se trate de dos operaciones diferentes (no se puede unir un control a si mismo), que ninguna de las dos flechas haya sido unida previamente con otro control y que sean del mismo tipo (flujo o señal). Si no está siendo arrastrada una flecha se verifica si el objeto arrastrado es de tipo operación (encapsulado en una caja) y, de ser así, se le indica al sistema que no se permite el arrastre para evitar que se encime una operación sobre otra. Solo se permite este arrastre si la operación arrastrada está siendo arrastrada sobre ella misma (para permitir desplazar el control pequeñas distancias). El parámetro "sender" no se utiliza, pero es requerido por el sistema al tratarse de un evento. En el parámetro "e" están las coordenadas hasta las cuales fue arrastrado el objeto. Estas coordenadas son relativas a la pantalla, por lo que se convierten a coordenadas relativas al control por medio de la función "PointToClient(Point(e.X, e.Y))". Además, el parámetro "e" contiene el objeto que esta siendo arrastrado.

EventDragDrop (object sender, DragEventArgs e)
Por medio de este método se detecta cuando se suelta un objeto sobre este control. El objeto arrastrado debió ser autorizado al sistema anteriormente en el método EventDragOver. Se crea la unión entre dos flechas si el objeto arrastrado fue una flecha y se cumplieron las condiciones establecidas en EventDragOver. Si fue la misma caja, se

establecen las nuevas coordenadas de la variable "Location". El parámetro "sender" no se utiliza, pero es requerido por el sistema al tratarse de un evento. En el parámetro "e" están las coordenadas hasta las cuales fue arrastrado el objeto. Estas coordenadas son relativas a la pantalla, por lo que se convierten a coordenadas relativas al control por medio de la función "PointToClient(Point(e.X, e.Y))". Además, el parámetro "e" contiene el objeto que esta siendo arrastrado.

EventKeyDown (object sender, KeyEventArgs e);
Este método se llama al ocurrir el evento de presionar una tecla. En caso de que la tecla oprimida sea "Supr" o "Del" se borra la operación. Para borrarlo se remueve del área de trabajo y de la lista. También se borran todas las uniones que existan entre ésta operación y otras del área de trabajo. El parámetro "sender" no se utiliza, pero es requerido por el sistema al tratarse de un evento. En el parámetro "e" esta contenida la tecla presionada.

EventEnter (object sender, EventArgs e)
Este método se ejecuta al ser seleccionado el control. Esto puede ocurrir al dar clic sobre el control, por medio de la tecla "Tab" o por medio del comando "focus". Al ocurrir esto se invoca al método "Invalidate()". El objetivo de "Invalidate()" es invocar al método "OnPaint" con el cual se vuelve a pintar el control. Al estar seleccionado este control, se pinta un contorno amarillo.

EventLeave (object sender, EventArgs e)
Este método se ejecuta al ser seleccionado otro control. Esto puede ocurrir al dar clic sobre otro control, por medio de la tecla "Tab" o por medio del comando "focus". Al ocurrir esto se invoca al método "Invalidate()". Al seleccionar otro control, se borra el contorno amarillo.

OnPaint (PaintEventArgs e)
Éste método es heredado de "UserControl". Se ejecuta cada vez que es necesario volver a pintar el control (es un evento). Algunas de las causas por las cuales se debe volver a pintar el control son: si sale y vuelve a entrar a la pantalla y si se coloca otra ventana sobre el control. Este evento además puede ser generado con el comando "Invalidate()" mencionado anteriormente en EventEnter e EventLeave.
En este método se pinta el contorno de amarillo si el control se encuentra seleccionado, se pinta de naranja si el control esta siendo ejecutado, y además se dibuja la imagen y las flechas de acuerdo a la operación correspondiente.

Operacion Copiar (ArrayList lista, Form Mitsubishi formarec, Comunicacion comrec)
Este método es una plantilla que implementan los controles que heredan de Operación. En este método se crea un objeto del mismo tipo y se copian los parámetros que contenga. Regresa la dirección del objeto creado.

Accion()
Este método es una plantilla que implementan los controles que heredan de Operación. En este método se realiza la acción específica a cada Operación (por ejemplo: abrir gripper, cambiar velocidad, etc.). Mientras se ejecuta este método, la variable *accionado*,

de tipo booleano, cambia a *true* para que en el método OnPaint() el control se pinte con un contorno naranja.

Operacion GetSiguiente()
Este método se utiliza para obtener la Operación unida a este control por medio de la flecha de salida. En caso de no existir un control de tipo Operación unido a este control, regresa *null*. Este método es sobrescrito por la clase partir, debido a que partir tiene dos flechas de salida.

Bool VerificarZonaFlecha(Point punto)
Este método es llamado por los métodos EventDragOver y EventDragDrop para verificar si en el punto recibido se encuentra una flecha. Regresa *true* si es que existe una flecha o *false* si no existe una flecha. Para verificar si existe una flecha debajo del punto, se invoca el método "Flecha.VerificarZonaFlecha(punto)" para cada flecha que contiene esta operación.

Flecha GetFlecha (Point punto)
Este método es llamado por los métodos EventDragOver y EventDragDrop para saber que flecha de control se encuentra bajo el punto recibido. Siempre debe ser llamado después de verificarse una flecha. Regresa la dirección de la flecha que se encuentra debajo del punto. Este método invoca el método "Flecha.VerificarZonaFlecha(punto)" para cada flecha que contiene esta operación. Si ese método regresa *true*, se regresa la dirección de esa flecha.

ActualizarUnion ()
Si un control es arrastrado hacia otra zona del área de trabajo y se encontraba unido a otro control este método actualiza las líneas de unión entre ambos controles. Este método invoca el método "Flecha.unida.ActualizarUnion()" para cada flecha que contiene esta operación.

Flecha

La clase flecha establece una entrada o salida para un objeto de tipo "Operación". La flecha solo puede ser dibujada en doce posiciones, tres en cada lado del control.

Flecha (Location loc, Direccion dir, Operacion ope, int num, Tipo tip)
Este constructor recibe 5 parametros:
loc, en el cual se especifica en cual de las 12 posiciones debe ser dibujada. Se almacena en la variable *location*.
dir, en el cual se especifica si es de entrada o salida. Se almacena en la variable *direccion*.
ope, en el cual se especifica que objeto de tipo "Operación" lo contiene. Se almacena en la variable *operacion*.
num, en el cual se especifica que número de flecha es dentro de operación. Se almacena en la variable *numflecha*.

tip, en el cual se especifica si la union es de flujo o de señal. Se almacena en la variable *tipo*.

Dibujar(Graphics g, Pen p)
Este método es invocado por el método OnPaint() por parte de un objeto que hereda de "Operación". Dibuja las flechas del control. Basado en las variables *direccion* y *location*, dibuja la flecha en un lugar (punto), en una direccion (apuntando hacia fuera o hacia adentro) y en una orientación(horizontal o vertical).

bool VerificarZonaFlecha(Point puntorec)
Este método es llamado por "Operacion.VerificarZonaFlecha()" para ver si en ese punto existe una flecha.

Point GetPunto()
Este método es llamado por un objeto de tipo "Union" para determinar desde que punto debe dibujarse la línea que une dos controles.

RemoverUnion()
Este método es llamado por el método EventKeyDown de "Operacion" para borrar las líneas que unen el control ademas del control. También borra las direcciones almacenadas en "Flecha" de con quien estaba unido.
Dentro de este método se invoca el método "Borrar()" de la clase "Union()". La dirección de la unión está almacenada en la variable *unida*.

Comandos – Inicio

"Inicio" pertenece al grupo Comandos. Esta clase hereda de "Operación", pero no implementa los siguientes métodos debido a que solo se puede tener un "inicio": Copiar (lista, forma, com) para evitar copiar este control y EventKeyDown para no poder suprimir este control.

Inicio (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)
En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crea la siguiente flecha:
flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo)

Accion()
Este método, heredado de "Operación", es sobrescrito para que "Inicio" pueda llamar a "com.Iniciar()" en este método se inicializa al robot.

Comandos – Punto

"Punto" pertenece al grupo Comandos y sirve para mover al robot al punto deseado. Esta clase hereda de "Operación", e implementa el método EventDoubleClick para establecer la variable de tipo *string* llamada *p*. Al método OnPaint heredado de "Operacion" se le agrega código para que imprima la variable *p* sobre el control. Se creó una ventana que se despliega al dar doble clic para que el usuario cambie el punto al que se mueve.

Punto (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)
En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crean las siguientes flechas:
flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo);
flecha[1] = new Flecha(Flecha.Location.Norte2, Flecha.Direccion.Entrada,this, 1, Flecha.Tipo.Flujo);

Accion()
Este método, heredado de "Operación", es sobrescrito para que "Punto" pueda llamar a "com.MoverPunto ()" en este método se mueve a un punto del robot.

Operacion Copiar(ArrayList lista, Mitsubishi formarec, Comunicacion comrec)
En este método se crea un nuevo objeto "Punto" y se copia la variable *p* al nuevo objeto.

Comandos – Abrir

"Abrir" pertenece al grupo Comandos y sirve para abrir el gripper del robot. Esta clase hereda de "Operación".

Abrir (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)
En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crean las siguientes flechas:
flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo);
flecha[1] = new Flecha(Flecha.Location.Norte2, Flecha.Direccion.Entrada, this, 1, Flecha.Tipo.Flujo);

Accion()
Este método, heredado de "Operación", es sobrescrito para que "Abrir" pueda llamar a "com.AbreGriper ()" en este método se abre el gripper del robot.

Comandos – Cerrar

"Cerrar" pertenece al grupo Comandos y sirve para cerrar el gripper del robot. Esta clase hereda de "Operación".

Cerrar (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)
En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crean las siguientes flechas:
flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo);
flecha[1] = new Flecha(Flecha.Location.Norte2, Flecha.Direccion.Entrada, this, 1, Flecha.Tipo.Flujo);

Accion()
Este método, heredado de "Operación", es sobrescrito para que "Cerrar" pueda llamar a "com.CierraGriper ()" en este método se cierra el gripper del robot.

Comandos – Velocidad

"Velocidad" pertenece al grupo Comandos y sirve para establecer la velocidad a la que trabajará el robot. Esta clase hereda de "Operación", e implementa el método EventDoubleClick para establecer la variable de tipo *string* llamada *v*. Al método OnPaint heredado de "Operacion" se le agrega código para que imprima la variable *v* sobre el control. Se creó una ventana que se despliega al dar doble clic para que el usuario cambie la velocidad del robot.

Velocidad (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)
En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crean las siguientes flechas:
flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo);
flecha[1] = new Flecha(Flecha.Location.Norte2, Flecha.Direccion.Entrada,this, 1, Flecha.Tipo.Flujo);

OnPaint(PaintEventArgs e)
Este método se sobrescribe para incluir código que pinte el número que representala velocidad elegida.

Accion()
Este método, heredado de "Operación", es sobrescrito para que "Retardo" pueda llamar a "com.Velocidad ()" en este método se mueve a un punto del robot.

Comandos – Retardo

"Retardo" pertenece al grupo Comandos y sirve para establecer un tiempo de retardo del robot. Esta clase hereda de "Operación", e implementa el método EventDoubleClick para establecer la variable de tipo *string* llamada *t*. Al método OnPaint heredado de "Operacion" se le agrega código para que imprima la variable *t* sobre el control. Se creó una ventana que se despliega al dar doble clic para que el usuario cambie el tiempo de retardo del robot.

Retardo (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)
En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crean las siguientes flechas:
flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo);
flecha[1] = new Flecha(Flecha.Location.Norte2, Flecha.Direccion.Entrada,this, 1, Flecha.Tipo.Flujo);

OnPaint(PaintEventArgs e)
Este método se sobrescribe para incluir código que pinte el número que representa el tiempo que resta de esta operación.

Accion()
Este método, heredado de "Operación", es sobrescrito para que "Retardo" mantenga al robot sin realizar acción alguna.

Comandos – Home

"Home" pertenece al grupo Comandos y sirve para enviar al robot a su posición de inicio. Esta clase hereda de "Operación".

Home (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)
En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crean las siguientes flechas:
flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo);
flecha[1] = new Flecha(Flecha.Location.Norte2, Flecha.Direccion.Entrada, this, 1, Flecha.Tipo.Flujo);

Accion()
Este método, heredado de "Operación", es sobrescrito para que "Home" pueda llamar a "com.Iniciar()" en este método se manda al robot a la posición de inicio.

Señales - Entrada

"Entrada" pertenece al grupo Señales y sirve para establecer si se recibe o no una señal de entrada. Esta señal puede provenir del PLC o de Ethernet industrial. Esta clase hereda de "Operación".

Entrada (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)
En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crean las siguientes flechas:
flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo);
flecha[1] = new Flecha(Flecha.Location.Norte2, Flecha.Direccion.Entrada, this, 1, Flecha.Tipo.Flujo);

Señales – Salida

"Salida" pertenece al grupo Señales y sirve para transmitir una señal al PLC o a Ethernet industrial. Esta clase hereda de "Operación".

Salida (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)
En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crean las siguientes flechas:
flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo);
flecha[1] = new Flecha(Flecha.Location.Norte2, Flecha.Direccion.Entrada, this, 1, Flecha.Tipo.Flujo);

Controles de Flujo – Esperar por

"Esperarpor" pertenece al grupo Controles de Flujo y sirve para detener el flujo hasta que se reciba una señal de entrada. Esta clase hereda de "Operación".

Esperarpor (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)
En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crean las siguientes flechas:
flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo);
flecha[1] = new Flecha(Flecha.Location.Norte2, Flecha.Direccion.Entrada, this, 1, Flecha.Tipo.Flujo);

flecha[2] = new Flecha(Flecha.Location.Oeste2, Flecha.Direccion.Entrada, this, 2, Flecha.Tipo.Señal);

Accion()

Este método, heredado de "Operación", es sobrescrito para que "Esperarpor" pueda verificar constantemente la señal de entrada hasta que ésta le permita proseguir.

Controles de Flujo – Partir

"Partir" pertenece al grupo Controles de Flujo. Esta clase hereda de "Operación".

Partir (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)

En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crean las siguientes flechas:

flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo);

flecha[1] = new Flecha(Flecha.Location.Norte1, Flecha.Direccion.Entrada, this, 1, Flecha.Tipo.Flujo);

flecha[2] = new Flecha(Flecha.Location.Norte3, Flecha.Direccion.Entrada, this, 2, Flecha.Tipo.Flujo);

Accion()

Este método, heredado de "Operación", es sobrescrito para que "Partir" pueda seleccionar una de dos posibles ramificaciones en base a la señal que tenga.

Controles de Flujo – Unir

"Unir" pertenece al grupo Controles de Flujo. Esta clase hereda de "Operación".

Unir (ArrayList lista, Mitsubishi formarec, Comunicacion comrec):base(lista, formarec, comrec)

En este constructor se invoca al constructor de "Operacion" y además se carga la imagen que se va a desplegar. Esta imagen se almacena en la variable *MyBitmap* de la clase "Operacion" de la cual hereda. Se crean las siguientes flechas:

flecha[0] = new Flecha(Flecha.Location.Sur2, Flecha.Direccion.Salida, this, 0, Flecha.Tipo.Flujo);

flecha[1] = new Flecha(Flecha.Location.Norte1, Flecha.Direccion.Entrada, this, 1, Flecha.Tipo.Flujo);

flecha[2] = new Flecha(Flecha.Location.Norte3, Flecha.Direccion.Entrada, this, 2, Flecha.Tipo.Flujo);

Accion()

Este método, heredado de "Operación", es sobrescrito para que "Unir" pueda juntar dos flujos.

Unión

Esta clase hereda de la clase "UserControl" y sirve para crear la parte gráfica de una "Unión" entre dos flechas de operaciones.

Union(Flecha flecha1, Flecha flecha2)
Se crean 5 líneas para unir las dos flechas en base a las coordenadas de las dos flechas que se adquieren por medio del método Flecha.GetPunto y se guardan referencias a las flechas involucradas en la "Union".

ActualizarUnion()
Este método se invoca cuando un control de tipo "Operacion" es arrastrado, de forma que la parte gráfica de la unión debe ser pintada en nuevas posiciones.

Borrar()
Este método es invocado por Operacion.EventKeyDown el cual borra un control de tipo operación y sus líneas (éste método).

Representa la unión conceptual entre las flechas de dos controles

Línea

Esta clase hereda de la clase "UserControl" y sirve para las propiedades como Location, Size y el evento OnPaint, Arrastrar y KeyDown.

Linea(Point p1, Point p2, Union uni, int num, Orientacion ori)
Este método recibe el punto de inicio y el punto final donde se crea la línea, a partir de esto se determinan sus Location y Size, se inicializan los eventos OnPaint, Arrastrar y KeyDown y tiene una referencia a la unión en la cual está almacenada.

Actualizar(Point p1, Point p2)
Se invoca este método cuando se tienen que volver a dibujar las líneas con nuevas coordenadas debido a que se arrastró algún control.

OnPaint(PaintEventArgs e)
Éste método es heredado de "UserControl". Se ejecuta cada vez que es necesario volver a pintar la línea (es un evento). Este evento además puede ser generado con el comando "Invalidate()" mencionado anteriormente en EventEnter e EventLeave.
En este método se pinta la línea de rojo si se encuentra seleccionada.
EventMouseDown (object sender, MouseEventArgs e)

Este método se activa al momento de ocurrir el evento de presionar un botón del ratón sobre la línea. En éste evento se almacena el punto en que se dio clic sobre la línea y se crea un rectángulo alrededor de este punto que determina a partir de que lugar se considera como un arrastre. El tamaño del rectángulo de arrastre se obtiene del sistema. El parámetro "sender" no se utiliza, pero es requerido por el sistema al tratarse de un evento. En el parámetro "e" están las coordenadas hasta las cuales fue movido el puntero del ratón. Estas coordenadas son relativas a la pantalla, por lo que se convierten a coordenadas relativas al control por medio de la función "PointToClient(Point(e.X, e.Y))".

EventMouseMove(object sender, MouseEventArgs e)
Este método se invoca cuando (por medio de un evento) se mueve el ratón mientras alguno de sus botones esté presionado. Se arrastra la línea El parámetro "sender" no se utiliza, pero es requerido por el sistema al tratarse de un evento. En el parámetro "e" están las coordenadas hasta las cuales fue movido el puntero del ratón.

EventMouseUp(object sender, MouseEventArgs e)
Evento en el que se borra el rectángulo de arrastre para poder ser utilizado en otro arrastre. Los 2 parámetros recibidos son requeridos por el sistema, pero no son implementados.

EventKeyDown (object sender, KeyEventArgs e);
Este método se llama al ocurrir el evento de presionar una tecla. En caso de que la tecla oprimida sea "Supr" o "Del" se borra la línea. Para borrarla se remueve del área de trabajo. El parámetro "sender" no se utiliza, pero es requerido por el sistema al tratarse de un evento. En el parámetro "e" esta contenida la tecla presionada.

EventEnter (object sender, EventArgs e)
Este método se ejecuta al ser seleccionada la línea. Esto ocurre al dar clic sobre la línea, y se invoca al método "Invalidate()". El objetivo de "Invalidate()" es invocar al método "OnPaint" con el cual se vuelve a pintar la línea de negro.

EventLeave (object sender, EventArgs e)
Este método se ejecuta al ser seleccionado otro control. Esto puede ocurrir al dar clic sobre otro control, por medio de la tecla "Tab" o por medio del comando "focus". Al ocurrir esto se invoca al método "Invalidate()" para que la línea se vuelva a pintar de negro.

Arrastrada(int x, int y)
Cuando se mueve la línea este método sirve para fijar las nuevas coordenadas.

2.1.1.4 Pruebas y resultados

Se crearon varias versiones del programa principal y se fueron modificando y mejorando conforme las pruebas de los mismos revelaban errores en su ejecución.

Al principio no era posible mover las líneas que unían un paso con el otro y los bloques no se repintaban cuando era necesario, esto ocasionaba que al mover la ventana los bloques no se desplazaran durante el movimiento sino después de minimizar la ventana y volverla a desplegar. Cuando se ejecutaba la secuencia creada no se desplegaba el paso que se estaba ejecutando debido a que no se repintaba la pantalla.

En la barra de herramientas de controles existía un botón de inicio, debido a esto se podían poner varios "inicios" dentro de la secuencia a ejecutar por el robot. En la versión final del programa se suprimió el botón inicio y en cada nuevo archivo se incluye siempre al principio del panel de trabajo el icono de inicio, lo que significa que siempre que el robot comienza una rutina va a "nest" o "do ready" antes de realizar lo demás.

En un inicio, las imágenes utilizadas para los controles en las barras de herramientas se obtenían en el código de nuestro programa de una carpeta específica, esto ocasionaba que si el programa se corría en otra computadora hubiera un fallo ya que el programa no encontraba la dirección de memoria en la que cada imagen había sido guardada. Para corregir esto se cambió la dirección en la que se encuentran las imágenes y ahora el programa busca las imágenes en donde se encuentra el ejecutable. Esto ayudó en la portabilidad del programa ya que puede utilizarse en cualquier computadora que contenga el ejecutable.

Al implementar la versión final del ejecutable del programa realizado el usuario puede:

- ✓ Moverse a alguno de los puntos existentes
- ✓ Cambiar la velocidad del robot
- ✓ Crear un retardo
- ✓ Abrir y cerrar el gripper
- ✓ Cargar y descargar puntos en el robot
- ✓ Controlar el flujo condicionalmente
- ✓ Abrir y guardar programas

Las figuras 2.1.5.a y 2.1.5.b muestran la apariencia de las ventanas con las que el usuario interactúa para poder programar a los robots Mitsubishi y Puma respectivamente.

Estas pantallas significan una posibilidad mucho más sencilla de programar a los robots sin la necesidad de conocer códigos de bajo nivel. De una forma visual y con una secuencia simple por bloques se puede construir una tarea muy completa para cada robot.

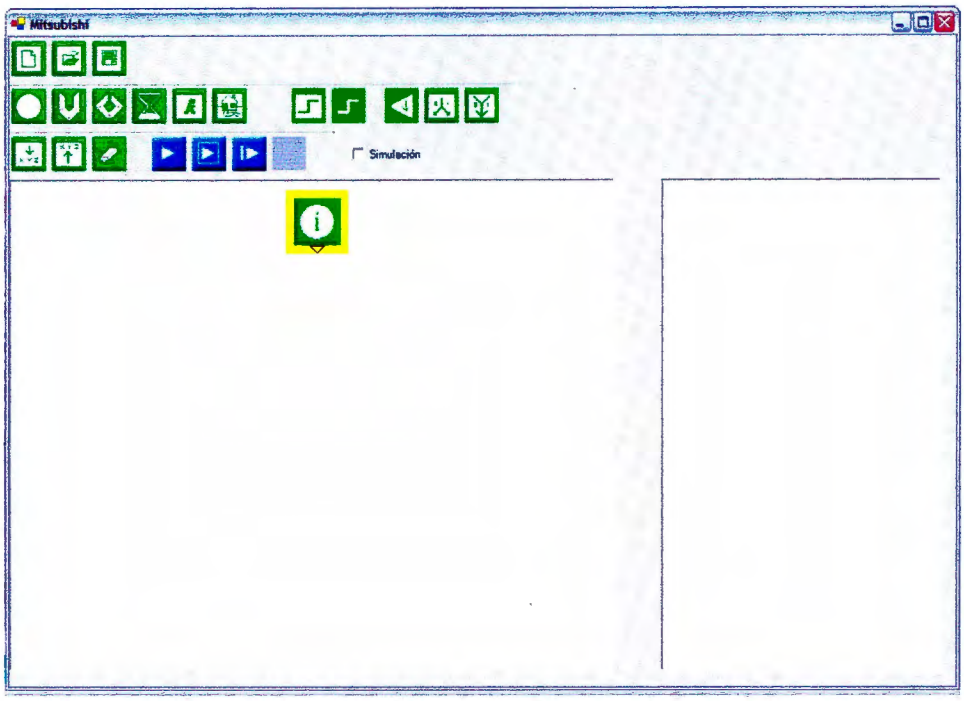


Figura 2.1.5 a. Ventana de usuario del robot Mitsubishi

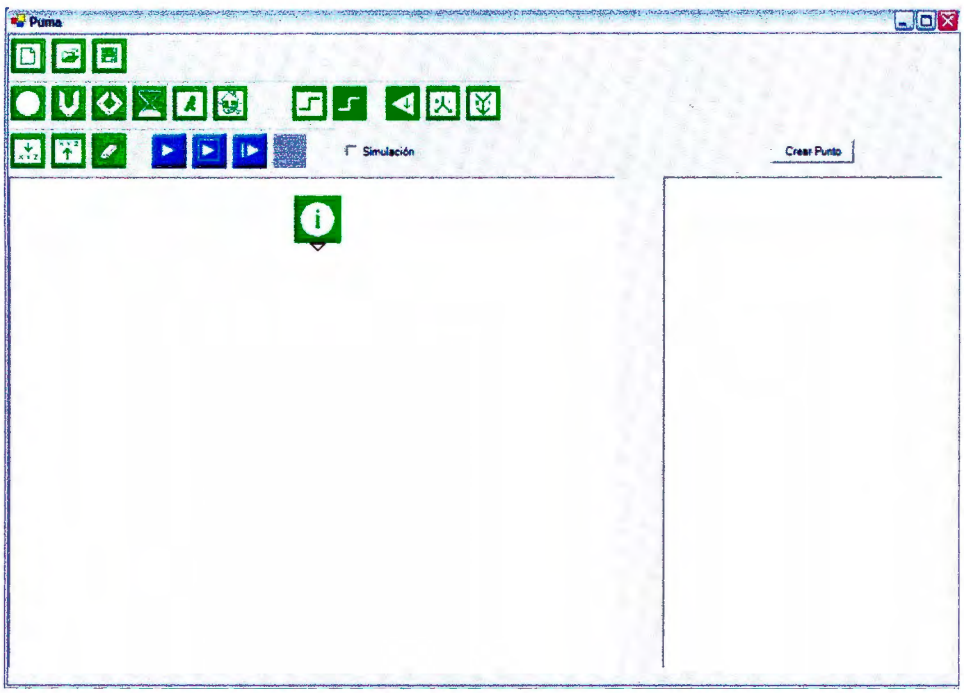


Figura 2.1.5 b. Ventana de usuario del robot Puma

Esta nueva forma de programar a los robots nos ofrece nuevas ventajas. La ejecución se realiza en tiempo real y desde la PC, esto permitirá implementar el programa en el sistema de monitoreo SCADA.

2.1.2 Comunicación con robots vía Puerto Serial

2.1.2.1 Funcionamiento de Puerto Serial

El puerto serial, también conocido por el estándar que lo norma, el RS-232, fue creado con el único propósito de contar con una interfaz entre los equipos terminales de datos (Data Terminal Equipment, DTE), y el equipo de comunicación de datos (Data Communications Equipment, DCE) empleando intercambio serial de datos binarios. De esta forma el equipo terminal de datos es el extremo cliente de los datos y el equipo de comunicación de datos es el dispositivo que se encarga de la unión entre los terminales, tal como un módem o algún otro dispositivo de comunicación.

El estándar RS-232 original especifica una velocidad máxima de 19,200 baudios y una longitud máxima de cable en 50 pies, aproximadamente 16 metros, lo cual resultaba conveniente para épocas pasadas; sin embargo, el paso del tiempo y la evolución de la tecnología obligaron el aumento de estos parámetros, emergiendo el RS422 y el RS485, que utilizan líneas balanceadas para eliminar algunos problemas que se presentan a mayores velocidades de transmisión.

La característica especial del RS-232, y que lo hiciera popular en el mundo de las computadoras es su diseño simple, en el cual los datos viajan como voltajes referidos a una tierra común, haciendo factible que pueda ser utilizado para vínculos síncronos como SDLC, HDLC, Frame Relay y X.25, además de la transmisión síncrona de datos.

Hay varias señales especificadas en el estándar RS232. El acomodo de éstas señales en las líneas de cableado se muestran en la tabla de pines a continuación:

9-pines	25-pines	
Escudo	1	Case Ground
1	8	DCD (Data Carrier Detect)
2	3	RX (Receive Data)
3	2	TX (Transmit Data)
4	20	DTR (Data Terminal Ready)
5	7	GND (Signal Ground)
6	6	DSR (Data Set Ready)
7	4	RTS (Request To Send)
8	5	CTS (Clear To Send)
9	22	RI (Ring Indicator)

Existe un adaptador serial estándar llamado MODEM Nulo que cruza la línea Tx con la línea Rx para permitir una conexión de dos computadoras. Esto es importante ya que si se cablea el puerto COM directamente a otro las líneas de transmisión Tx se conectaran juntas y ambos puertos intentarán transmitir en la misma línea, lo mismo pasaría con las líneas Rx. Un MODEM nulo cruza las líneas de control también.

La mayoría de los conectores cuentan con pequeños números al lado de los pines.

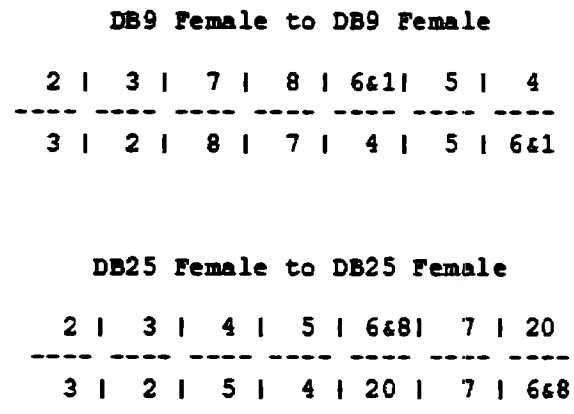


Figura 2.2.1 a. Pines del MODEM Nulo

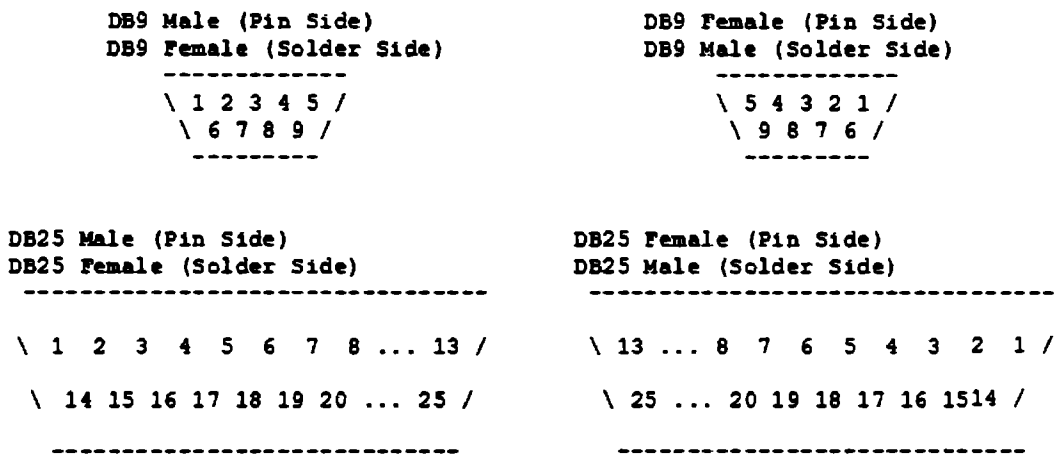


Figura 2.2.1 b. Diagrama de distribución física de los pines para los conectores DB9 y DB25

2.1.2.2 Librería MSCom para Visual C#

Visual Studio NET no cuenta con un "framework" de comunicación serial; sin embargo, se puede establecer la comunicación a través de un puerto serial utilizando el MSComm OCX que se incluye en Visual Studio 6. Debido a que MSComm tiene un control de licencia, se deben tener instalados por lo menos los componentes Active X.

Se debe agregar el control a una forma y no sólo inicializar el control directamente desde el código ya que esto requiere de información de estado especial de OCX y una licencia de desarrollo. Al dibujarlo en una forma, Visual Studio se encarga de esto.

Se agrega el MSComm COM/Control OCX a la forma:

1. Clic en el botón derecho del Mouse sobre la Caja de Herramientas
2. Elegir "Personalizar la Caja de Herramientas"
3. Seleccionar y agregar el "Control de Comunicación de Microsoft"
4. Dibujar el nuevo control en la forma (ícono de un teléfono)

Propiedades e información del evento:

com.CommPort
Establece o adquiere el Puerto serial de la computadora a utilizar.

com.PortOpen
Abre o cierra el Puerto serial.

com.RThreshold
Establece el número de caracteres a recibir antes de disparar el evento OnComm. Se establece en 0 para deshabilitar el llamado de un evento. Se establece en 1 para disparar OnComm cada que se recibe un carácter.

com.InputMode
Una de las constantes en MSCommLib.InputModeConstants para especificar si se envía/reciben strings de texto o arrays de byte. Por default trabaja con texto ya que es más sencillo de trabajar con el mismo aunque no es tan confiable como los bytes.

com.Settings
Se utiliza para preparar el Puerto en el formato "baud,p,d,s" donde baud = razón de baudio, p = paridad, d = # bits de datos, y s = # bits de paro. Ejemplo: com.Settings = "9600,n,8,1"

com.Handshaking
Otra constante de MSCommLib.HandshakeConstants para especificar el tipo de "handshaking": nulo, RTS/CTS hardware hs, y/o XOn/XOff software hs.

com.InBufferCount
Regresa el número de caracteres esperando en el buffer receptor.

com.Input
Regresa y remueve una corriente de datos del buffer receptor. Se usa para revisar la información que espera. Regresa un string si se encuentra en modo de texto o un byte array en modo binario/bytes.

com.Output
Escribe una corriente de datos a transmitir al buffer. Ejemplo: com.Output = "Hola" envía "Hola" por el Puerto serial.

com.CommEvent
Regresa una constante de tipo MSCommLib.CommEventConstants, MSCommLib.ErrorConstants, o MSCommLib.OnCommConstants que representa el error o evento más reciente que ocurrió.

com.NullDiscard
Si su valor es verdadero, el control serial ignora todos los caracteres 0x00 (nulos) que entren.

com.InputLen
El número de caracteres que la propiedad Input lee del buffer receptor. El establecer InputLen en 0 lee todo el contenido de un buffer receptor cuando com.Input se usa.

2.1.2.3 Envío y recepción de datos

La información serial fluye a través del puerto com byte a byte y debe ser correctamente procesada por el buffer.

Las señales que se utilizaron para cada robot durante el diseño del programa que le corresponde son:

Función de las señales en el puerto serial del Mitsubishi:

TxD: Transmitted Data

Esta línea es utilizada para transmitir datos desde el cronómetro de la señal emisora (DTE) al cronómetro de la señal de transmisión del transmisor (DCE). Es mantenida en estado de 1 lógico cuando nada se transmite. La terminal comenzará a transmitir cuando un 1 lógico esté presente en las siguientes líneas:
- Autorización de envío.
- Terminal de datos lista.
- Datos listos para enviar.
- Detección de portadora.

RxD: Received Data

Este circuito es utilizado para recibir datos desde el DCE al DTE. La terminal comenzará a transmitir cuando un 1 lógico esté presente en las siguientes líneas:

- Autorización de envío
- Terminal de datos lista.
- Datos listos para enviar.
- Detección de portadora.

El estándar especifica que los niveles de salida son -5 a -15 volts para el 1 lógico y +5 a +15 volts para el 0 lógico, mientras que los niveles de entrada son -3 a -15 volts para un 1 lógico y +3 a +15 volts para un 0 lógico.

Esto asegura que los bits puedan ser leídos correctamente aún con grandes distancias entre la DTE y la DCE, especificados como 16.5 metros o 50 pies, aún cuando estas señales soportan mayores distancias dependiendo de la calidad del cableado y el blindaje.

RTS: Request to Send

En esta línea el DTE envía una señal cuando está listo para recibir datos del DCE. El DCE revisa esta línea para conocer el estado del DTE y saber si puede enviar datos.

CTS: Clear to Send

En este circuito el DCE envía una señal cuando está listo para recibir datos del DTE.

DSR In

Esta señal es usada para indicar que la computadora personal es utilizada para transmitir y recibir datos.

GND

Tierra de señales para las líneas de datos y control

DTR Out

Esta señal se usa para indicar que la unidad motora está lista para transmitir y recibir datos.

El robot Mitsubishi se comunica a través de RxD y TxD. Por medio de CTS se verifica constantemente si el robot se encuentra realizando alguna operación, de forma que no se le manda la instrucción siguiente hasta que no haya concluido la actual.

Con esa misma señal se verifica la conexión correcta del cable serial y se puede comprobar si el control del robot se encuentra en modo encendido o apagado.

Diagrama de Conexión Serial Mitsubishi

Computadora	DB 9B		Mitsubishi	DB25A	
TxD	Pin	2 ⟶	3	Pin	RxD
RxD	Pin	3 ⟵	2	Pin	TxD
RTS	Pin	4 ⟶	5	Pin	CTS
CTS	Pin	5 ⟵	4	Pin	RTS
GND	Pin	7 ⟷	7, 8, 20, 6	Pin	GND
DSR	Pin	6 ⟵	20	Pin	DTR
DTR	Pin	20 ⟶	6	Pin	DSR

Función de las señales en el puerto serial del Puma

TxD: Transmitted Data

RxD: Received Data

GND: Tierra de señales para las líneas de datos y control

Baud Rate Jumper

Es una conexión que se realice para que el baud rate se mantenga en el robot y pueda procesar los datos de manera correcta.

El robot Puma se comunica a través de RxD y TxD. Cuando el robot termina de realizar alguna operación envía por medio de la señal RxD un "enter" y un punto, con esto se puede saber en que momento mandar la instrucción siguiente.

Diagrama de Conexión Serial PUMA

Computadora		DB9B		PUMA	DB25A
GND	Pin	1 ⟷	5	Pin	GND
TxD	Pin	3 ⟶	7	Pin	RxD
RxD	Pin	2 ⟵	6	Pin	TxD
		⟶	2,9	Pin	Baud rate jumper

2.1.2.4 Protocolos de Comunicación

Los datos llegan a los puertos de forma asíncrona; es decir, su llegada es imprevisible. Esto sugiere que el dato que llega tiene que procesarse inmediatamente, puesto que pueden llegar otros datos. De esta tarea se encarga el hardware del PC, de forma que cuando detecta la llegada de un dato, interrumpe el flujo del proceso para ceder el control a la rutina de Windows en lugar de una rutina de la aplicación. Esto es así por dos razones:

- Windows debe mantener el control de la multitarea. Si la llegada de un dato hiciera que se transfiera el control del procesador a la aplicación, Windows perdería su habilidad para gestionar la multitarea. Esto quiere decir que Windows tiene que estar entre la aplicación y el hardware.
- Windows no puede dirigir el dato recibido directamente a la aplicación. La razón es que los datos que se reciben en el puerto de comunicación no llegan con la identidad de la aplicación que los tiene que recibir. Por lo tanto, Windows tiene que guardar en un buffer los datos que llegan para una aplicación. La aplicación debe haberle pedido a Windows la propiedad del puerto utilizando la función correspondiente. Esta misma función establece dos buffers, uno para la entrada y otro para la salida.

Cuando una aplicación solicita a Windows la propiedad de un puerto, Windows sólo se lo dará si ninguna otra aplicación lo tiene. Por el mismo motivo, mientras una aplicación tiene el control de un puerto Windows se lo prohíbe a las demás aplicaciones que lo soliciten. Cuando esta aplicación finalice la operación de E/S con un puerto, debe dejar el control del mismo para que otras aplicaciones puedan utilizarlo, lo cual requiere llamar a la función que permita cerrar el puerto.

Para establecer una comunicación, de forma general se deben seguir los siguientes pasos:

1. Abrir el puerto de comunicaciones (COM1, COM2, etc.)
2. Establecer la máscara de comunicaciones para especificar los eventos que serán atendidos.
3. Definir el tamaño de buffer de las colas de entrada y salida.
4. Construir una estructura de tipo DCB que especifique la configuración del puerto (DCB- Device Control Block). Esta estructura contiene, entre otros, los datos velocidad de transmisión, paridad, bit por carácter y bits de parada.
5. Configurar el puerto con los datos de la estructura de tipo DCB.
6. Envía datos al puerto de comunicaciones.
7. Recibir datos por el puerto de comunicaciones.
8. Verificar errores.
9. Cerrar el puerto cuando la comunicación haya finalizado.

Fijación de Parámetros de la RS232C en la consola del Mitsubishi

Baud Rate
Se fija con el SW3 localizado en la puerta lateral de la unidad motora a través del siguiente rango:

Dip SW3 Bit No.	Baud rate factor		
	X1	X16	X64
1	1200	75	-
2	2400	150	-
3	4800	300	75
4	9600	600	150
5	-	1200	300
6	-	2400	600
7	-	4800	1200
8	-	9600	2400

Tabla 1. Fijación del Rango del Baud Rate

Fijación del formato en transmisión

		Null	X1	X16	X64
1	Factor de Baud Rate	0	0	1	1
2		0	1	0	1
		5 bits	6 bits	7 bits	8 bits
3	Tamaño de carácter	0	0	1	1
4		0	1	0	1
		Par	Impar		
5	Paridad	1	0		
6		1	0		
		Null	1bit	1-1/2 bit	2 bit
7	Bits de parada	0	0	1	1
8		0	1	0	1

Tabla 2. Configuración de parámetros en SW2

Parámetros con los que se trabaja en el Mitsubishi
Paridad: par
Tamaño de carácter: 7
Baud Rate: 9600
Bits de parada: 2

Parámetros con los que se trabaja en el PUMA
Paridad: Ninguna
Tamaño de carácter: 8
Baud Rate: 9600
Bits de parada: 1

2.1.2.5 Información general de la clase Comunicación

Microsoft C# incluye un control llamado "*Microsoft Communications Control*" el cual permite poder establecer una comunicación vía puerto serial entre máquinas este control se utiliza en la clase comunicación; la cual está basada en el estándar RS232. Para poder utilizar este control en nuestra aplicación se añadió el control ActiveX mscomm32.ocx; este control cuenta con el evento **OnComm** el cual permite manipular la información tanto en el envió y recepción.

Creando un objeto de OnComm se puede acceder y con esto modificar diversas propiedades como son:

CommPort el cual permite modificar o acceder a cualquier puerto serial en este se especifica el puerto COM que se desee utilizar.

El evento OnComm siempre se genera una vez que cambie el valor de la propiedad CommEvent el cual indicará que se ha producido un evento o error en la comunicación.

Esta propiedad CommEvent contiene la constante numérica que corresponde al evento que se ha generado; estas constantes se muestran a continuación:

Constante	Valor	Descripción
comEvSend	1	Evento "enviar datos".
comEvReceive	2	Evento "recibir datos".
comEvCTS	3	Cambio en la línea (CTS) "preparado para enviar".
comEvDSR	4	Cambio en la línea "equipo de datos preparado". (DSR)
comEvCD	5	Cambio en la línea (CD) "detección de portadora".
comEvRing	6	Detección de llamada
comEvEOF	7	Fin de fichero

Tabla 3. Constante de eventos

Por otro lado los datos que se envían de la computadora a otro dispositivo se hace a través de la propiedad **Input**; la cual está determinada por el valor de la propiedad InputMode la cual se muestra a continuación:

Constante	Valor	Descripción
comInputModeText	0	Los datos se recuperan como texto mediante la propiedad Input
comInputModeBinay	1	Los datos se recuperan como binarios a través de la propiedad Input

Tabla 4. Constantes de propiedad InputMode

Asimismo, esta librería contiene protocolos de comunicación entre dos dispositivos; estos servirán para poder recibir y transmitir datos vía puerto serial entre la computadora personal y otro dispositivo; estos se muestran a continuación:

Constante	Valor	Descripción
Compone	0	Sin protocolo.
comXonXoff	1	Protocolo XON/XOFF.
comRTS	2	Protocolo RTS/CTS (petición de envío/preparado para enviar).
comRTSXOnXOff	3	Ambos protocolos (RTS y XON/XOFF).

Tabla 5. Constantes de protocolos

Las propiedades RTheshold y SThreshold indican, respectivamente, los caracteres que puede admitir el *buffer* de transmisión y los caracteres que se van a recibir respectivamente, antes de que el control genere el evento OnComm. Si SThreshold es cero no se generará el evento OnComm en la transmisión y si lo es RThreshold, no se generará el evento OnComm en la recepción; es decir se desactiva la interpretación de los eventos "*comEvReceive*" y "*comEvSend*" respectivamente.

La propiedad de InputLen indica el número de caracteres que se leerán cada vez que se utilice la propiedad Input para envío de datos; las propiedades "*InBufferSize*" y "*OutBufferSize*" corresponden al tamaño del Buffer de entrada y salida respectivamente; así mismo para borrar el contenido de estos Buffer se utilizan las propiedades "*InBufferCount*" y "*OutBufferCount*".

Clase "Comunicación" del robot "Puma"

La comunicación con este robot es a través de dos hilos (TxD y RxD) sin ningún protocolo de comunicación; tanto la transmisión como envió de datos se da carácter por carácter mediante código ASCII la velocidad de información es de 9600 bits por segundo (bps); con ocho bits por carácter y un bit de parada.

Cada instrucción que se envía es procesada y ejecutada por el robot una vez que se envía un retorno de carro; el robot al recibir cualquier instrucción transmite a la computadora el mismo código que se envió más una respuesta si es que la instrucción lo solicita como puede ser un mensaje de error o datos como puntos guardados en la memoria; el envío de datos por parte del robot a la computadora termina con un retorno de carro y un punto.

Para cumplir con estos parámetros en la recepción y transmisión de datos, se colocó como protocolo de comunicación "ninguno" ya que solo se maneja una comunicación de dos hilos por lo que no es necesario sincronizar las señales del RTS, DTR y CTS.

La propiedad SThreshold está en cero pues no se quiere que se genere interrupción al enviar datos, RThreshold está en uno para que se genere la interrupción al recibir un solo carácter del robot y el InputLen está en uno.

La única interrupción que se maneja en la comunicación con el robot es la correspondiente a "recibir datos" (comEvReceive) que tiene como constante numérica dos; con esta se manipulan los caracteres recibidos para realizar las acciones como subir puntos, bajar puntos y saber que comando se le ha enviado al robot como puede ser mover a un punto, abrir o cerrar tenaza etc.

El funcionamiento de esta interrupción es de la siguiente manera: a través de dos variables "s" y "s2" se van almacenando los caracteres que el robot va enviando mientras que en otra variable "s1" se almacenan toda la cadena de caracteres; cuando se tiene almacenado el carácter "retorno de carro" y "punto" significa que el robot acabó de ejecutar una acción como moverse a un punto o de enviar datos en este punto la variable "s1" tiene toda la cadena de caracteres que envió el robot; luego toda esta cadena se divide por cada salto de línea que se encuentre utilizando la función "*Split*" creando de esta forma un arreglo con diversas cadenas de caracteres; en la localidad cero de esta cadena se encontrará la instrucción que se le mandó al robot y en las localidades subsecuentes la respuesta del robot como pueden ser las coordenadas de los puntos, mensajes de error en el modo de control, mensajes de error de conexión, etc.

De esta forma se compara las localidades de este arreglo con la instrucción correspondiente sabiendo que datos debe recibir el robot y con base en estos poder manipularlos; entre estos datos se encuentran:

- "*COMP mode disable*" que significa que el control del robot no está en el modo adecuado para que el robot realice una determinada instrucción.

- "*llist*" que significa que la instrucción que se le envió al robot es la localización de los puntos que se tienen guardados en memoria; esta instrucción permite guardar los puntos en el programa lo cual se hace a través de la función Puntos().

- "*do move pnt br*" que significa que se mandó la instrucción de moverse a un punto, "*do ready*" que significa que se mandó la instrucción de mandar a inicio al robot, "*do closet*" o "*do openi*" que significa que se mandó a cerrar o abrir la tenaza del robot y finalmente "*do speed br always*" que significa que se cambia la velocidad de movimiento de robot.

Una vez que el robot terminó su trabajo, las variables son borradas para una nueva recepción de datos.

Las variables globales de esta clase son:

- Bool ejecutando: indica dentro de una función si una instrucción sigue en ejecución o ya se terminó, se inicializa en falsa y cambia a verdadera cada vez que se llama a la función EsperaEjecutar() la cual indica que se ha finalizado la ejecución de una determinada instrucción; asimismo esta variable se vuelve a colocar en falso cuando se reconoce una de las instrucciones dentro del evento de la interrupción.

- Bool flag: indica si ya se han bajado los puntos de la memoria del robot, se inicializa en falso y cambia a verdadero cuando se llama al método BajarPuntos() y se vuelve a colocar en falso cuando se reconoce la instrucción "*llist*" dentro de la interrupción.

- Bool comp: indica si el control del robot está en el modo correcto o de lo contrario se manda un mensaje de texto dando la instrucción de cambiar el modo del control a "COMP", la variable se inicializa en falso, cambia a verdadero cada vez que se mande la instrucción "*do move xploz*" y si la respuesta del robot es "COMP mode disable" se vuelve a colocar en falso, en otro caso, se coloca en verdadero indicando que el control se encuentra en el modo correcto.

- Bool conectado: indica si el puerto serial está bien conectado; esta se inicializa en falso, cambia a verdadero cada vez que se conecta correctamente vía puerto serial y esto se identifica mandando la instrucción "*here*" en el método Conectar(), si hay respuesta del robot significa que está bien conectado en otro caso se vuelve a pasar la variable a falso y se manda un mensaje de texto señalando que el cable serial no está bien conectado.

- String br: sirve para convertir datos enteros a string para que los números de los puntos puedan ser manipulados fácilmente; esta variable se usa en el evento de la interrupción, los métodos "MoverPunto()" y "Velocidad()" .

- String s: sirve para almacenamiento de un carácter de recepción y compararlo con s2 para ver si se enviaron un punto y retorno de carro se utiliza en el evento "OnComm".

- String s2 sirve para almacenamiento de un carácter de recepción se utiliza en el evento "OnComm".

- String s1: sirve para almacenar todos los caracteres que transmite el robot se utiliza en el evento "OnComm"

- String textopuntos: en esta variable se guardan las coordenadas de todos los puntos que se tengan guardados en la memoria del robot; esto se hace al mandar la instrucción de bajar puntos.

- String [] nombres, es un arreglo que en cada entrada guarda los nombres de cada punto que se tenga guardado.

- String [] coordenadas, es un arreglo que en cada entrada guarda las coordenadas de cada punto que se tenga guardado, estas coordenadas corresponden al nombre guardado en el arreglo anterior.

Los Métodos que contiene la clase comunicación del robot "PUMA" son los siguientes, todas las funciones internas que se utilizan se mandan añadiendo un retorno de carro tal y como se mencionó en los párrafos anteriores.

- Conectar( ): Para establecer las propiedad de conexión por puerto serial, se señala el puerto COM a utilizar, tamaño de pilas de entrada y salida de datos, se inicializan las pilas de transmisión y recepción, se establece la velocidad de transmisión de información a 9600 bits por segundo, ocho bits por carácter, un bit de parada.

- Desconectar( ): Cierra la conexión del puerto serial cuando se terminó de ejecutar una determinada instrucción.

- Iniciar( ): Calibra al robot mediante la instrucción "*cal*" y manda al robot a posición de Inicio a través de la instrucción "*do ready*". Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- Home ( ): Mover robot a posición de Inicio mediante la función "*do ready*". Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- MoverPunto (string x): recibe un valor entre cero y doscientos correspondiente al punto al que se desea mover al robot. La instrucción que se envía es "*do move pnt x*". Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- Velocidad(string x): cambiar a la velocidad que se recibe como palabra de datos; esta cadena de datos que recibe es un número entre 1 y 600 que indica la velocidad mínima y máxima respectivamente; en esta se manda la instrucción "*do speed x always*". Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- AbreGriper( ): Se envía la instrucción "*do openi*" para que el robot abra la tenaza. Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- CierraGriper( ): Se envía la instrucción "*do closei*" para que el robot cierre la tenaza. Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- EsperaEjecutar( ): Método que indica que el robot ha terminado de ejecutar una determinada instrucción. Utiliza las funciones "*ThreadSleep(x)*" y

"*Application.DoEvents()*" que da un tiempo para realizar otros eventos en forma simultánea mientras la bandera de Ejecutando se mantiene en verdadero.

- BajarPuntos( ): Instrucción que indica al robot de la posición de los puntos guardados en memoria. Utiliza la función Conectar() para establecer conexión con el puerto serial, se inicializa la variable "textopuntos" que es donde se almacenarán los puntos, se coloca la variable flan en verdadero y luego en la interrupción se llama al método Puntos () el cual se encarga de tomar los puntos guardados en la memoria del robot para que manipularlos.

- Puntos(string [] Datos, int i): esta función recibe un arreglo con todos los puntos que se tienen guardados en memoria del robot y una variable entera que indica el tamaño del arreglo; esta función se encarga de filtrar la información que manda el robot y solo quedarse con el nombre de los puntos y sus respectivas coordenadas.

- BorrarPuntos( ): Borra los puntos guardados en memoria del robot. Llama a la función Conectar( ) que establece la comunicación por puerto serial, se manda la instrucción "*zero*" indicando que se borren todos los puntos del uno al doscientos. Al final se llama a la función Desconectar() para cortar la conexión con el puerto serial e indicar que se ha acabado de borrar los puntos.

- SubirPuntos( ): Sube los puntos guardados en el programa a la memoria del robot. Esta función llama al método Conectar(), verifica que la variable donde están guardados los puntos no está vacía y divide a esta cadena por cada retorno de carro que se dio en un arreglo de caracteres posteriormente se manda a llamar a los métodos "AdquiereNombre()" y "AdquiereCoordenada()" para obtener los nombres y coordenadas de los puntos y subirlos correctamente a la memoria del robot a través de la función "SubePuntos()" ; cuando se finaliza el proceso de subir puntos se llama la método Desconectar( ) que corta la conexión con el puerto serial.

- SubePuntos( ): este método manda al robot las instrucciones "*do here pnt*" con lo que se crea el nombre del punto; este se adquiere del arreglo que regresa la función AdquiereNombre(), posteriormente se manda la instrucción "*here pnt*" para poder modificar las coordenadas del punto; y finalmente se envían las coordenadas que se adquieren del arreglo que regresa la función AdquiereCoordenadas().

- AdquiereCoordenadas(string [] Datos2, int longitud) este método recibe una cadena de caracteres que contiene tanto los nombres y coordenadas de los puntos que se guardaron en la memoria del robot sin estar filtrados; es decir con símbolos que se deben quitar tales como espacios, comas y puntos; también se recibe el tamaño del arreglo; esta función se encarga de quitar todos los espacios entre las coordendas de los puntos y anexar solo una coma la nueva cadena de caracteres se guarda en un nuevo arreglo que es el que se regresa.

- AdquiereNombre(string [] Datos2, int longitud) se encarga de obtener solo los nombres con los que se identifica cada punto y eliminar los espacios que se tengan; esta función regresa un arreglo con los nombres de los puntos.

- CrearPunto(string valor): Crea un punto en una determinada posición con el nombre que recibe "valor"; para esto se envía al robot la instrucción "*do here pnt valor*".

Clase "Comunicación" del robot "Mitsubishi"

La comunicación con este robot es a través de TxD, RxD (transmisión y recepción de datos), RTS, CTS (cambiar a modo de transmisión y listo para transmitir), DSR y DTR (listo para comunicar y Terminal de datos lista) se utiliza el protocolo de comunicación RTS/CTS que establece la sincronización de petición de envío de datos y datos preparados para enviar; tanto la transmisión como envió de datos se da carácter por carácter mediante código ASCII la velocidad de información es de 9600 bits por segundo (bps); con siete bits por carácter y dos bits de parada.

Cada instrucción que se envía es procesada y ejecutada por el robot una vez que se envía un retorno de carro; el robot únicamente transmite datos a la computadora si la instrucción lo requiere como es el caso de "bajar puntos" añadiendo un retorno de carro cuando está acción haya concluido. Por otro lado cuando se le envía al robot una instrucción de ejecutar una determinada acción como por ejemplo cerrar o abrir tenaza, el robot no envía ningún carácter que indique que ha terminado de ejecutar una instrucción.

Las propiedades de SThreshold está en uno para saber si se están enviando datos al robot, RThreshold está en uno para que se genere la interrupción al recibir un solo carácter del robot y el InputLen está en uno.

En los eventos generados por OnComm se utilizan tres interrupciones: la de "recibir datos" (comEvReceive) que tiene como constante numérica dos; con esta se pueden manipular los caracteres recibidos para realizar las acciones como subir puntos, bajar puntos y saber que comando se le ha enviado al robot. En esta, se van almacenando a través de una variable los caracteres que el robot va enviando; cuando se tiene almacenado el carácter "retorno de carro" significa que el robot acabó de enviar datos (caracteres). Este evento se utiliza en el método BajaPunto ( ) para controlar el envío de la instrucción "pr más el número de punto"; aquí se va almacenando los caracteres que envía el robot en la variable "s", y cuando el robot envía un retorno de carro se compara con la cadena de caracteres "0,0,0,0,0" que indica que no hay localidad guardada en ese punto; en el caso de que la cadena de caracteres recibida sea diferente la almacena en la variable "textopuntos" la cual desplegará los puntos contenidos en un cuadro de texto visible para el usuario. Una vez que el robot terminó su trabajo, las variables son borradas para permitir una nueva recepción de información.

Las otras dos interrupciones utilizadas son la uno y tres correspondientes a "Envío de datos" y "Cambio de línea preparado para enviar" respectivamente; estas interrupciones se utilizan para saber cuando el robot ha dejado de ejecutar una instrucción que no implique el envío de datos tales como mover a un punto, cambio de velocidad entre otras; la interrupción número tres el robot siempre envía por RTS *"Request to Send"* pulsos cuadrados de período 75ms; y cuando ejecuta una instrucción la señal que manda el robot se pone en baja durante el tiempo que se tarde en ejecutarla; de esta forma y con ayuda de una bandera "ejecutando" se permite saber que se salió de la interrupción y con esto se puede determinar cuando el robot ha ejecutado la instrucción estableciendo que se le puede enviar una nueva instrucción a ejecutar; esta bandera se inicializa como falsa en las variables globales la cual siempre pasa a verdadero cuando se produce la interrupción uno y con esta se sabe cuando empieza el envío de datos al robot y cuando se ha terminado de mandar la instrucción completa; la bandera ejecutando pasa a falsa una vez que el robot haya terminado de ejecutar como es el caso de moverse, abrir tenaza, etc.

Para saber que una instrucción terminó de ejecutarse al terminar de mandar la instrucción se inicializa una variable entera llamada "a" en 30 que indica un tiempo que esperará más la computadora entre cada ejecución finalizada de instrucción; luego en la interrupción tres esta variable se va decrementando hasta cero indicando este tiempo extra que se da para asegurar que se ha terminado de ejecutar la instrucción y con base en esta pasar a ejecutar otra nueva instrucción, permitiendo saber cuando finaliza el envío de información al robot y cuando este empieza a ejecutar la instrucción.

Las variables globales de esta clase son:

- Bool ejecutando: bandera que indica si se sigue enviando datos al robot. Se inicializa en falsa como variable global, cambia a verdadero cada vez que se entra a la interrupción uno (envío de datos) y se vuelve a colocar en falso en la interrupción tres (preparado para enviar) una vez que el contador "a" es cero; es decir que indica que se terminó de ejecutar la instrucción.

- Bool flag: Indica si se ha terminado de bajar todos los puntos guardados en la memoria del robot. Se inicializa en falsa como variable global, cambia a verdadero en el método BajarPunto( ) y se vuelve a colocar en falso en la interrupción dos una vez que se haya terminado de bajar todos los puntos.

- Bool controlon: bandera que indica si el control del robot está en el modo correcto. Se inicializa en falsa como variable global; pasa a verdadero una vez que se estableció una buena conexión vía puerto serial y regresa a falsa una vez que se haya acabado de utilizar la conexión.

- String cont: variable que sirve para leer todos los puntos que se tienen en la memoria del robot es un contador que se inicializa en uno y llega la cuenta hasta 200 que es el último punto donde se pueden guardar datos.

- String s: variable que sirve para almacenamiento de un carácter de recepción.

Los Métodos que contiene la clase comunicación del robot "Mitsubishi" son los siguientes; todas las funciones internas que se utilizan se mandan añadiendo un retorno de carro tal y como se mencionó en los párrafos anteriores

- Conectar( ): Para establecer las propiedad de conexión por puerto serial; se establece el puerto COM que se utilizará, el tamaño de las pilas de recepción y transmisión, se inicializan estas pilas, se establece el "Número de caracteres que se leerán al entrar a la interrupción", "Modo de recepción de datos" y abrir puerto; asimismo se utilizan las funciones *ApplicationDoEvents( )* que permite realizar otros eventos en forma simultanea con el programa.

- Desconectar( ): Cierra el puerto de conexión cuando se termina de ejecutar una instrucción.

- Iniciar( ): se manda a posición de Inicio a través de la función "*nt*". Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- Home ( ): Mover robot a posición de Inicio mediante la función "*nt*". Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- MoverPunto (string x, bool cerrado): recibe un valor entre cero y doscientos correspondiente al punto al que se desea mover al robot y un valor booleano que indica si se quiere que se mueva el robot con la tenaza abierta o cerrada. La instrucción que se envía es "*mo x,C*" o "*mo x,O*" dependiendo si se quiere mover al robot con la tenaza cerrada o abierta Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- Velocidad(string x): cambiar a la velocidad que se recibe como palabra de datos; esta cadena de datos que recibe es un número entre 1 y 9 que indica la velocidad mínima y máxima respectivamente; en esta se manda la instrucción "*sp x*" Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- AbreGriper( ): Se envía la instrucción "*go*" para que el robot abra la tenaza. Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- CierraGriper( ): Se envía la instrucción "gc" para que el robot cierre la tenaza. Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- EsperaEjecutar( ): Método que indica que el robot ha terminado de ejecutar una determinada instrucción. Utiliza las funciones "*ThreadSleep(x)*" y

"*Application.DoEvents()*" que da un tiempo para realizar otros eventos en forma simultánea mientras la bandera de ejecutando se mantiene en verdadero.

- BajarPuntos( ): Instrucción que indica al robot de la posición de los puntos guardados en memoria. Utiliza la función Conectar() para establecer conexión con el puerto serial, se inicializa la variable "textopuntos" que es donde se almacenarán los puntos se inicializa el contador "cont" en uno y se llama la función BajarPunto() que permitirá bajar punto de una pila de datos.

- BajarPunto( ): esta función pone la bandera "flag" en verdadero indicando que se esta leyendo una localidad de la pila de puntos esto se hace hasta que se lee el punto 200 que es la última localidad donde se puede guardar puntos.

- BorrarPuntos( ): Borra los puntos guardados en memoria del robot. Llama a la función Conectar( ) que establece la comunicación por puerto serial, se manda la instrucción "*pc 1, 200*" indicando que se borren los puntos del uno al doscientos. Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- SubirPuntos( ): Sube los puntos guardados en el programa a la memoria del robot. Esta función utiliza la variable "textopuntos" donde se encuentran guardados los puntos que se crearon; esta función debe dividir todo la cadena en subcadenas que contengan las coordenadas de cada punto, esto se hace a través de la función "*Split*" que separa la cadena por cada salto de línea que se encuentre. Asimismo a cada subcadena se le anexar la instrucción "*pd*" a través de la función "Anexapd( )" que indica que esa cadena contiene las coordenadas de un punto el cual se desea subir a la memoria del robot. Al final se llama a la función EsperaEjecutar que indica que el robot ha terminado de ejecutar la instrucción.

- Anexapd(string sr1): este método recibe una cadena de caracteres que contiene a todos los puntos que se crearon; este se encarga de separar la cadena en subcadenas de caracteres y anexarles al inicio la instrucción "*pd*" indicando que la cadena contiene las coordinas del punto que se desea guardar en la memoria del robot. El método regresa una cadena de caracteres que contiene a todos los puntos con la instrucción "pd" al inicio.

2.2 Diagrama Electroneumático

El diagrama electro-neumático que se tiene en cada una de las estaciones de la celda es el siguiente:

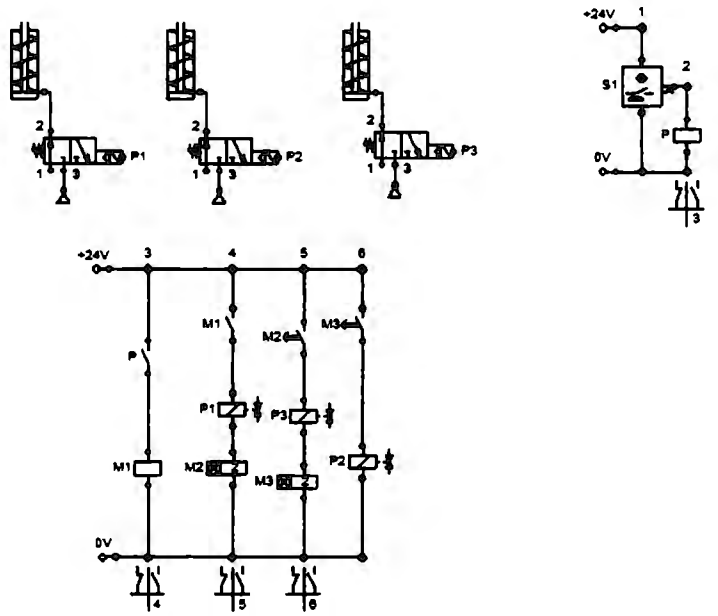


Figura 2.2. Diagrama Electroneumatico de Estación 1, 2 y 3

2.2.1 Replanteamiento de conexiones

Se pensó en una nueva forma de conectar los elementos del sistema de forma que se redujera el cable al máximo y que se implementara Ethernet. Se obtuvo un nuevo diseño más sencillo y fácil de entender.

El ASRS aún no ha sido conectado dentro de la comunicación Ethernet que se estableció pero puede ser anexado en un futuro.

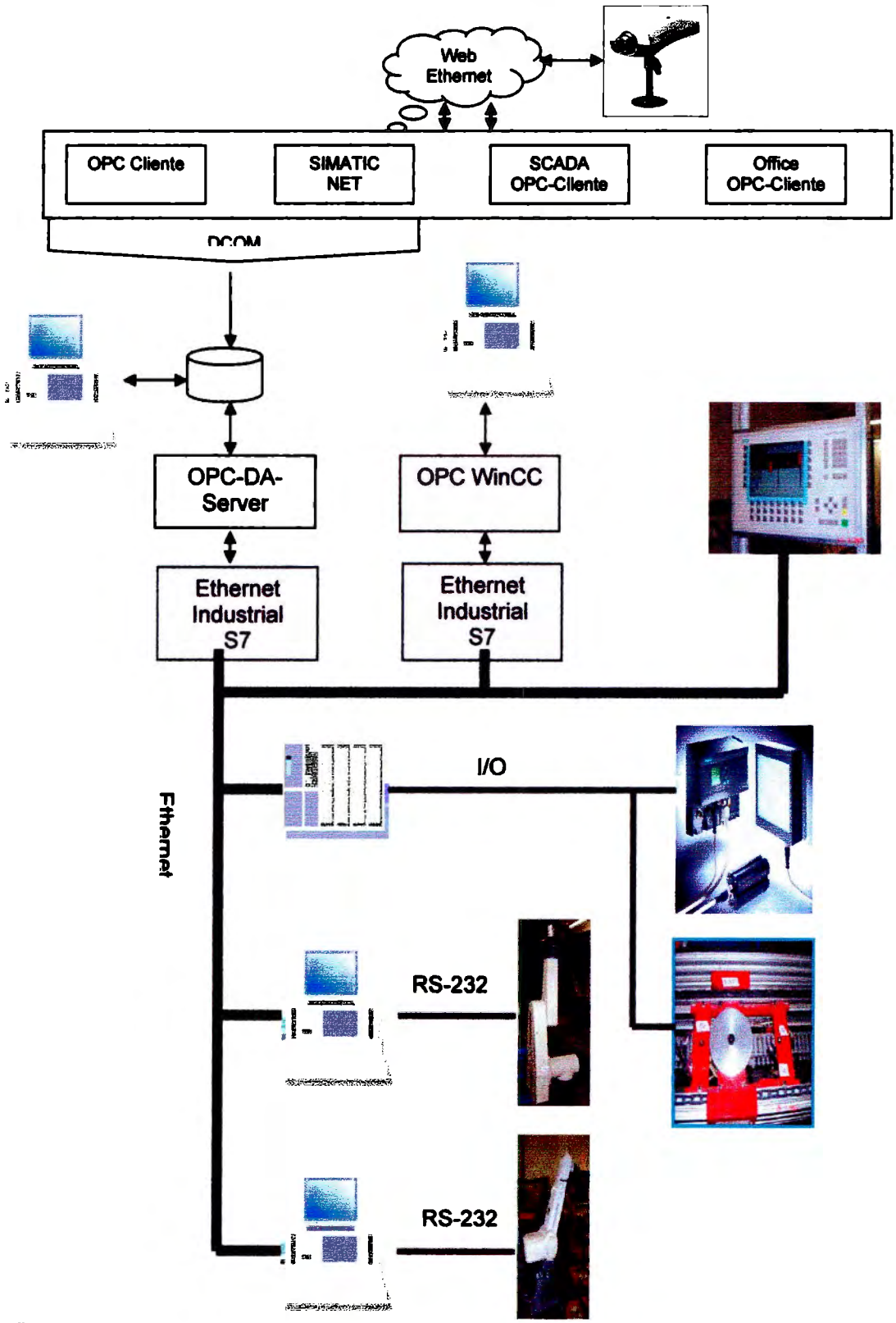


Figura 2.2.1 b. Diagrama de conexiones de la Celda de Manufactura actual

La celda cuenta con 4 estaciones, en la estación 1 se encuentra el robot Puma, en la estación 2 el robot Neptuno, el cual por el momento no está funcionando, en la estación 3 se encuentra el robot Mitsubishi y finalmente, en la estación 4 el sistema de visión.

Las estaciones 1,2 y 3 cuentan con un sensor y tres actuadores. La estación 4 cuenta con un motor, 3 actuadores y 3 sensores. También se cuenta con un sensor de cambio de banda y un actuador para pasar a la otra banda.

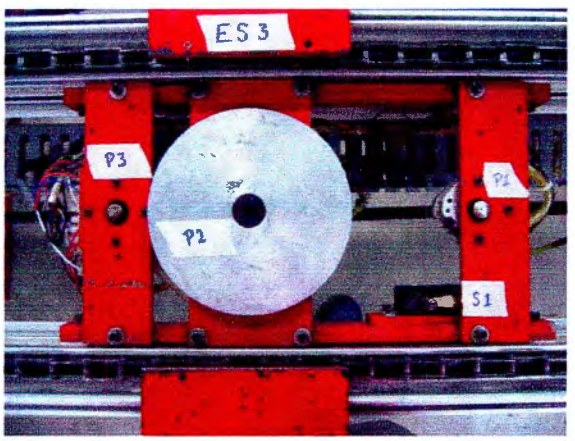


Figura 2.2.1 c. Componentes de la Estación 1 de la Celda

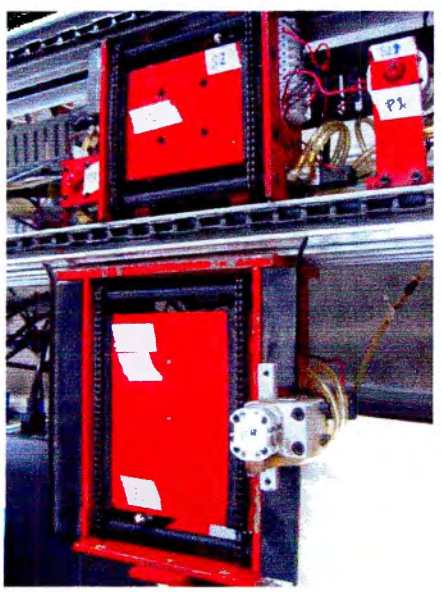
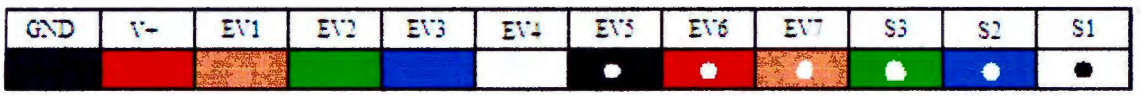


Figura 2.2.1 d. Componentes de la Estación 4 de la Celda

En cuanto al cableado de sensores y actuadores hacia el PLC, existen 3 cables principales, cada uno respeta la distribución siguiente:



EV: electroválvula
S: sensor
GND: tierra
V: voltaje

En el cable 1 se encuentra la estación 1, en el cable dos están la estación 2, la 3 y el sensor y actuador de cambio de sentido, en el cable 3 está la estación 4, manejándose la activación del motor y su cambio de sentido como dos electroválvulas.

Estos cables van conectados a las entradas y salidas del PLC de forma consecutiva.

2.2.2 Definición de equipo

Adicional a los elementos propios de la Celda de Manufactura, se tiene:

- Un PLC Simatic S7300 de espacio de direcciones de entrada/salida de 8192 bytes, una memoria principal de 1400 KB y una tarjeta de memoria de 8MB.
- Un CP 343 Simatic NET, tarjeta de red Ethernet Industrial
- Un hub para comunicación con Ethernet 10 base T
- Un Panel Siemens
- Tres computadoras

Figura 2.2.2 a. PLC y módulos de la Celda de Manufactura actual

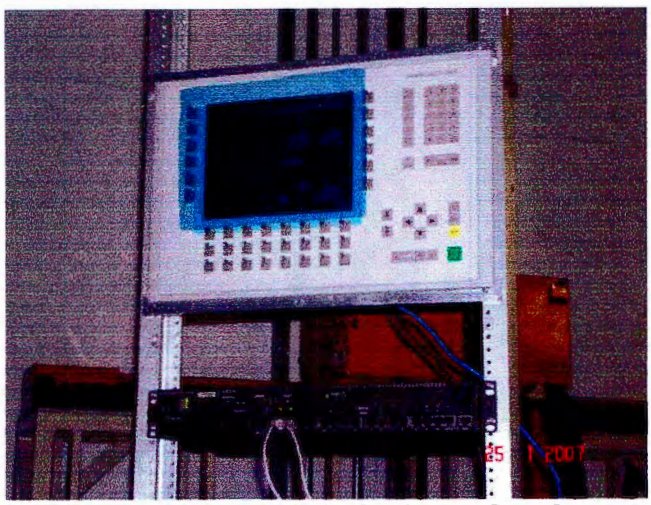


Figura 2.2.2 b. Panel y Hub de la Celda de Manufactura actual

2.3 Comunicación OPC

La tendencia hacia la reducción de costos en la ingeniería de automatización ha significado un acercamiento de los controladores lógicos programables (PLCs), drivers, transductores y sensores a los procesos técnicos actuales. Esto ha resultado en una mayor distribución de estos dispositivos de campo por medio de un bus de campo como un medio común de comunicación para el intercambio de información.

La base de todos los mecanismos OLE es el COM (Component Object Model), el cual define un estándar que permite a los objetos ser definidos como unidades aparte en Windows y acceder estas unidades más allá de los límites de un proceso.

Los objetos pueden entenderse como extensiones de un sistema operativo. No son dependientes de lenguajes programables y están disponibles en principio a todas las aplicaciones. El modelo de componente COM es una arquitectura cliente-servidor.

Podemos definir al servidor como el objeto que provee servicios y al cliente como la aplicación que usa los servicios de un objeto.

Para permitir a los objetos ser accesados fuera de una computadora local se creo una extensión del COM conocida como DCOM (Distributed COM) en la cual los objetos utilizados por una aplicación son distribuidos dentro de una red.

La especificación OPC agrupa las interfaces y sus métodos en tres clases jerárquicas. Esta estructura es conocida como el modelo de clases. Tanto la interfaz de automatización como la de personalización están basadas en este modelo.

Clase "Model"
Esta clase consiste en tres clases de objeto:

- ✓ OPC server
- ✓ OPC group
- ✓ OPC item

En un servidor OPC n grupos OPC pueden ser inicializados y en 1 grupo OPC n items OPC pueden ser inicializados.

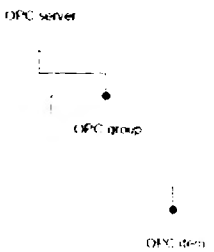


Figura 2.4 a. Modelo de clases de OPC

Clase "OPC Server"

La clase más alta es la "OPC Server". Ésta tiene varios atributos que contienen información acerca del estatus, la versión, etc., de un objeto de "OPC Server". La clase "OPC Server" también tiene métodos con los que el cliente administra los objetos de esta clase directamente usando mecanismos COM. Los otros objetos son creados por métodos de OPC correspondientes.

Clase "OPC Group"

Ésta clase administra las variables de proceso individuales, los "items OPC". Usando estos objetos del grupo un cliente puede formar unidades de items IOC semánticamente significativos y ejecutar operaciones con ellos. Todas las variables de proceso de una máquina A podrían, por ejemplo, ser colocadas en un grupo y las variables de proceso de la máquina B en un segundo grupo. Por otra parte, todas las variables de proceso de una página de monitoreo de un sistema de interfaz de operador pueden estar juntas en un grupo. Cuando la página es desplegada, el grupo puede ser activada.

Clase "OPC Item"

Un objeto de item representa una unión hacia una variable de proceso, por ejemplo al módulo de entrada de un controlador programable. Una variable de proceso es un dato de las entradas/salidas del proceso que puede ser escrito y/o leído, por ejemplo la temperatura de un tanque.

Un objeto de la case "Server" puede ser considerado el contenedor para los objetos de la clase "group". Un objeto de la clase "group" puede ser considerado un contenedor de objetos de la clase "item".

El servidor OPC de SIMATIC NET puede acceder los datos del SIMATIC S7 vía una conexión S7 (TCP/IP). Con ProTool como cliente OPC uno puede leer estos datos del servidor de OPC de SIMATIC NET vía una conexión OPC.

2.3.1 Envío de señales robot-PLC

2.3.1.1 Comunicación servidor-cliente

Cliente OPC

El cliente OPC puede realizar las siguientes acciones con el servidor:

- Lectura/Escritura síncrona
- Lectura/Escritura asíncrona
- Monitoreo de variables (reporte de cambio de valores)

Servidor OPC

El servidor OPC es la comunicación central entre el cliente OPC y el controlador respectivo. Vía mecanismos COM/DCOM, éste provee interfaces estandarizadas hacia el cliente OPC que permiten a cada aplicación COM acceder a variables de cualquier controlador.

El servidor OPC se encuentra conectado al controlador respectivo vía la implementación de los protocolos de comunicación.

Conexión entre el servidor OPC y el controlador

Los valores de las variables son leídos vía las conexiones configuradas en el servidor OPC usando los bloques de protocolo respectivos. Estas conexiones se configuran utilizando el NetPro tool.

Leyendo variables del controlador

La lectura y actualización de variables puede realizarse de dos formas:

Servicio	Descripción
Servicio de variables	Una o varias variables de proceso se especifican mediante direccionado absoluto o simbólico. Debido a esto las variables pueden ser monitoreadas o modificadas(se pide su valor o estatus) cíclicamente del servidor OPC.
Servicio de bloques	Si las variables a ser monitoreadas son transferidas al programa del servidor OPC controlado mediante bloques de datos, entonces esto se llama servicio de bloques. Debido a esto, no se direccionan las variables de proceso en si, sino las áreas de datos como un bloque de datos. Un monitoreo cíclico del buffer de recepción por el servidor OPC solo tiene sentido para recibir items

Tabla 6. Lectura de variables del controlador.

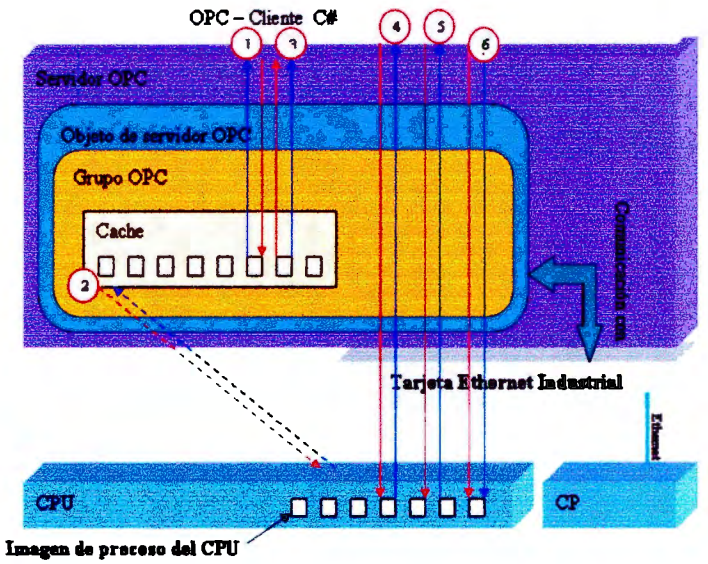


Figura 2.3.1.1 a. Explicación de la relación entre servidor y cliente OPC

La figura 2.3.1.1 a. muestra la dirección de llamado (flechas rojas) y el flujo de datos resultante (flechas azules).

Se intercambia información entre el servidor OPC y el controlador utilizando una tarjeta de Ethernet Industrial CP 343.

El intercambio de información entre el cliente y el servidor OPC ocurre como comunicación interprocesos (COM).

"Cache" se refiere a un buffer temporal que el servidor OPC genera para un grupo en particular. Contiene una imagen local de las variables de proceso definidas para este grupo (las cuales son administradas por el grupo como "items" OPC).

"Dispositivo" se refiere a las variables de proceso en el controlador.

En la tabla 7 se explican las llamadas y flujos de dato respectivos, los números no indican una secuencia definida, son simplemente una herramienta para mapear la instrucción con el diagrama:

No.	**Instrucción**	**Nota**
1	Trabajo de lectura síncrona en el CACHE.	El valor del item OPC es leído con el valor que se encuentra actualmente en CACHE.
2	Refrescando el CACHE	El servidor OPC actualiza todo el CACHE con los valores del Dispositivo después de un periodo de tiempo definido con "requestedUpdateRate".

3	OnDataChange-Event	Para items OPC parametrizados como "activos", el servidor OPC ha hecho un registro un cambio de datos dentro de su CACHE. Esto se reporta al cliente OPC.
4	Trabajo de lectura asíncrona al Dispositivo	El cliente OPC inicia un trabajo de lectura asíncrono vía el grupo OPC. El servidor OPC lee las variables de proceso pedidas de la imagen de proceso del S7-CPU y entrega al cliente OPC vía el evento OnReadComplete. Después escribe los valores de lectura a su CACHE.
5	Trabajo de lectura síncrona al Dispositivo	El valor del item OPC es leído con el valor actualmente localizado en la imagen de proceso del CPU de S7.
6	Trabajo de escritura síncrono o asíncrono	Los trabajos de escritura solo son ejecutados hacia el Dispositivo, sin importar si son síncronos o asíncronos.

Tabla 7. Tipos de relaciones servidor – cliente.

Acceso a CACHE y Dispositivo

La siguiente tabla sintetiza las posibles operaciones para Dispositivo y CACHE:

Operación	Dispositivo	CACHE	Nota
Lectura síncrona	Si	Si	La lectura del CACHE requiere que el grupo y el item correspondiente esten activos.
Lectura asíncrona	Si	No	El servidor OPC lee la variable del Dispositivo. Esto también refresca su CACHE.
Escritura síncrona	Si	No	Los trabajos de escritura siempre son escritos al Dispositivo.
Escritura asíncrona	Si	No	Los trabajos de escritura siempre son escritos al Dispositivo.
Monitoreo (controlado por evento)	X	Si	El monitoreo de variables solo es posible usando el CACHE.

Tabla 8. Tipos de operaciones servidor – cliente.

En la figura Figura 2.3.1.1 b se puede observar la forma como se comunica el OPC Server con el PLC generando grupos, ítems y variables, estos pueden ser modificados automáticamente debido a que el servidor y el PLC están en constante actualización.

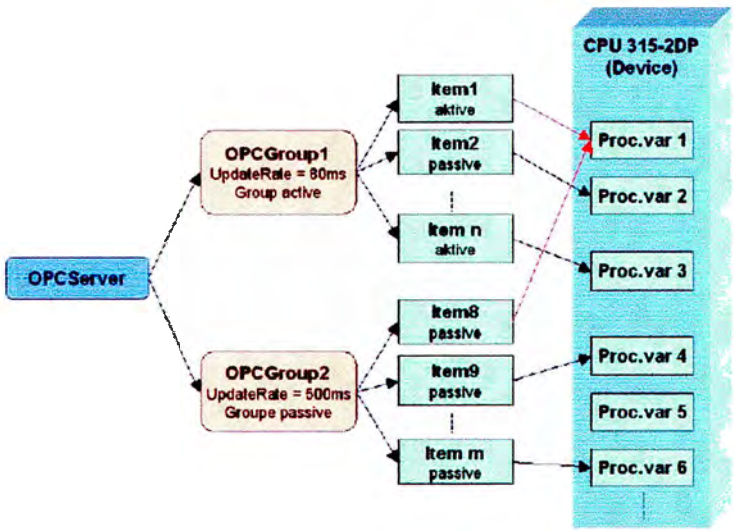


Figura 2.3.1.1 b. Lógica de programación para OPC

Se crean dos grupos, uno para escribir variables y el otro para leerlas.

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	PLC_2_Mitsu0	BOOL	FALSE	
+0.1	PLC_2_Mitsu1	BOOL	FALSE	
+0.2	PLC_2_Mitsu2	BOOL	FALSE	
+0.3	PLC_2_Mitsu3	BOOL	FALSE	
+0.4	PLC_2_Mitsu4	BOOL	FALSE	
+0.5	PLC_2_Mitsu5	BOOL	FALSE	
+0.6	PLC_2_Mitsu6	BOOL	FALSE	
+0.7	PLC_2_Mitsu7	BOOL	FALSE	
+1.0	Mitsu_2_PLC0	BOOL	FALSE	
+1.1	Mitsu_2_PLC1	BOOL	FALSE	
+1.2	Mitsu_2_PLC2	BOOL	FALSE	
+1.3	Mitsu_2_PLC3	BOOL	FALSE	
+1.4	Mitsu_2_PLC4	BOOL	FALSE	
+1.5	Mitsu_2_PLC5	BOOL	FALSE	
+1.6	Mitsu_2_PLC6	BOOL	FALSE	
+1.7	Mitsu_2_PLC7	BOOL	FALSE	
=2.0		END_STRUCT		

Figura 2.3.1.1 c. Base de datos DB1 de S7 para el robot Mitsubishi

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	PLC_2_PUMA0	BOOL	FALSE	Variable provisional
+0.1	PLC_2_PUMA1	BOOL	FALSE	Variable provisional
+0.2	PLC_2_PUMA2	BOOL	FALSE	Variable provisional
+0.3	PLC_2_PUMA3	BOOL	FALSE	Variable provisional
+0.4	PLC_2_PUMA4	BOOL	FALSE	Variable provisional
+0.5	PLC_2_PUMA5	BOOL	FALSE	Variable provisional
+0.6	PLC_2_PUMA6	BOOL	FALSE	Variable provisional
+0.7	PLC_2_PUMA7	BOOL	FALSE	Variable provisional
+1.0	PUMA_2_PLC0	BOOL	FALSE	Variable provisional
+1.1	PUMA_2_PLC1	BOOL	FALSE	Variable provisional
+1.2	PUMA_2_PLC2	BOOL	FALSE	Variable provisional
+1.3	PUMA_2_PLC3	BOOL	FALSE	Variable provisional
+1.4	PUMA_2_PLC4	BOOL	FALSE	Variable provisional
+1.5	PUMA_2_PLC5	BOOL	FALSE	Variable provisional
+1.6	PUMA_2_PLC6	BOOL	FALSE	Variable provisional
+1.7	PUMA_2_PLC7	BOOL	FALSE	Variable provisional
=2.0		END_STRUCT		

Figura 2.3.1.1 d. Base de datos DB2 para el robot Puma

Para que el PLC pueda comunicarse con los Robots, es necesario crear 2 bases de datos (DB). Estas bases de datos se crean en el Administrador Simatic y deben llamarse DB1 y DB2. El DB1 es para la comunicación entre el robot Mitsubishi y el PLC. El DB2 es para la comunicación entre el robot Puma y el PLC.

Como se muestran en las figuras 2.3.1.1 c y 2.3.1.1 d, los primeros ocho bits son para que del PLC mande información a los robots. Los siguientes ocho bits (byte 2) son para que los Robots manden información al PLC.

Tanto en el programa del PLC, como en los programas de los robots, se monitorean continuamente las variables de los DB. No se utilizan eventos para informar a los programas cuando existe un cambio de variable.

2.4 Funcionamiento del sistema SCADA

La implementación del sistema SCADA se hace con el fin de controlar la producción en la celda, proporcionando comunicación con los dispositivos de campo y controlando el proceso de forma automática desde la pantalla de la computadora.

Para cada proceso que se desea observar y manejar se debe crear un proyecto en WinCC. En ese proyecto se deberán definir todos los elementos y ajustes necesarios.

El proyecto WinCC se ejecuta en un equipo que funciona como servidor para el procesamiento de los datos y como equipo de mando. No será posible acceder al mismo desde otros equipos. El equipo en el que se crea un proyecto para estación monopuesto se configura como servidor. El equipo se comunica con el sistema de automatización a través de la comunicación de procesos.

El usuario utiliza las variables de proceso para la comunicación entre WinCC y el autómata programable. Las propiedades de las variables de proceso varían según cuál sea el controlador de comunicación que se use. Es por esta razón por la que el usuario crea las variables de proceso en la Administración de variables bajo un determinado controlador de comunicación, su unidad de canal y sus conexiones.

2.4.1 Aplicación en Graphic Designer

El Graphics Designer es un editor que sirve para la creación y la dinamización de imágenes de proceso.

La imagen en el Graphics Designer se compone de 32 niveles, en los cuales se pueden insertar objetos. La asignación a un nivel define la situación de un objeto en la imagen. Los objetos del nivel 0 están totalmente en el fondo de la imagen y los objetos del nivel 32 están en primer plano. Los objetos se insertan siempre en el nivel activo; pero es posible desplazarlos rápidamente a otro nivel. La asignación a un nivel puede modificarse en la ventana "Propiedades de objeto" mediante el atributo "Nivel".

Además, dentro de un nivel es posible modificar la situación entre los objetos. Para ello hay disponibles cuatro funciones en el menú "Disponer / dentro del nivel". Cuando se crea una imagen de proceso, los objetos de un nivel se disponen de forma estándar en la secuencia de su configuración: así, el objeto que se ha insertado en primer lugar, se encontrará, dentro del nivel, totalmente al fondo; y cada objeto nuevo pasará a ocupar, respectivamente, una posición más adelantada.

Con la paleta de niveles, es posible ocultar todos los niveles, a excepción del activo. De esta forma, los objetos del nivel activo pueden tratarse adecuadamente. La utilización de la técnica de niveles tiene sentido, sobre todo, para la configuración de imágenes que contienen un gran número de distintos tipos de objeto.

Por ejemplo, pueden colocarse todos los objetos de tipo "Pistón retraído" en el nivel 1 y todos los objetos del tipo "Pistón expandido" en el nivel 2. Si entonces se desea cambiar algún atributo de salida, sólo habrá que visualizar el nivel 2 y seleccionar todos los objetos de este nivel. No es posible una selección individual de los campos E/S en la imagen completa.

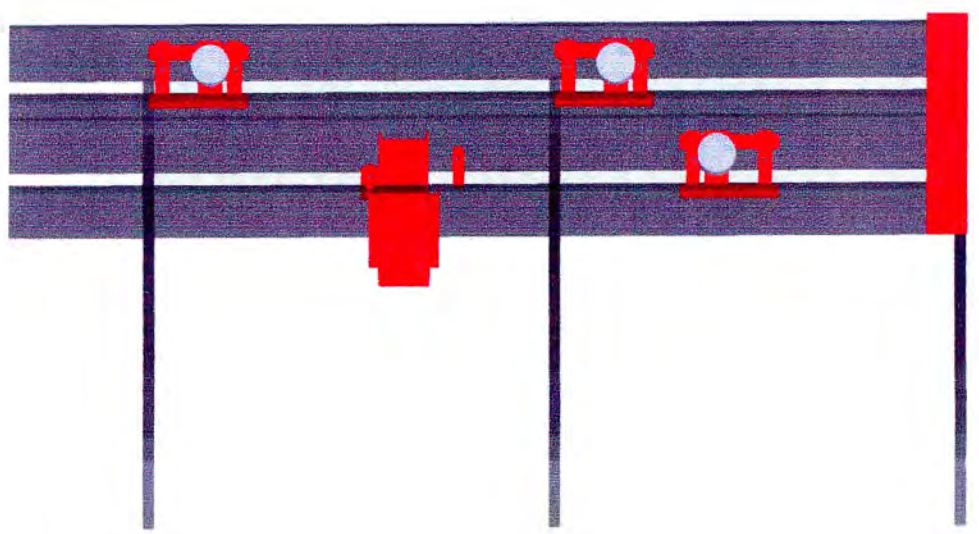


Figura 2.4.1 a.Vista de la pantalla de WinCC para la Celda de Manufactura con pistones retraídos

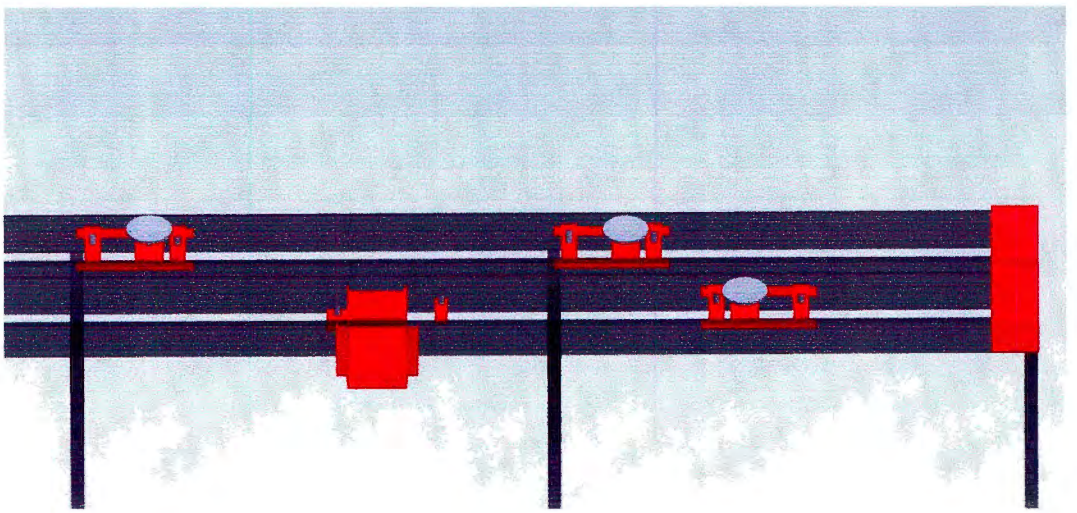


Figura 2.4.1.b. Vista de la pantalla de WinCC para la Celda de Manufactura con pistones activados

Las propiedades de un objeto

La forma, el aspecto, la situación y la conexión con el proceso de un objeto se determinan mediante las "Propiedades del objeto". En el Graphics Designer, estas propiedades pueden modificarse según las necesidades.

Las propiedades de un objeto se describen mediante un gran número de "Atributos". Para modificar una propiedad del objeto, ha de asignarse un nuevo valor a los atributos correspondientes.

La ventana "Propiedades del objeto" contiene en la pestaña "Propiedades", todos los atributos que posee un objeto seleccionado o una selección múltiple de objetos. Los atributos se dividen en grupos de propiedades; por ejemplo, "Geometría" o "Colores". El tipo y la cantidad de los grupos de propiedades y atributos disponibles dependen del tipo de objeto seleccionado. El grupo de propiedades "Fuente" sólo se muestra para aquellos tipos de objetos con los cuales puede representarse un texto.

De forma alternativa a la modificación de los atributos en la ventana "Propiedades del objeto", los objetos pueden adaptarse también con el ratón y el teclado, o mediante la utilización de las barras de herramientas y de paletas. Sin embargo, de esta forma sólo pueden modificarse determinadas propiedades del objeto, como tamaños geométricos básicos, colores, estilos de línea o su visualización.

Para la creación de imágenes de proceso, es importante, en primer lugar, adaptar los valores estáticos de los atributos, para definir, la forma, aspecto, situación o posibilidad de manejo de un objeto. Sin embargo, la ventana "Propiedades del objeto" permite también la dinamización de imágenes de proceso. Mediante el vínculo de los atributos con cuadros de diálogo dinámicos, acciones C o variables; es posible adaptar,

dinámicamente, las propiedades de un objeto a las exigencias de los procesos que se van a representar.

La ventana "Propiedades del objeto" está subdividida en los elementos barra de herramientas, pestaña "Propiedades" y pestaña "Evento".

Selección de la acción

Se visualiza la acción, que se ejecuta al entrar el evento. La acción seleccionada se marca con uno de los siguientes iconos:

- Rayo blanco = no existe acción para el evento
- Rayo azul = para este evento existe una acción a través de una conexión directa.
- Rayo verde = para este evento existe una acción C.
- Rayo amarillo = para este evento existe una acción C que todavía no está compilada.

Puede modificarse la acción configurada haciendo doble clic en la columna "Ejecutar", o bien llamando al menú emergente de la columna "Acción".

Activar eventos

Evento	Ejecutar con	Descripción
Ratón	Clic del ratón	Se activa cuando, tras hacer clic y soltar el botón del ratón, el puntero se sitúa encima del objeto.
Ratón	Pulsar izq/der	Tras pulsar el botón del ratón, se acciona directamente sobre el objeto.
Ratón	Soltar izq/der	Al soltar el botón de ratón, se acciona para el objeto, sobre el cuál se encontraba el puntero cuando se pulsó el botón del ratón.
Teclado	Pulsar	Se acciona cuando se pulsa una tecla. Las teclas <F10> y <ALT+ Imprimir> no pueden utilizarse para las operaciones de bajo nivel.
Teclado	Soltar	Se activa al soltar una tecla del teclado. Las teclas <F10> y <ALT+ Imprimir> no pueden utilizarse para las operaciones de bajo nivel.
Foco	Modificación del foco	Se produce cuando se adquiere o se pierde el foco; se llama a esta función mediante una acción C o una acción VBS; o mediante la selección del objeto con <TAB> (secuencia TAB).

Tabla 9. Tipos de eventos en WinCC.

Para poder accionar los pistones de manera manual a través de la pantalla de la computadora de monitoreo debe seguirse una configuración en los pistones de la Celda por medio del Graphic Designer de WinCC. Para cada pistón existen dos etapas, una en la que el pistón se encuentra retraído y otra en la que se encuentra expandido.

El objetivo es que al dar clic sobre un pistón retraído el mismo se expanda y viceversa. También se busca que cuando un pistón se encuentra expandido podamos ver la imagen del mismo afuera y cuando está retraído lo veamos dentro.

Los pasos a seguir para establecer el accionamiento de un pistón a partir de un clic sobre su imagen en la pantalla se encuentran en el Manual de usuario en la sección Anexos.

De igual modo, los pasos a seguir para establecer la visualización de un pistón a partir de un clic sobre su imagen en la pantalla se encuentran en el Manual de usuario en la sección Anexos.

2.4.2 Programación del Panel de Control HMI (Interfase hombre-máquina)

Figura 2.4.2.a. Esquema de conexiones Panel-PLC-Computadora

La programación del panel se hace con WinCC flexible. Al crear un proyecto en WinCC flexible o al abrir uno ya existente, aparece la estación de trabajo en la pantalla del equipo de configuración. En la ventana de proyecto se representa la estructura del proyecto y se visualiza su estructura.

WinCC flexible incluye un editor específico para cada tarea de configuración. Por ejemplo, la interfaz gráfica de usuario de los paneles de operador se configura en el editor "Imágenes". Para configurar los avisos se emplea el editor "Avisos de bit".

Todos los datos de configuración que pertenecen a un mismo proyecto se almacenan en la base de datos del proyecto.

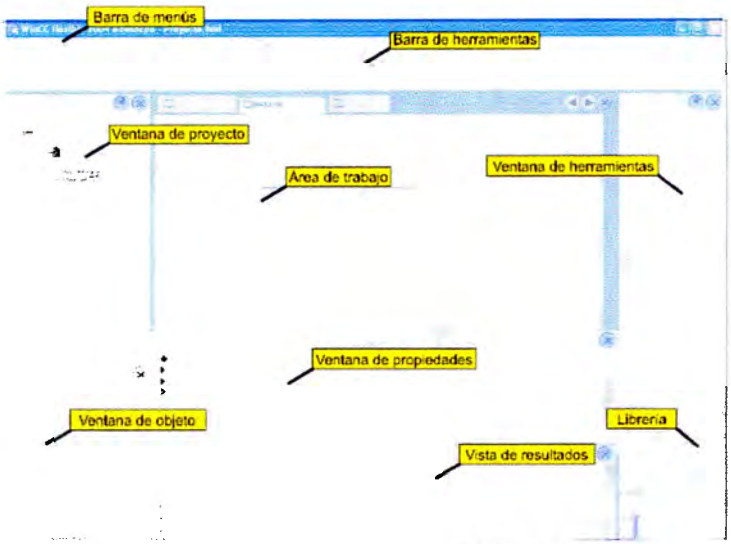


Figura 2.4.2.b. Área de trabajo en WinCC flexible.

Los pasos a seguir para la programación del panel se especifican en el Manual de usuario de la sección de Anexos.

2.5 Sistema de Análisis de Formas

El sensor de visión SIMATIC VS 110 es una solución rápida, de bajo costo y efectiva para simplificar una inspección visual de objetos pequeños.

Este opera usando una técnica de luz de fondo. Se crea una sombra de la forma del objeto a probar. La prueba consiste en la comparación de la silueta con un modelo de entrenamiento chocando si el objeto esta presente, si la silueta y el área están correctos, si esta alineado correctamente y si el objeto esta terminado.

2.5.1 Instalación de equipo

El sensor de visión consiste de:

- Una cabeza de sensor con un chip para sensar los objetos. El sensor es sensible a la luz en un rango cercano al infrarrojo.
- Una unidad de evaluación para procesar la información de la imagen, permite que el control del sensor y la comunicación con otros dispositivos como un controlador SIMATIC o PLC.
- Una unidad de iluminación infrarroja para crear una sombra de la imagen del objeto a evaluar.
- Cables para conectar los componentes individuales.

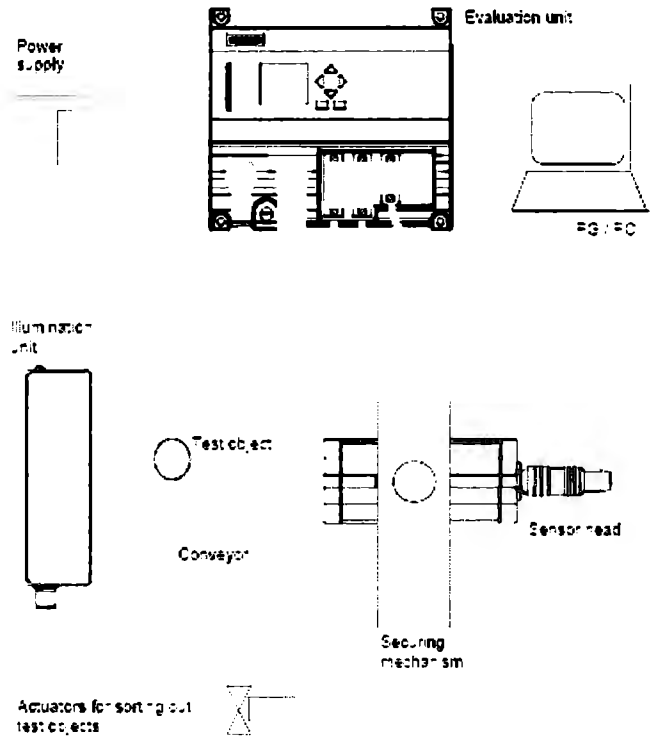
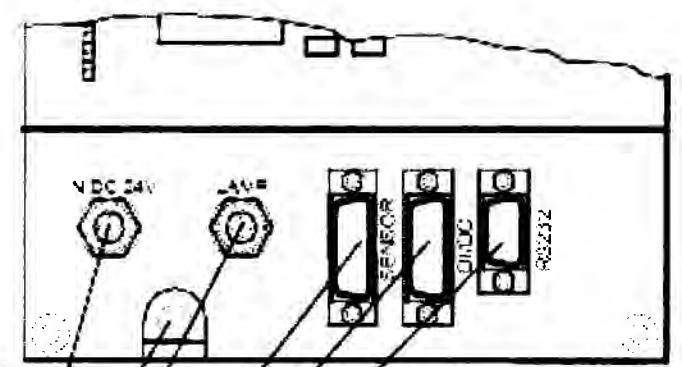


Figura 2.5.1.a. Configuración del sistema de análisis de forma.



Connector	Type	No. Pins x Cable Cross Section	M/F
Power supply	Circular M12	4 x 0.55 mm²	Male
Functional ground	Screw connector	-	-
Illumination unit	Circular M12	4 x 0.22 mm²	Female
Sensor head (shielded cable)	HD sub-D	25 x 0.09 mm²	Female
I/O	Sub-D	15 x 0.14 mm²	Female
RS-232	Sub-D	9	Male

Figura 2.5.1.b. Tipo de conectores del sistema de análisis de forma.

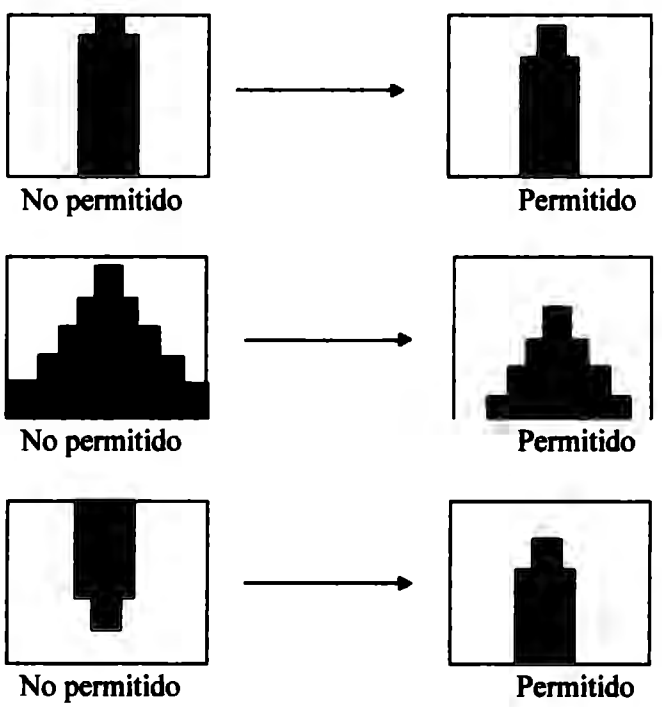
Para obtener un entrenamiento y una evaluación correcta es necesario tener capturado el objeto completamente en el campo del sensor y para esto se tiene dos métodos párale accionamiento:

- Accionamiento Automático:
  Cuando se utiliza este método las imágenes del objeto son guardadas y evaluadas continuamente. Este método solo se utiliza cuando el objeto se pasa de forma horizontal en el campo de visión del sensor.

- Accionamiento Externo:
  En este método se puede activar la evaluación con una señal externa y la imagen es capturada y evaluada solo cuando se genere esta señal.

2.5.2 Implementación

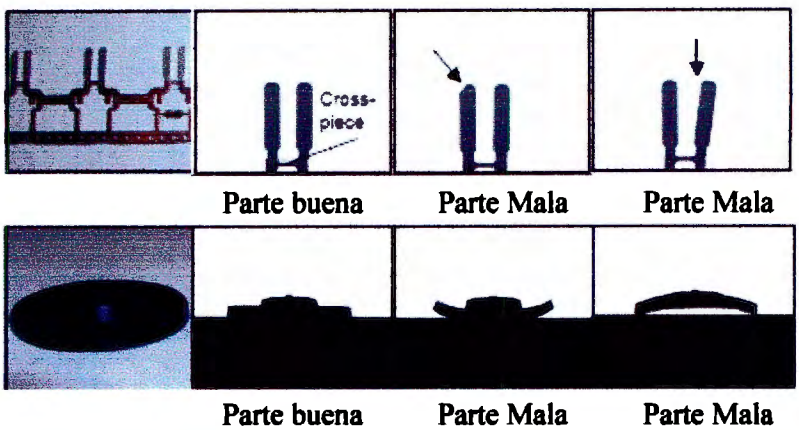
El objeto a evaluar solamente debe tocar el eje inferior del campo de visión del sensor y como resultado algunos arreglos pueden ser permitidos y otros no.



Aplicaciones:

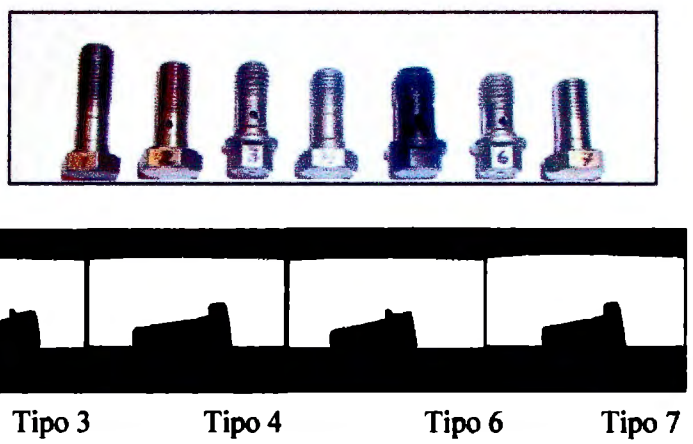
Este sistema es utilizado para realizar un análisis de calidad de las piezas por lo que se tiene diversas aplicaciones:

- Distinguir entre una pieza buena o mala



- Distinguir entre una vista A y una Vista B (Checar el sentido de la pieza)

- Distinguir entre diferentes tipos de modelos



- Distinguir la posición del objeto

Correcta Incorrecta Incorrecta

Este sistema tiene la posibilidad de realizar dos análisis que son muy importantes en la industria como lo son el límite de calidad y un límite de desviación vertical del objeto; Este análisis se realiza automáticamente después de que el usuario ha programado los rangos y la unidad de evaluación toma la decisión si acepta o rechaza la pieza evaluada.

Limite de calidad:

Se hace automáticamente y el rango de evaluación va de 0 a 99.9%. Y este nos permite saber si el objeto fue maquinado o ensalmado correctamente.

Limite de desviación vertical:

Este proceso evalúa la desviación vertical entre el modelo y el objeto a evaluar, este proceso se realiza de la misma forma que la evaluación de límite de calidad, el rango es de 0 a 9.99mm. Y este nos permite saber si el objeto esta en una posición correcta.

3 Resultados y Pruebas Globales

Al unir el programa que generaba la interfaz gráfica y el programa de control del robot se debieron ajustar varios detalles. Se realizó una serie de pruebas para detectar errores en el funcionamiento global del programa y se fueron actualizando las versiones del mismo hasta llegar al resultado final. Varios de los problemas que se presentaron se explican en las pruebas globales de los párrafos siguientes.

Dos de las causas por las cuales se puede presentar un fallo en la comunicación serial entre el robot y la computadora son que el cable se encuentre desconectado o flojo o que el control manual del robot se encuentre en On en el caso del Mitsubishi o en Manual en el caso del Puma. Si cualquiera de estas dos condiciones se cumplía cuando se corría el programa durante las primeras pruebas, la aplicación se congelaba. Para resolver esto se detectó el estado de las señales del robot en ambos casos y en el código del programa se implementó el caso en que alguna o ambas condiciones se cumplieran de forma que ahora el programa despliega una ventana con un mensaje para el usuario indicando que no existe comunicación con el robot por alguna de las dos causas mencionadas.

Una vez que el robot concluía la secuencia fue difícil saber el momento exacto en el que esto sucedía. Como una solución a este problema se incluyó en el programa la señal CTS (clear to send), la cual al cambiar su estado ayudaba al código del programa a establecer el momento en que un paso concluía y se podía seguir con uno nuevo.

Al momento de bajar puntos a la aplicación la barra de herramientas de reproducción se encontraba habilitada, lo que daba la posibilidad de presionar el botón de ejecución sin que se hubiese terminado de ejecutar la comunicación entre el robot y el programa para bajar puntos y esto generaba un error. Para resolver este problema se deshabilitaron tanto la barra de reproducción como la de manejo de puntos mientras alguno de los botones de éstas barras se encontrara en acción.

El icono para Moverse a un Punto se encontraba vacío por default, al guardarlo y reabrirlo generaba un error. La forma de solucionar esto fue dando un valor en código que posteriormente cambia de acuerdo al usuario.

Al momento de bajar datos del robot, es decir, adquirir la información acerca de los puntos que el mismo tenía almacenados, se perdían datos ya que se borraba el buffer antes de tiempo. Este problema fue resuelto permitiendo que el buffer se borre automáticamente al terminar de recibir cada dato.

Una vez hechas las pruebas suficientes tanto para el programa que maneja al robot Puma como para el del Mitsubishi se crearon las versiones finales con sus ejecutables. Los ejecutables para los robots se instalaron en una computadora reciente con Windows XP que se encuentra en la celda de manufactura. Ahí se corrieron una vez más para corroborar su buen funcionamiento.

Las figuras 3a y 3b muestran las dos diferentes formas de programar al robot Puma para realizar la misma secuencia. La programación con código de bajo nivel requiere de conocimiento previo de las instrucciones a mandar, además de que resulta difícil seguir la secuencia del robot. Con la interfaz gráfica la programación es sencilla debido a que es visual y no requiere de conocimiento previo, aunado a esto, mediante el nuevo programa es posible tener un monitoreo en tiempo real del estado del robot.

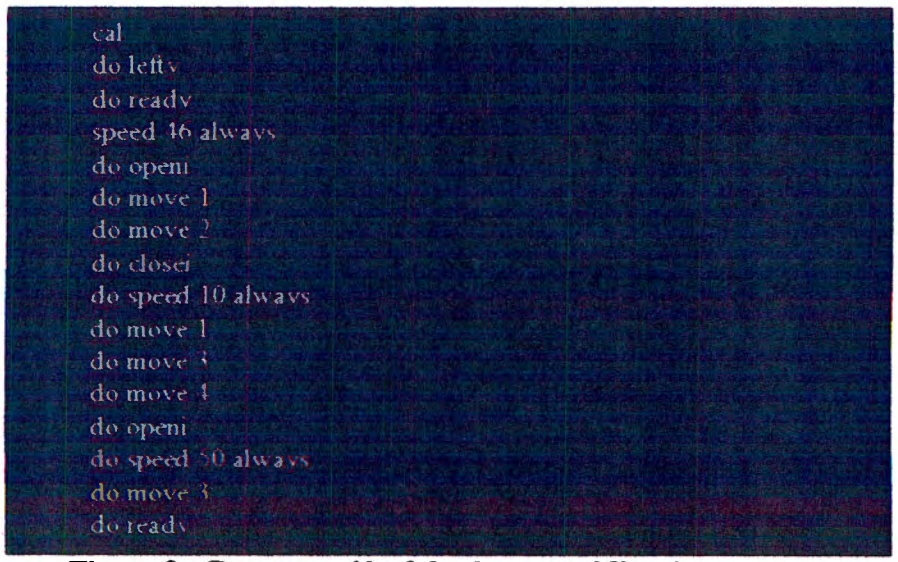


Figura 3a. Programación del robot con código de bajo nivel

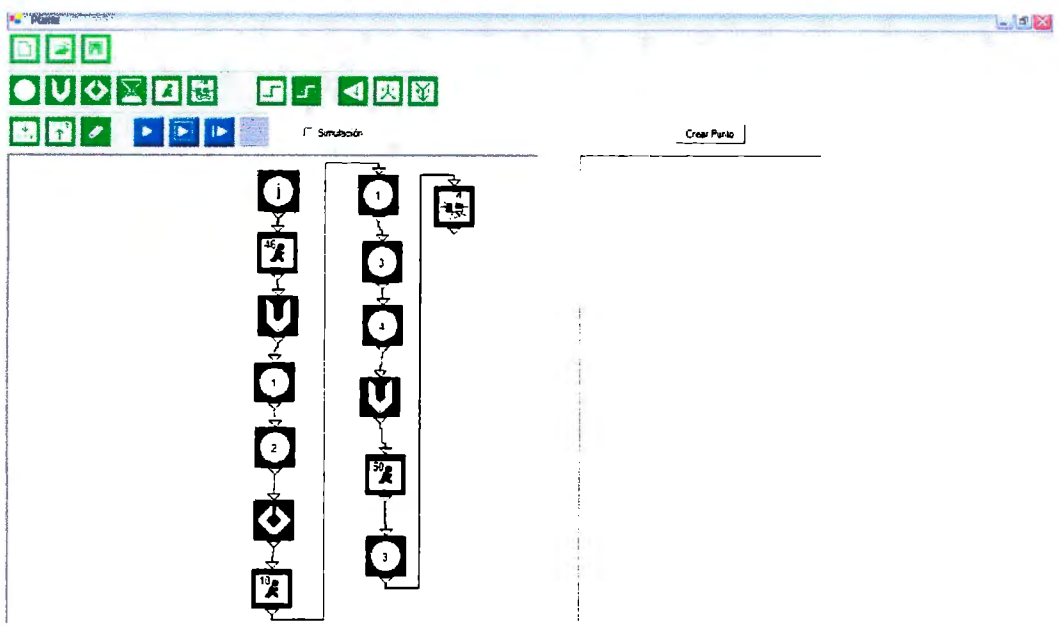


Figura 3b. Programación del robot mediante Interfaz Gráfica

Las figuras 3c, 3d y 3e muestran el movimiento del robot Puma siguiendo una secuencia programada. En la pantalla de la computadora se puede observar el paso que el Puma ejecuta en ese momento. Dentro de la secuencia que realiza, el robot lleva una probeta de un punto a otro para después depositarla abriendo su gripper y volver a su punto de inicialización o "Nest".

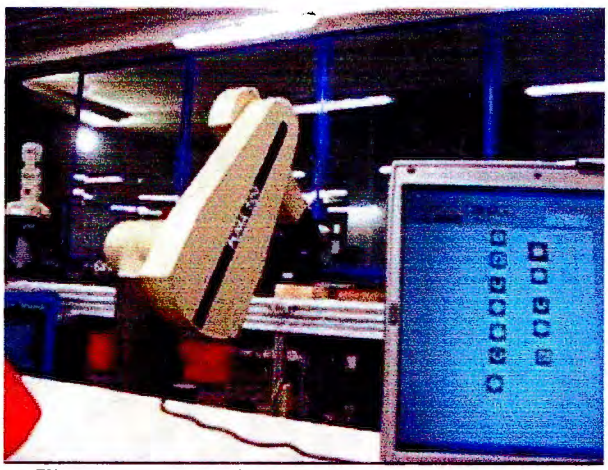


Figura 3c. El robot Puma ejecuta el 8° paso de su secuencia.

Figura 3d. El robot Puma ejecuta el 10° paso de su secuencia.

Figura 3e. El robot Puma ejecuta el 12° paso de su secuencia.

4 Conclusiones

- Correcta integración de un proceso de manufactura en tiempo real
- Desempeño Satisfactorio de control de robots a través del programa
- Comunicación entre sistema de monitoreo y PLC satisfactorio
- Correcto envío y recepción de señales entre PLC y robots
- Control de robots y proceso de manufactura en tiempo real
- Correcta implementación de Ethernet Industrial en la celda

5 Trabajo Futuro

Se espera que en semestres futuros se continúe mejorando la Celda de Manufactura, poniendo especial énfasis en lo siguiente:

La anexión del ASRS a la nueva estructura de la Celda de Manufactura, teniendo el almacén una computadora que se encuentre comunicada mediante OPC a la computadora de monitoreo.

6 Bibliografía

- www.automatas.org/redes/scadas.htm
- www.profesores.frc.utn.edu.ar/industrial/sistemasinteligentes/UT3/plc/PLC.html
- www.monografias.com/trabajos/objetos/objetos.shtml
- www.fanuc.co.jp/en/product/robot/networking/networking.html
- www.devhood.com/tutorials/tutorial_details.aspx?tutorial_id=320
- Manuales de Brazos Robóticos Mitsubishi y Puma.
- Manual de Siemens: "DP OPC Starter Kit". Documento en PDF.

- Documento PDF de Siemens: "Mass Data Acquisition with an OPC Client in C# based on .NET" en línea
  http://support.automation.siemens.com/WW/view/en/21447513
- Manual de Siemens: "Step by Step: Ethernet Communication between OPC Server and S7-200 incl. CP243-1." Documento en PDF.

7 Anexos

7.1 Manual de usuario

7.1.1 Estructura de la Celda de Manufactura

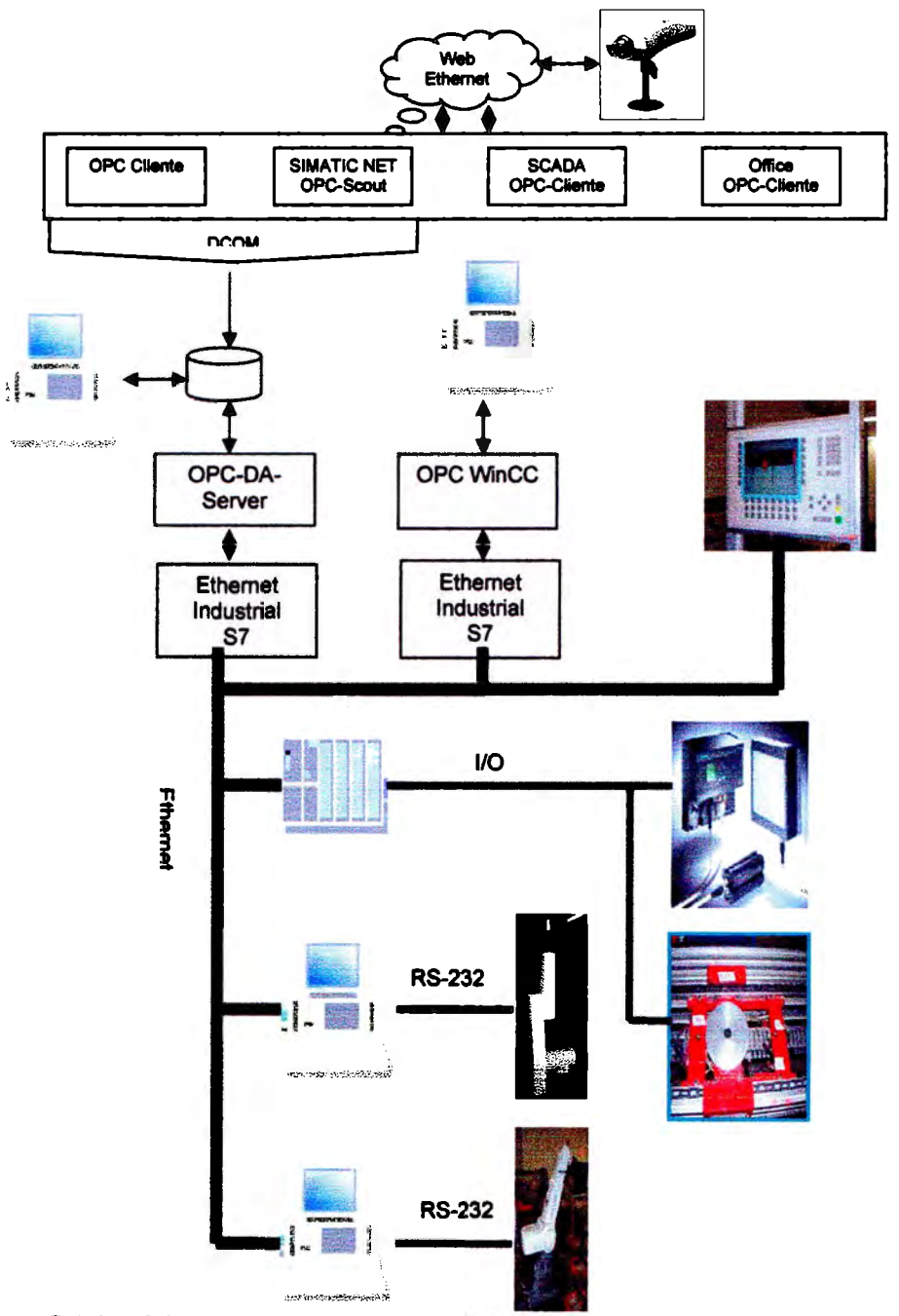


Figura 7.1.1 a. Diagrama de conexiones de la Celda de Manufactura actual

La celda cuenta con 4 estaciones, en la estación 1 se encuentra el robot Puma, en la estación 2 el robot Neptuno, el cual por el momento no está funcionando, en la estación 3 se encuentra el robot Mitsubishi y finalmente, en la estación 4 el sistema de visión.

Las estaciones 1,2 y 3 cuentan con un sensor y tres actuadores, la estación 4 cuenta con un motor, 3 actuadores y 3 sensores.

También se cuenta con un sensor de cambio de sentido y un actuador para pasar a la otra banda.

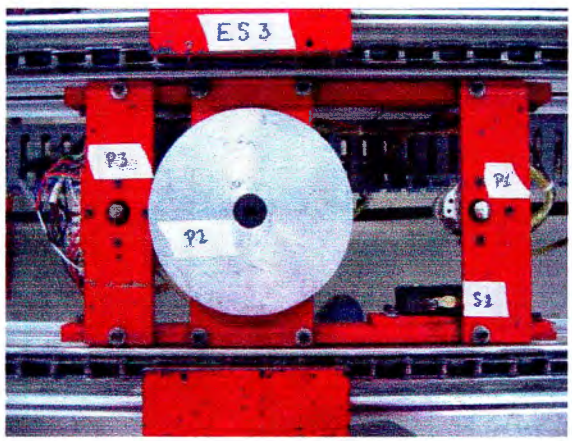


Figura 7.1.1 b. Componentes de la Estación 1 de la Celda

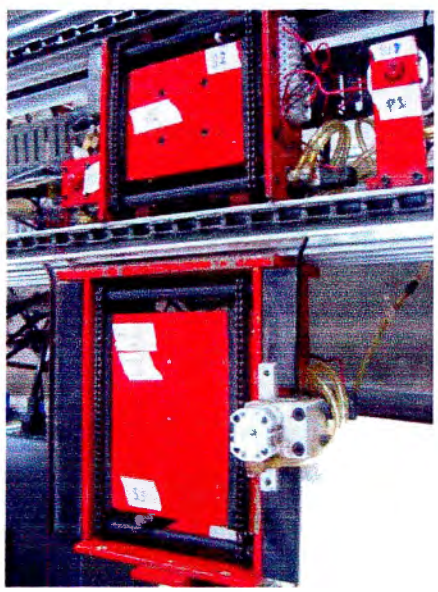
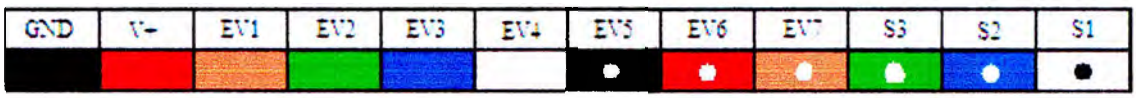


Figura 7.1.1 c. Componentes de la Estación 4 de la Celda

En cuanto al cableado de sensores y actuadores hacia el PLC, existen 3 cables principales, cada uno respeta la distribución siguiente:



EV: electroválvula
S: sensor
GND: tierra
V: voltaje

En el cable 1 se encuentra la estación 1, en el cable dos están la estación 2, la 3 y el sensor y actuador de cambio de sentido, en el cable 3 está la estación 4, manejándose la activación del motor y su cambio de sentido como dos electroválvulas. Estos cables van conectados a las entradas y salidas del PLC de forma consecutiva.

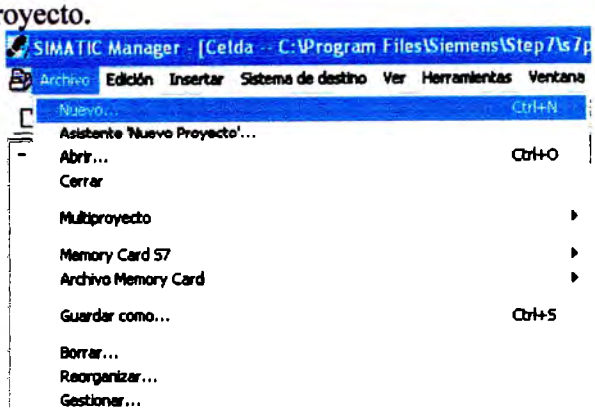
Para crear la red de Ethernet entre las tres computadoras se estableció para cada una su dirección IP y su nombre, los cuales se enlistan a continuación.

10.48.159.117	PC MONITOREO
10.48.159.118	Camara IP
10.48.159.119	PCservidor
10.48.159.120	PCmitsubishi
10.48.159.121	PLC
10.48.159.122	Pcpuma
10.48.159.123	HMI

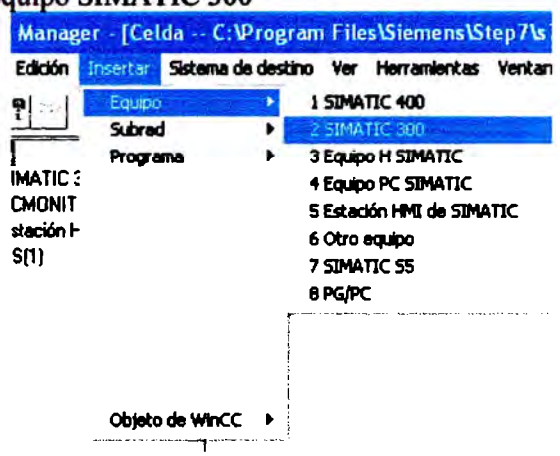
Subnet mask:	255.255.248.0
Default geteway:	10.48.159.250
Preferes DNS server:	10.48.138.135
Alternate DNS server:	10.48.138.135

7.1.2 Configuración en STEP7

1. Abrir la administrador SIMATIC.
2. Abrir nuevo proyecto.



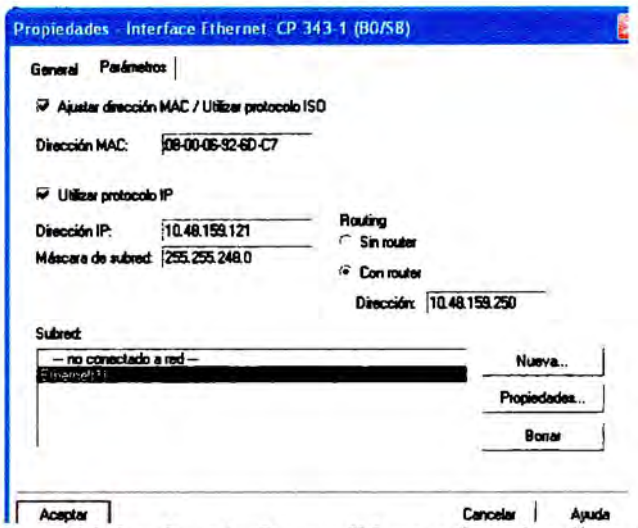
3. Insertar nuevo equipo SIMATIC 300



4. Entrar a configuración del hardware y agregar lo siguiente:

Slot	Módulo ...	Referencia
1	PS 307 2A	6ES7 307-1BA00-0AA0
2	CPU 314C-2 DP	6ES7 314-6CF00-0AB0
X2	DP	
2.2	DI24/DO16	
2.3	AI5/AO2	
2.4	Contaje	
2.5	Posicionamiento	
3		
4	DI16xDC24V	6ES7 321-1BH00-0AA0
5	DI16xDC24V	6ES7 321-1BH00-0AA0
6	DO16xDC24V/0.5A	6ES7 322-1BH00-0AA0
7	DO16xDC24V/0.5A	6ES7 322-1BH00-0AA0
8	CP 343-1	6GK7 343-1EX11-0XE0
9		
10		
11		

5. Configurar el CPU dependiendo de lo que se necesite haciendo doble clic en la celda
6. Configurar el CP 343-1 y asignarle direccion IP, marcara de subred y direccion de router.



7. Configurar los módulos de entradas y salidas y asignarles direcciones
8. Compilar y subir al modulo

7.1.3 Implementación de la comunicación OPC con STEP 7 y SIMATIC NET

El servidor OPC de SIMATIC NET puede acceder los datos del SIMATIC S7 vía una conexión S7 (TCP/IP). Con ProTool como cliente OPC uno puede leer estos datos del servidor de OPC de SIMATIC NET vía una conexión OPC.

Requerimientos de Hardware:
PC con sistema operativo Microsoft Win 2000/XP
CP o tarjeta Ethernet estándar
S7-300 con CPU
Cable RJ45 IE

Requerimientos de Software:
Step 7 de SIMATIC
SIMATIC NET V6.2

<u>1) Configuración del servidor OPC SIMATIC NET</u>

1. Abrir el proyecto Step 7 con el SIMATIC Manager.
2. Por la vía "Insert Station" se inserta una "Estación SIMATIC PC" en el proyecto.

Figura 7.1.3 a. Insertando una estación PC

3. Se da doble clic a "Configuración" para abrir la configuración de Hardware de la estación PC. Del catalogo de Hardware ("SIMATIC PC station") se inserta el servidor OPC V6.1 en la casilla 1 y un IE General en la casilla 2 para la tarjeta de red estándar de la PC.

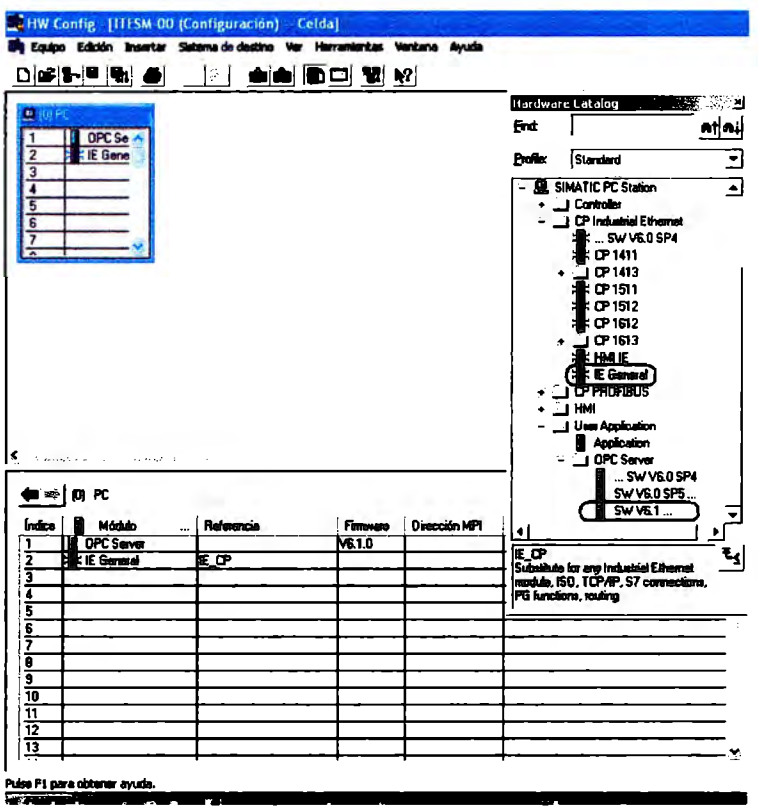


Figura 7.1.3 b. Insertando los componentes

4. Se da doble clic al servidor OPC para abrir la ventana de Propiedades. Se selecciona la pestaña S7. En esta ventana se define si el servidor OPC debe usar símbolos de STEP 7. Si se selecciona la opción "All", todos los símbolos de STEP 7 se hacen disponibles en el servidor OPC.

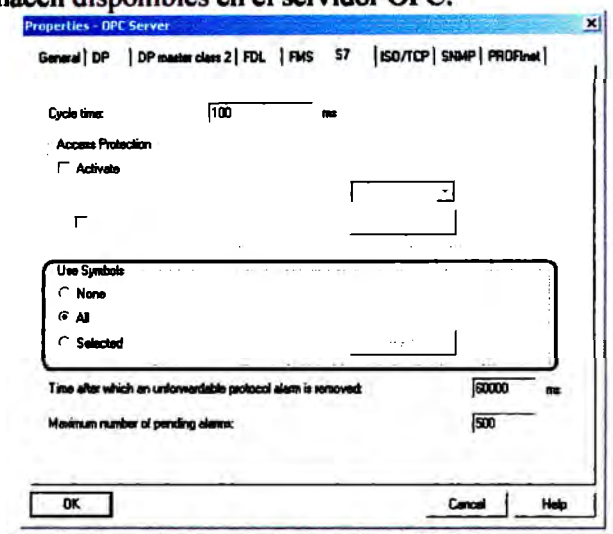


Figura 7.1.3 c. Activando los símbolos de STEP 7 para el servidor OPC

5. Se abre la ventana de Propiedades del IE General con un doble clic. Por la vía "Propiedades" se liga el IE General a Ethernet del controlador S7.

6. Se utiliza la dirección de IP de la tarjeta de red establecida en Windows como dirección de IP para el IE General. Se puede leer la dirección de IP actual por medio del prompt MSDOS y el comando "ipconfig".

Ejemplo:

Dirección IP del controlador:	192.168.0.1
Dirección IP de la PC:	192.168.0.2
Máscara de subred:	255.255.255.0

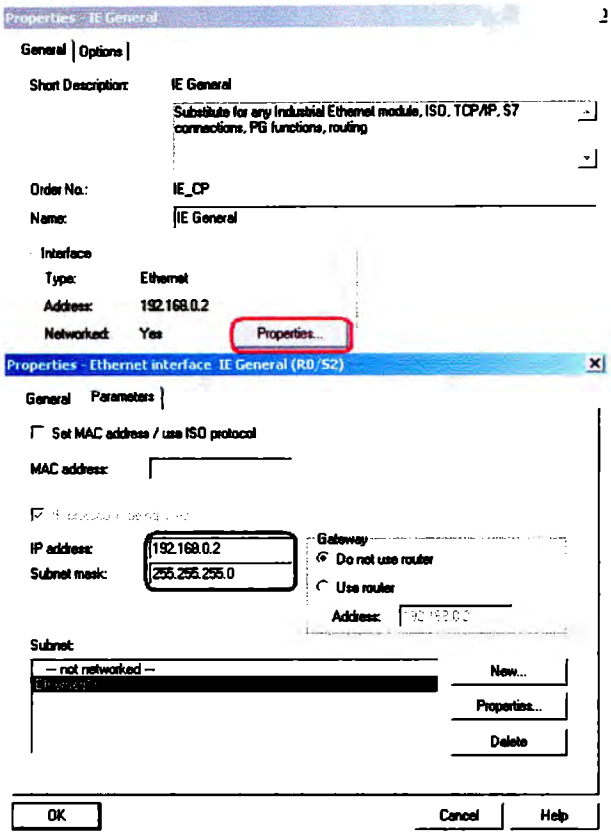


Figura 7.1.3 d. Estableciendo la red del IE General

7. Se debe guardar y compilar la configuración.
8. Vía Options > Configure Network se cambia al programa NetPro. Se debe marcar el servidor OPC en la estación PC y vía Insert > New Connection se inserta una conexión de S7.

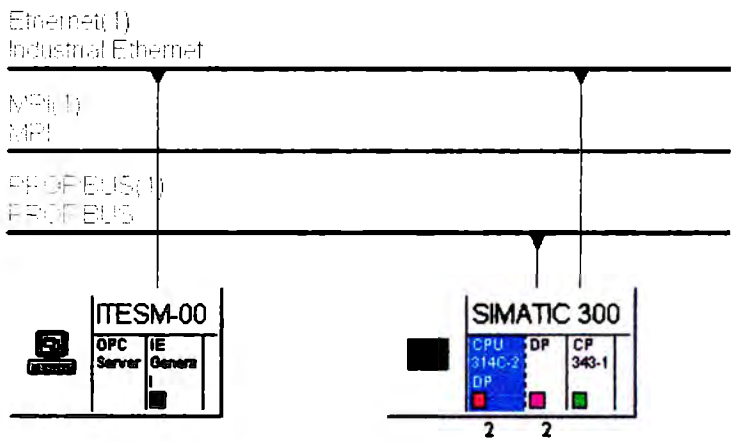


Figura 7.1.3 e. Insertando una conexión S7 nueva

9. Se selecciona el controlador S7 como compañero de conexión y "S7 connection" como tipo de conexión. Se cierra la ventana con "OK".

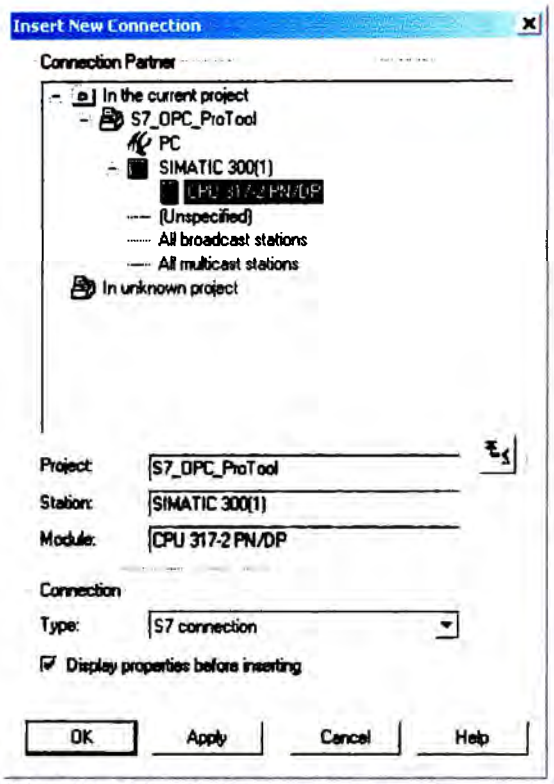


Figura 7.1.3 f. Definiendo los parámetros de conexión

10. Se guarda y compila la configuración en NetPro.
11. Se debe abrir el "Station Configuration Editor" vía la barra de herramientas o el icono en el escritorio.

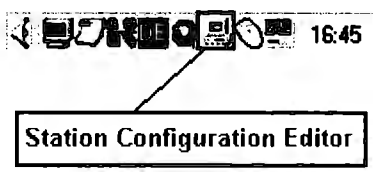


Figura 7.1.3 g. "Station Configuration Editor"

12. En el "Station Configuration Editor" se agrega el servidor OPC al Index 1 y el IE General al Index 2 vía "Add…".

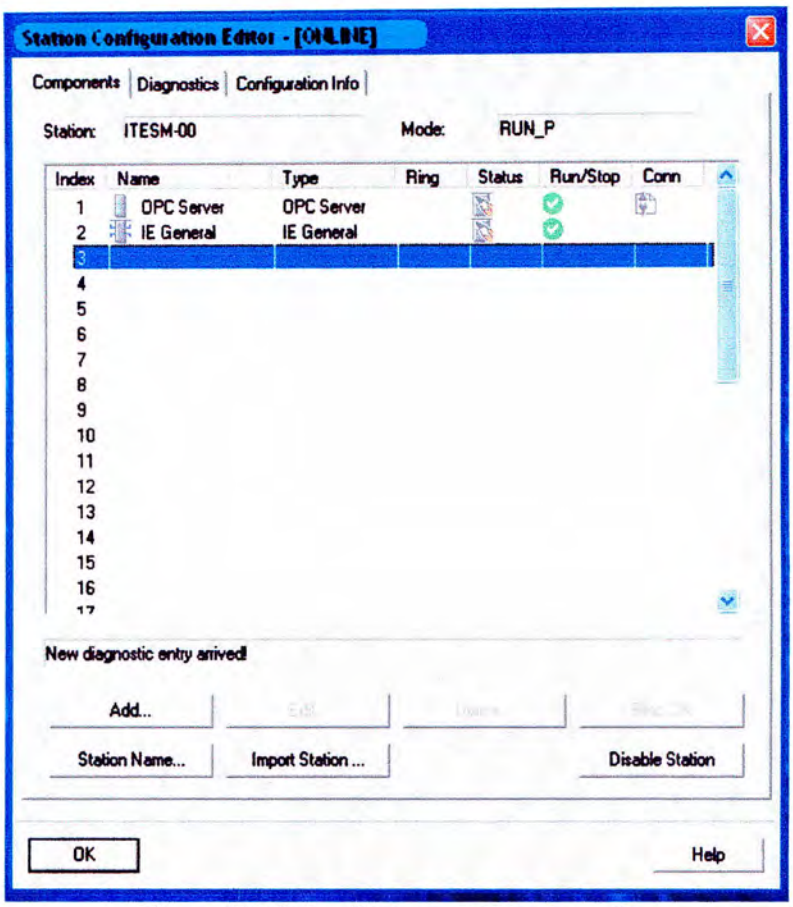


Figura 7.1.3 h. Componentes agregados

13. Ahora se importa la estación PC del NetPro.
14. Se cierra NetPro y la ventana de Configuración de Hardware de la estación PC.

2.) Probando la conexión S7 del servidor OPC al controlador S7 con OPC Scout

1. Iniciar el OPC Scout vía Start > SIMATIC > SIMATIC NET > Industrial Ethernet > SOFTNET Industrial Ethernet > OPC Scout.
2. Conectarse al servidor OPC al hacer doble clic a "OPC.SimaticNet". En la ventana que se abre teclear el nombre del grupo de su elección y reconocerlo con "OK".

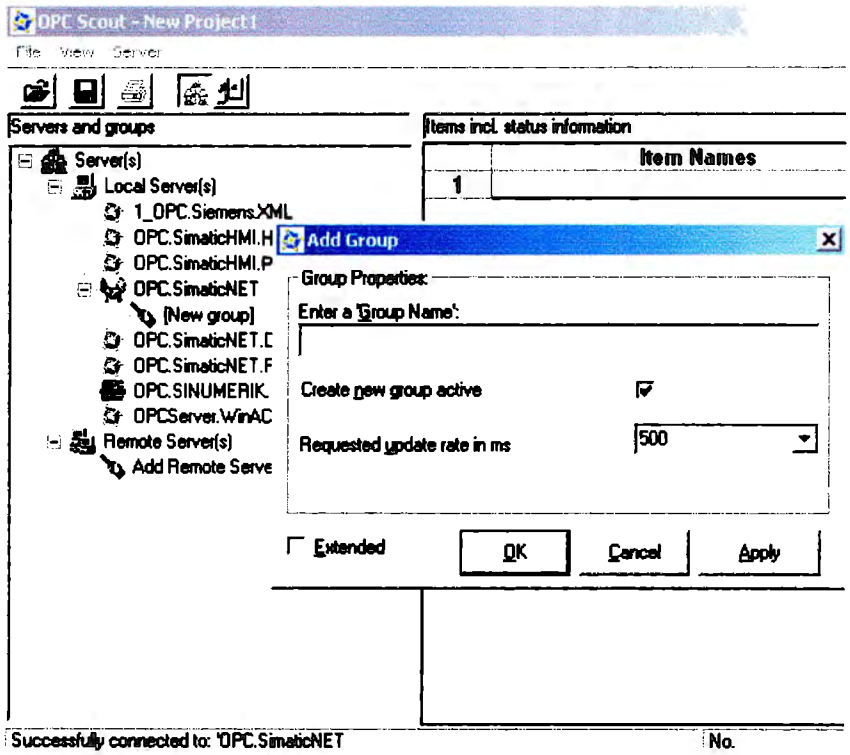


Figura 7.1.3 i. Conectándose con el servidor OPC

3. Se da doble clic al nombre de grupo para abrir el navegador OPC. Los símbolos de STEP 7 incorporados se pueden obtener en el fólder "\SYM". Abrir el navegador haciendo doble clic en "\SYM".

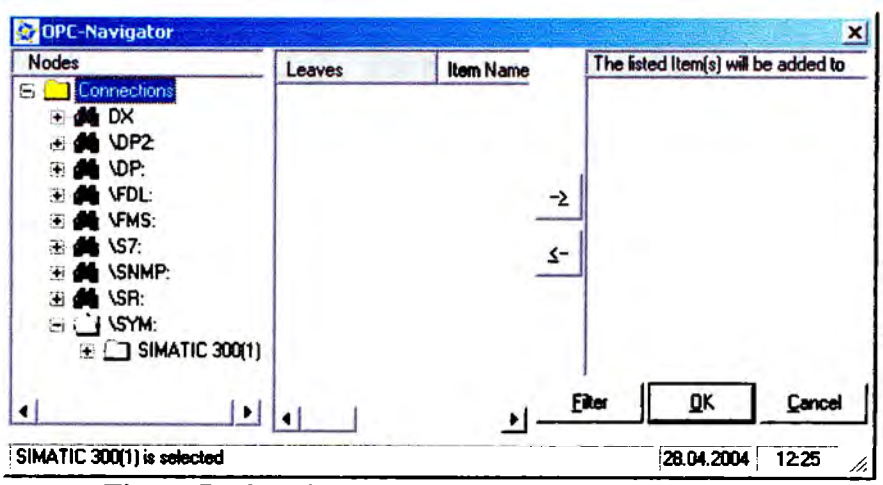


Figura 7.1.3 j. Conexión de S7 en el navegador OPC

4. Navegar al siguiente nivel de abajo haciendo doble clic en la estación SIMATIC. Si se marca la CPU en la navegación se pueden ver los símbolos de la tabla de símbolos de STEP 7. Usando el botón derecho se seleccionan las variables como elementos OPC.

5. Para definir elementos para el servidor OPC se debe abrir el fólder con un doble clic. La conexión S7 creada en NetPro se despliega. Se da doble clic en la conexión S7 una vez más u los objetos aparecen en el árbol con los objetos que pueden ser accesados. Vía "New Definition" se pueden definir elementos propios. Usando la flecha derecha se seleccionan los elementos para la lista en el OPC Scout.

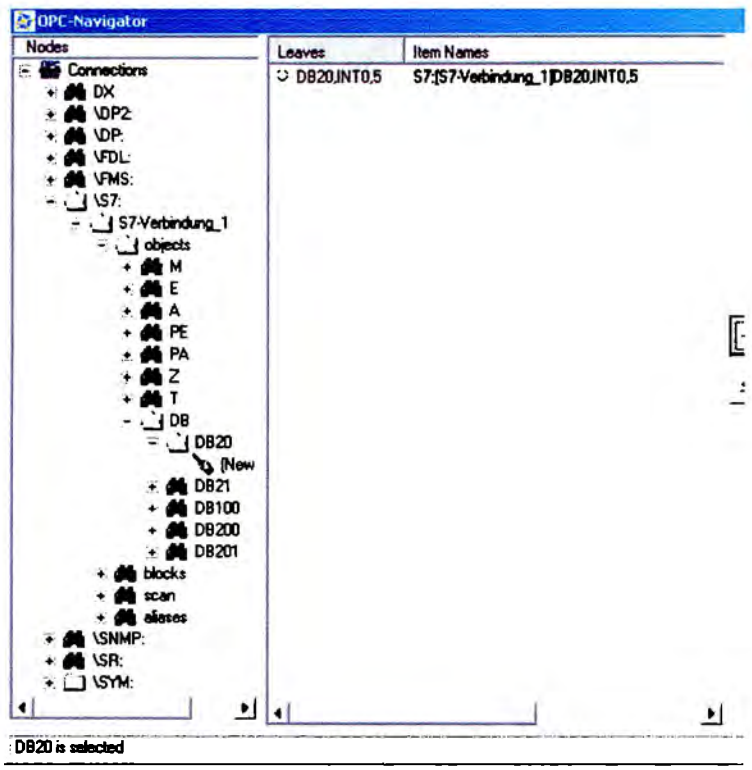


Figura 7.1.3 k. Creando elementos propios

6. Se cierra el navegador de OPC con "OK". Los elementos seleccionados son transferidos al OPC Scout. Si la calidad es buena entonces la conexión se establece y se puede leer/escribir elementos.

7.1.4 Propiedades del DCOM con dcomcnfg

7.1.4.1 Configuración en el cliente

Para permitir a un cliente utilizar un objeto COM en otra computadora, las propiedades del objeto COM deben estar configuradas en el cliente y en una computadora

remota. Se puede configurar el DCOM y los objetos COM requeridos usando el programa dcomcnfg del sistema Windows.

En las siguientes páginas se describe la forma en que tanto la computadora cliente como la computadora servidor deben estar configuradas.

Para llamar al dcomcnfg se pueden hacer dos cosas:

1. Escribir "dcomcnfg" en la ventana de "Run".
2. En el menú de inicio de Windows seleccionar "Settings -> Control Panel". En la ventana que se abre seleccionar el icono "Administrative Tools" y después el icono "Component Services".

En Windows XP se debe dar clic derecho en la ventana que se abre y seleccionar el menu de contexto "My Computer".

Esto abre la ventana de propiedades.

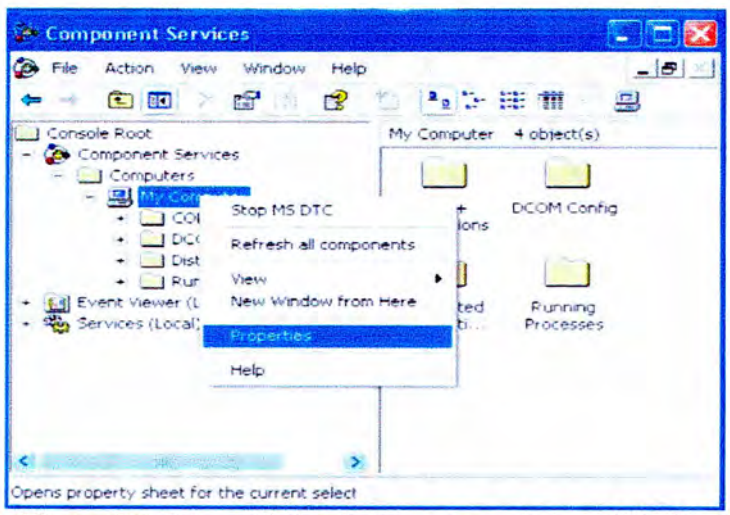


Figura 7.1.4.1 a. Abriendo la ventana de propiedades

Las pestañas "General", "Opciones" y "MSDTC" solo aparecen en Windows XP y no debe cambiarse nada en ellas. Las pestañas "Default Properties", "Default Protocols" y "Default COM Security" tienen las mismas opciones que en otras versiones de Windows.

Se debe tomar en cuenta que si la configuración de seguridad del sistema se cambia el sistema debe reiniciarse para activar los cambios.

Pestaña "Default Properties"

En esta pestaña se especifican las propiedades básicas de DCOM. Las configuraciones hechas para el DCOM dependen de si la computadora con el servidor

OPC se conecta en un dominio o es operada como un grupo de trabajo. Si se opera como dominio la computadora servidor puede chequear los derechos de configuración de otras cuentas en la red.

Para la operación DCOM con el servidor OPC en un grupo de trabajo se deben seguir los pasos mostrados en la figura siguiente.

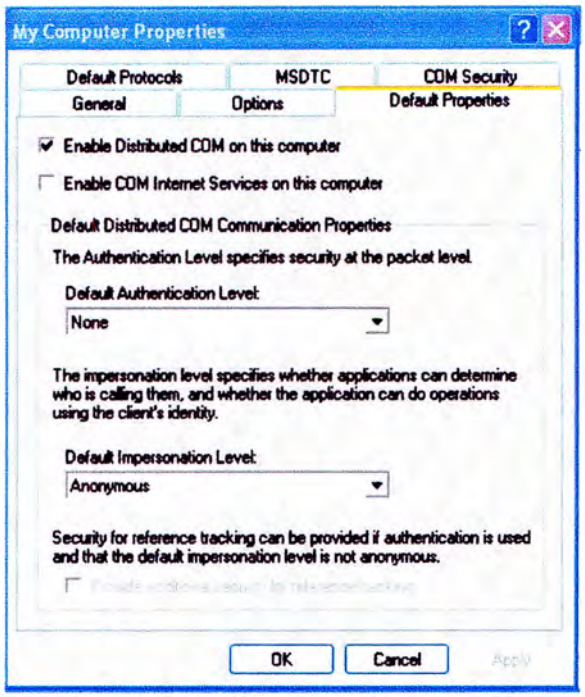


Figura7.1.4.1 b Configuración de las propiedades preestablecidas para un grupo de trabajo.

Pestaña "Default Security"

Aquí se pueden especificar los derechos para operación DCOM. Estas propiedades son utilizadas por todos los objetos COM que no cuentan con sus propias configuraciones.

Para que la comisión de DCOM sea sencilla se debe seguir la siguiente configuración para los usuarios "EVERYONE", "INTERACTIVE", "NETWORK" y "SYSTEM" tanto en el cliente como en el servidor:

- Access permissions: (Allow access)
- Launch permissions: (Allow launch)
- Configuration permissions: (Full control)

Configuración DCOM/ pestaña "Applications"

En Windows XP se debe abrir primero el folder "DCOM Configuration" en "My Computer" para desplegar todos los objetos COM disponibles en la computadora.

En esta sección se selecciona el objeto COM que se quiere configurar y se abre la ventana de configuración al hacer clic en el botón de "propiedades".

La configuración DCOM para la computadora del cliente OPC difiere de la del servidor OPC. EL servidor OPC para SIMATIC NET se lista como "OPC.SIMATICNET".

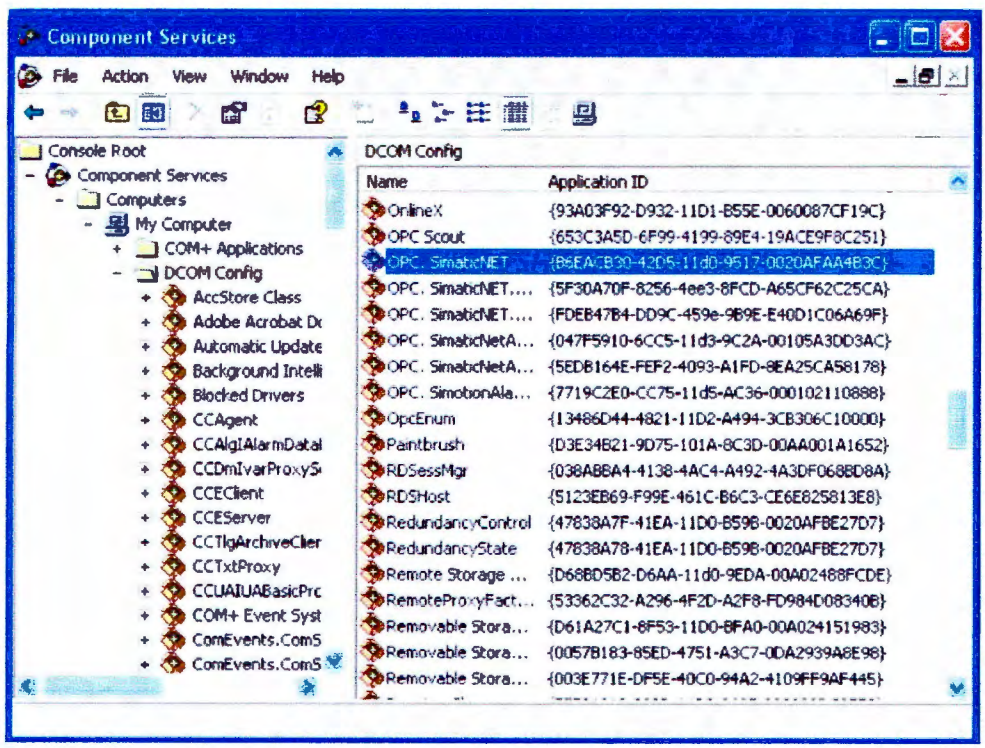


Figura 7.1.4.1 c. OPC.SimaticNET

Para registrar el servidor OPC en la computadora cliente se debe especificar su ubicación en la petaña "Location. Para la operación DCOM se debe marcar la casilla "Run application on the following computer". Posteriormente se da clic en "Browse" para seleccionar a la computadora servidor.

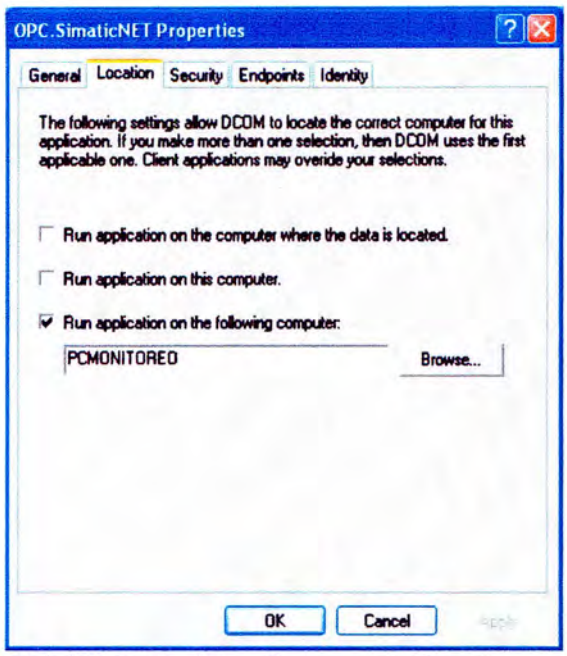


Figura 7.1.4.1 d. Registro de computadora de monitoreo

El OPC Scout es un cliente OPC que se encuentra en DCOM como un objeto registrado. Si se despliegan las propiedades de la aplicación DCOM del OPC Scout, "None" es desplegado como nivel de autenticidad. Debido a esto se asume un sistema abierto.

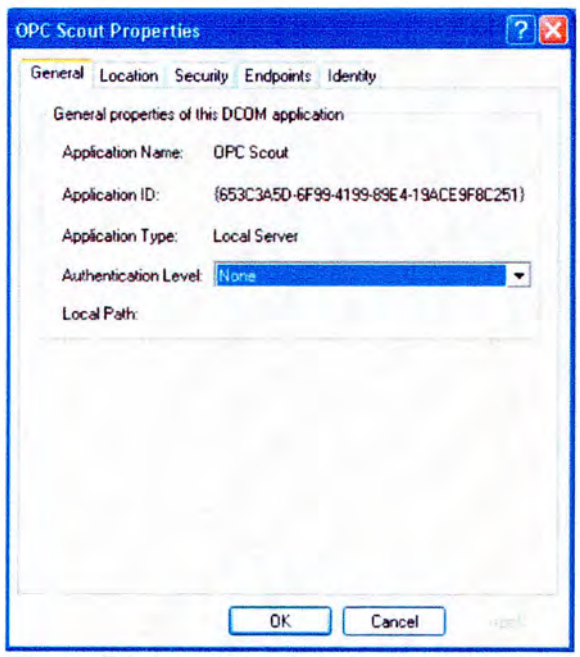


Figura 7.1.4.1 e. Propiedades de OPC Scout

Pestaña "Default Protocols"

En esta se especifican los protocolos de red que están disponibles para DCOM. El servidor OPC para SIMATIC NET fue probado con el protocolo "Connection-oriented TCP/IP".

Ya que el orden de los protocolos en la ventana decide su prioridad, DCOM usa el protocolo que se encuentra hasta arriba de la lista si se encuentra disponible. Para operar el servidor OPC PROFInet el protocolo "Connection-oriented TCP/IP" debe ser el primero de la lista.

7.1.4.2 Configuración en el servidor

En la PC en la que se opera el servidor OPC, se deben establecer las cuentas que tienen el permiso de utilizar el servidor. Además del permiso especial en conjunción con el OPC, la cuenta del usuario del servidor OPC debe tener permisos de usuario también.

Se debe seleccionar la aplicación y dar clic en el botón de propiedades.

Pestaña "General": Registrando al servidor OPC

Con el servidor OPC, las características default asumen un sistema abierto y asumen también que el usuario adaptará las características cuando sea necesario aumentar la seguridad.

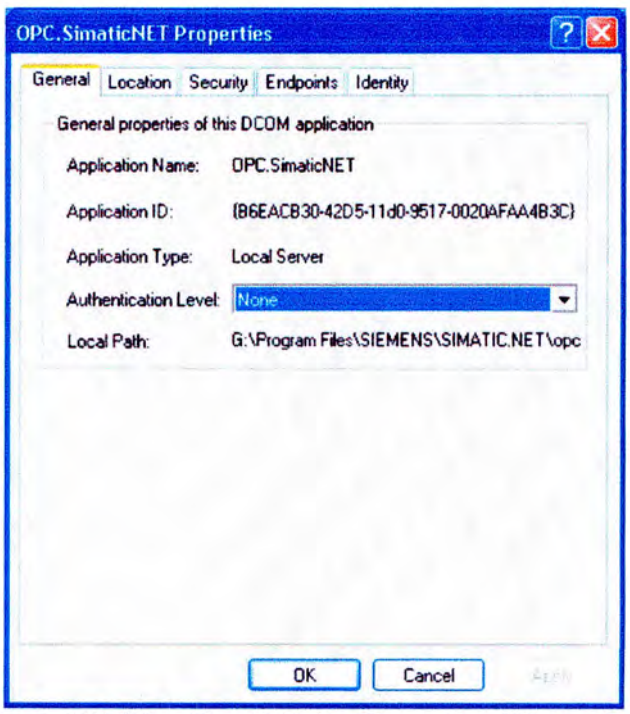


Figura 7.1.4.2 a. Propiedades Generales del servidor

Pestaña "Location"

Esta pestaña se utilize para especificar la computadora en la que el servidor inicia. Es por eso que para el servidor se debe seleccionar la opción "Run application on this computer".

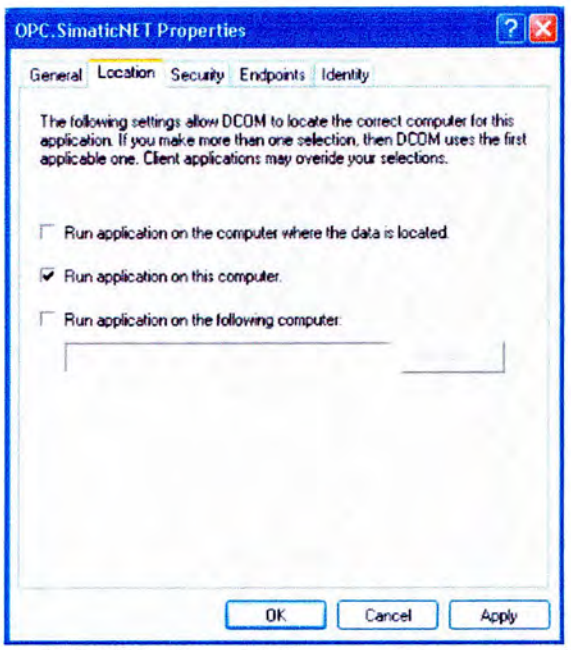


Figura 7.1.4.2 b. Se establece que éste es el servidor

Pestaña "Identity"

Las opciones especificadas aquí dicen que cuenta va a ser accesada para checar los permisos del usuario del objeto. Aquí es possible seleccionar varias opciones:

- "Interactive User"
- "The Launching User"
- "This User"

"Interactive User" se utilize cuando la cuenta del usuario que se encuentra actualmente conectado es utilizada. Si no hay usuario que utilice la computadora, no hay usuario interactivo y el objeto COM no puede ser creado. Si se opera sin un usuario conectado, se debe usar la opción "This User". "Interactive user" es la opción de default para el servidor OPC de SIMATIC NET.

"The Launching User" significa que la cuenta del usuario que corrió el cliente OPC se usa. Este usuario debe también contra con los permisos necesarios; en otras palabras, debe ser tecleado en la pestaña "Security". Este modo no debe ser utilizado con el

servidor OPC para SIMATIC NET ya que el servidor OPC sería iniciado más de una vez por diferentes cuentas de usuarios y esto no está permitido.

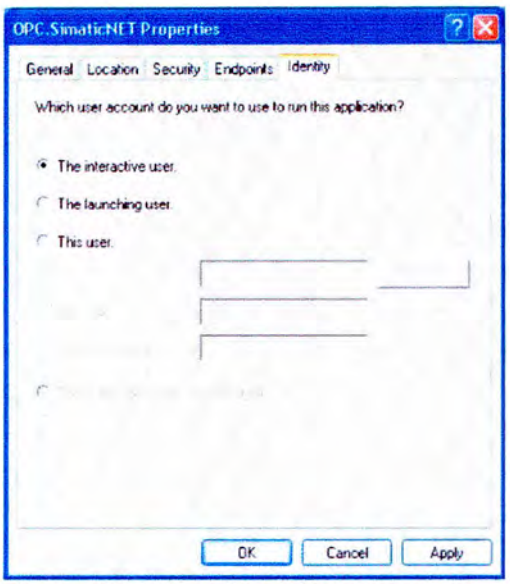


Figura 7.1.4.2 c. Identidad del servidor

Pestaña "Security"

Se especifica el permiso de acceso para el servidor OPC. Para los tres aspectos relevantes a objetos COM, se pueden usar ya sea los permisos de default o permisos definidos por el usuario para el objeto COM seleccionado.

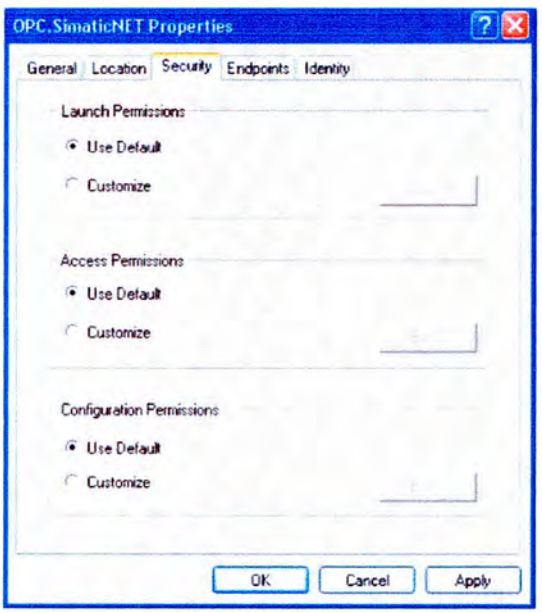


Figura 7.1.4.2 d. Seguridad

7.1.5 SCADA con Graphic Designer

El objeto gráfico ofrece la posibilidad de insertar en una imagen los gráficos creados con otros programas. Se pueden insertar gráficos o imágenes de los siguientes formatos: BMP, DIB, ICO, CUR, EMF, WMF, GIF y JPG.

El tamaño y las propiedades que adopta un objeto gráfico en Runtime se definen en el Graphics Designer.

Pasos a seguir para insertar un objeto gráfico:

1. Abrir la imagen en que se quiera insertar un objeto gráfico.
2. Dar clic en la paleta de objetos sobre el objeto Smart "Objeto gráfico".
3. Colocar el puntero del ratón en el punto de la imagen en la que se quiera insertar un objeto gráfico.
4. Arrastrar el objeto gráfico manteniendo pulsado el botón de ratón al tamaño deseado.
5. Abre el cuadro de diálogo "Configuración de objeto gráfico".
6. Seleccionar la imagen que se debe visualizar en el objeto gráfico.
7. Confirmar la entrada con "Aceptar".

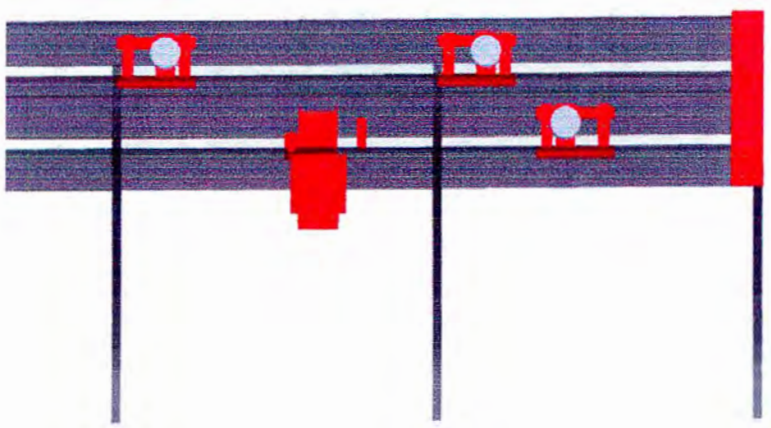
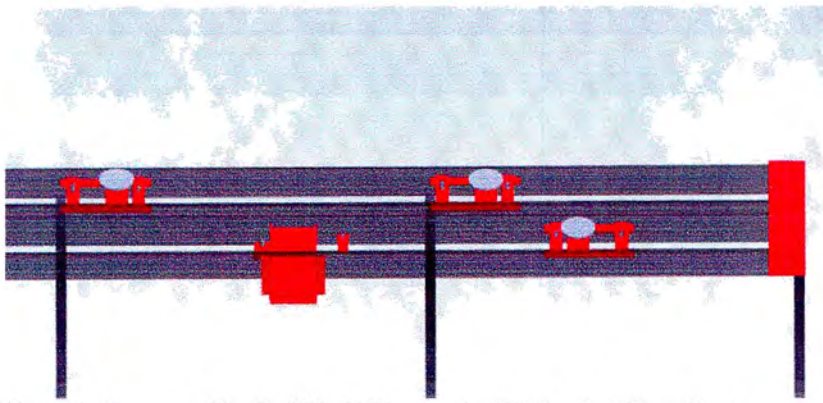


Figura 7.1.5. a. Vista de la pantalla de WinCC para la Celda de Manufactura con pistones retraídos



7.1.5. b. Vista de la pantalla de WinCC para la Celda de Manufactura con pistones activados

Para poder accionar los pistones de manera manual a través de la pantalla de la computadora de monitoreo debe seguirse una configuración en los pistones de la Celda por medio del Graphic Designer de WinCC. Para cada pistón existen dos etapas, una en la que el pistón se encuentra retraído y otra en la que se encuentra expandido.

El objetivo es que al dar clic sobre un pistón retraído el mismo se expanda y viceversa. También se busca que cuando un pistón se encuentra expandido podamos ver la imagen del mismo afuera y cuando está retraído lo veamos dentro.

Los pasos a seguir para establecer el accionamiento de un pistón a partir de un clic sobre su imagen en la pantalla se enlistan a continuación:

1. Se debe dar clic derecho sobre el objeto (un pistón) y seleccionar la opción de "Propiedades".
2. En la ventana que aparece a continuación se selecciona la pestaña de Eventos y se da clic sobre el evento Ratón dentro de la lista de Objeto Gráfico.
3. Se da doble clic sobre "Clic en ratón" y aparece una nueva ventana. En esta debe programarse el Origen y el Destino, en Origen se selecciona un valor constante de 0 o 1 dependiendo de el valor que se quiera dar a la variable (falso o verdadero) y en el Destino se selecciona la opción "Variable" y se da clic sobre el icono amarillo que aparece a la derecha, éste conduce a una lista de variables, que son las variables de Step 7. En esta lista se selecciona la variable que corresponde al pistón que seleccionamos.
4. Se da clic en Aceptar.

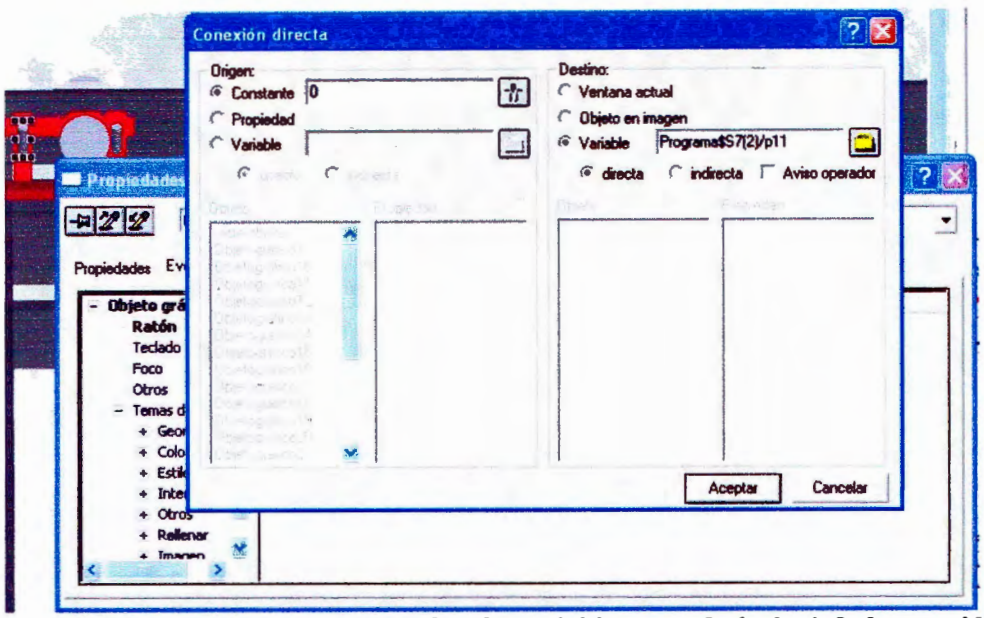


Figura 7.1.5. c Establecimiento de valor de variable para el pistón 1 de la estación 1 mientras está expandido

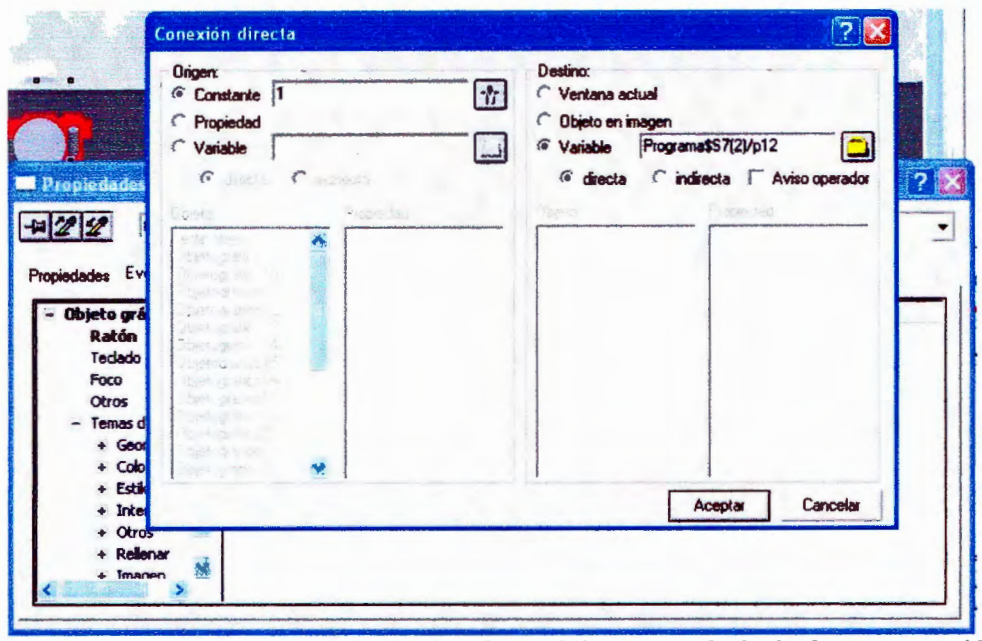


Figura 7.1.5. d Establecimiento de valor de variable para el pistón 2 de la estación 1 mientras está retraído

Los pasos a seguir para establecer la visualización de un pistón a partir de un clic sobre su imagen en la pantalla se enlistan a continuación:

1. Se debe dar clic derecho sobre el objeto (un pistón) y seleccionar la opción de "Propiedades".
2. En la ventana que aparece a continuación se selecciona la pestaña de Propiedades y se da clic sobre "Otros" dentro de la lista de Objeto Gráfico.
3. Se da doble clic sobre "Visualización" y aparece una nueva ventana. En esta debe programarse la visibilidad de la variable de "Expresión/Fórmula" al dar clic sobre el icono con puntos suspensivos que aparece a la derecha, éste conduce a una lista de variables, que son las variables de Step 7. En esta lista se selecciona la variable que corresponde al pistón que seleccionamos.
4. Posteriormente se elige la opción Booleano en "Tipo de datos" y se escribe si se quiere que el objeto pueda verse cuando la variable tiene valor de Verdadero o Falso.

Por ejemplo, si el pistón se encuentra expandido entonces la variable de ese pistón tiene valor de 1 (verdadero) y queremos que esa imagen pueda verse. En cambio, cuando el pistón está retraído la imagen del pistón expandido no debe verse, esto es cuando la variable tiene valor de 0 (falso).

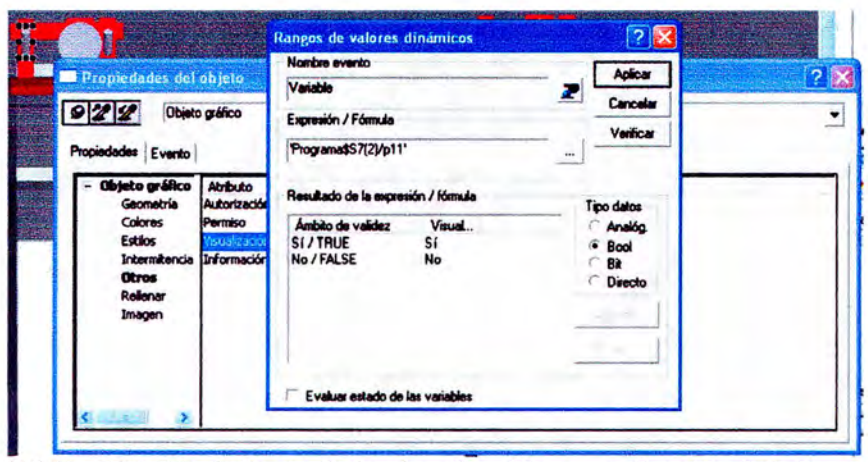


Figura 7.1.5. e Establecimiento de la visualización para el pistón 1 de la estación 1 expandido.

7.1.6 Configuración y programación del Panel de control HMI

7.1.6.1 Configuración del Panel en STEP 7

Primero se establece la configuración de red vía Herramientas>Configuración de redes.

Cuando aparece la ventana con las redes existentes se busca el panel en la carpeta Equipo que se encuentra en la lista de la derecha. Se escoge estación HMI Simatic y se arrastra en comunicación vía profibus.

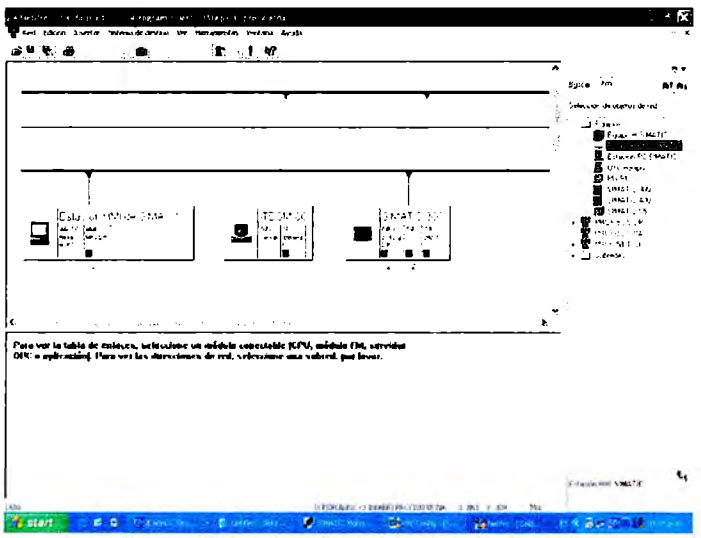


Figura 7.1.6.1. a Estableciendo la red.

Posteriormente, en la parte de Hardware se arrastra el objeto MP270 Keys de la carpeta Simatic HMI Station hacia el cable de Profibus.

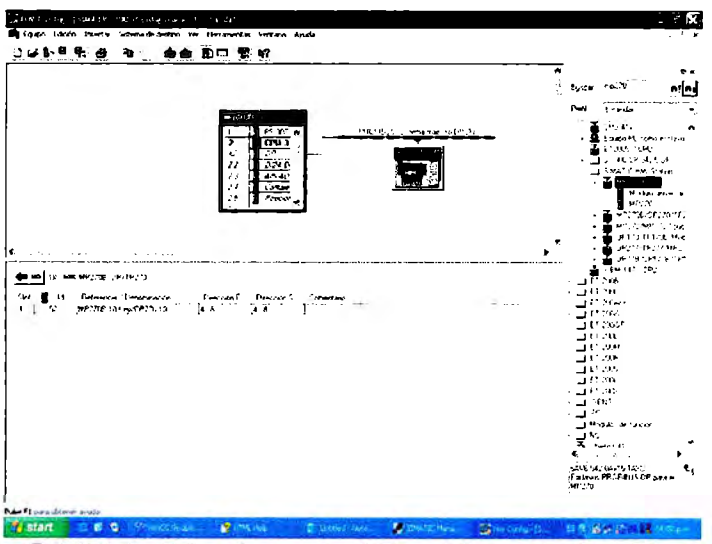


Figura 7.1.6.1. b. Seleccionando el Panel de control en Hardware.

Se da doble clic sobre el panel que aparece en la ventana y se abre una nueva ventana de Propiedades, se selecciona la pestaña Acoplamiento y se da clic en el botón Acoplar y después en Aceptar para cerrar la ventana.

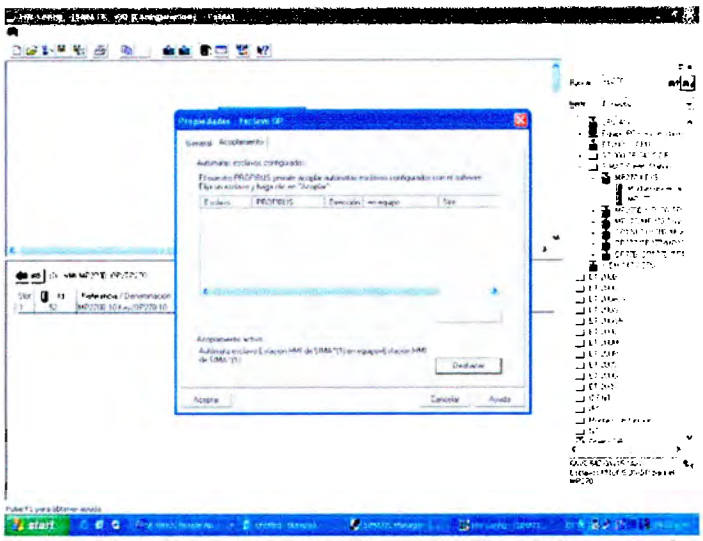


Figura 7.1.6.1. c. Acoplamiento del Panel de control.

7.1.6.2 Programación del Panel de control

El objetivo de la programación del Panel es mostrar en el mismo 5 imágenes de diferentes secciones de la Celda de Manufactura dependiendo de lo que el usuario desee ver en ese momento. Esas imágenes representan:

- La vista principal de la celda
- La estación 1
- La estación 2
- La estación 3
- La estación 4

Para cada estación se genera entonces un "close-up" en el que se pueden ver sus componentes y existe la opción de seleccionar alguno de ellos de modo que se cambie la posición que tiene. Es decir, al seleccionar un pistón de alguna estación, si el pistón se encuentra expandido se retrae y viceversa.

Los pasos para la programación del Panel por medio de WinCC flexible se explican a continuación:

1. En la ventana de la izquierda, llamada Proyecto, se selecciona la carpeta imagen y se da clic sobre el ícono "Agregar imagen". Al aparecer la plantilla se da clic derecho para cambiar su nombre. En este caso la primera imagen con que se trabaja es "Principal".

2. Sobre el área de trabajo de "Principal" se inserta el dibujo de la celda de manufactura, en este caso es el mismo que se insertó en el Graphic Designer. En la imagen que se

inserta se agrega para cada estación un texto con el botón del panel que la activa. Posteriormente se agregan los botones sobre la imagen.

3. Para cada estación se crea un dibujo equivalente a un "close-up" de la misma y se realiza el mismo procedimiento que para la imagen "Principal". Se agrega la imagen para cada estación y se nombran Módulos del 1 al 4. Posteriormente en el área de trabajo de cada Módulo se inserta el dibujo con las leyendas de los botones que activaran cada pistón.

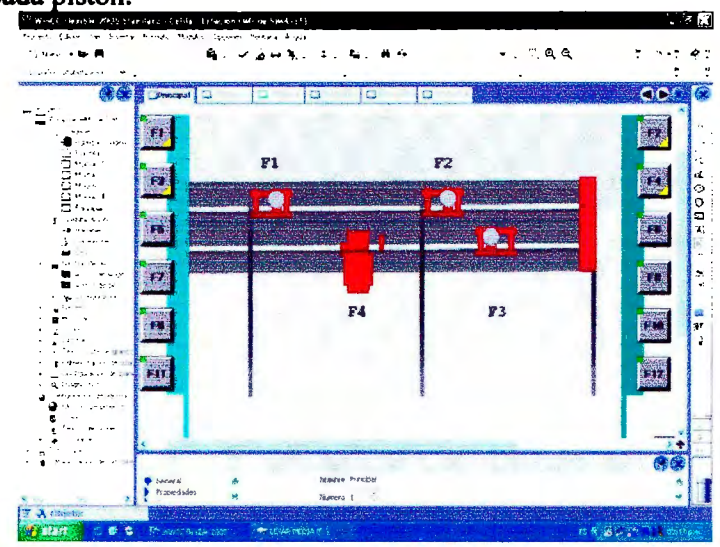


Figura 7.1.6.2. a. Imagen principal del Panel

4. Para cada uno de los botones que representan una estación de la celda se establece la configuración de la imagen que se va a desplegar. En el caso de F1 se activará el Modulo 1. Se da clic sobre el botón F1, al aparecer en la ventana inferior para "softkey F1" las opciones se da clic en Eventos y después en Pulsar, en la ventana de la derecha se selecciona la opción ActivarImagen y se da en Nombre de la imagen "Modulo 1".

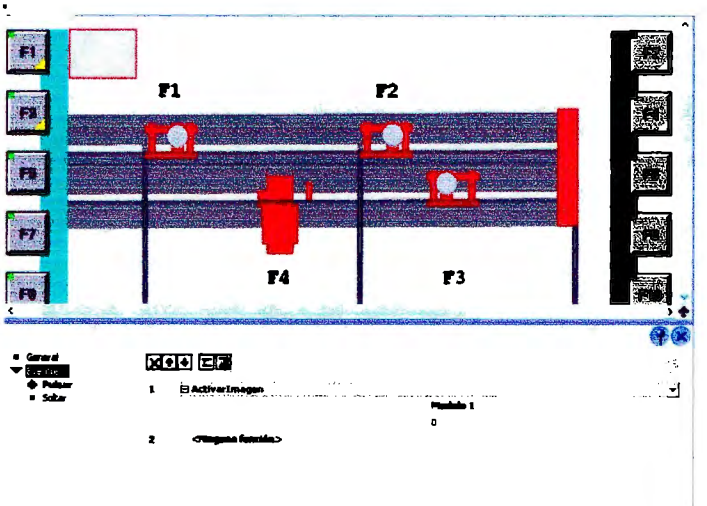


Figura 7.1.6.2. b. Selección de estación 1 con el botón F1

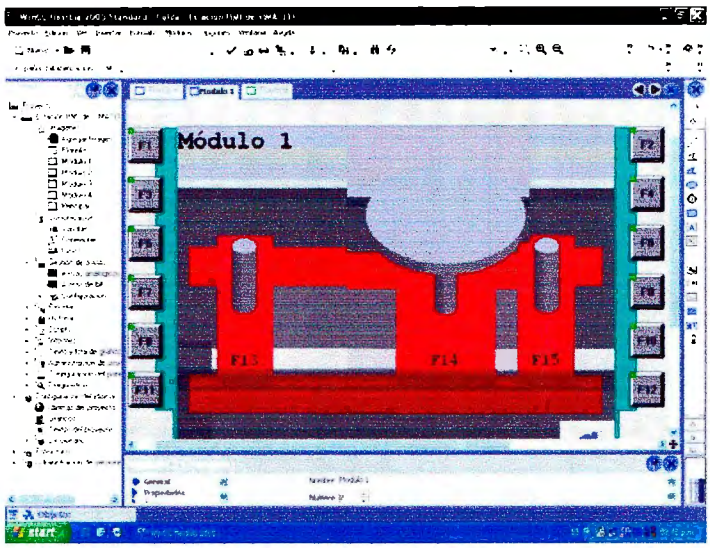


Figura 7.1.6.2. c. Imagen Módulo 1 y sus componentes

5. Para la imagen "Modulo 1" se configura cada uno de los botones que activa a los pistones.

6. Para el pistón 1 se da clic sobre F13, en la ventana de "Softkey 13" se da clic en Eventos, después en Pulsar y en la ventana de la derecha se selecciona la opción InvertirBit. Se escribe la variable p11 (pistón 1 de la estación 1) en la fila Variable (Entrada/Salida).

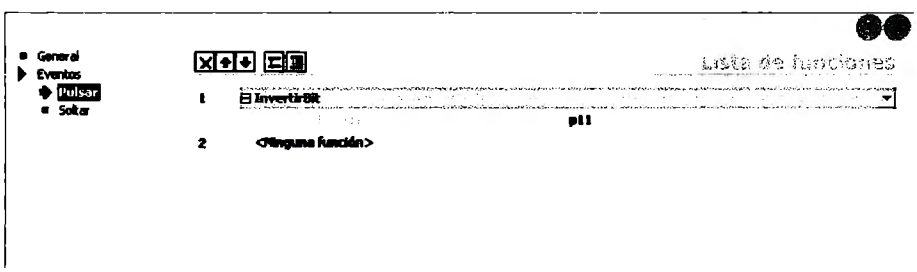


Figura 7.1.6.2. d. Selección del pistón 1 con el botón F13

7. Una vez realizada la opción que se desea en un módulo se puede querer volver a la imagen "Principal". Esta se establece con el botón F20. Se da clic sobre F20 y en la ventana de "Softkey 20" se da clic en Eventos, después en Pulsar y en la ventana de la derecha se selecciona la opción ActivarImagen. Se da en Nombre de la imagen "Principal".

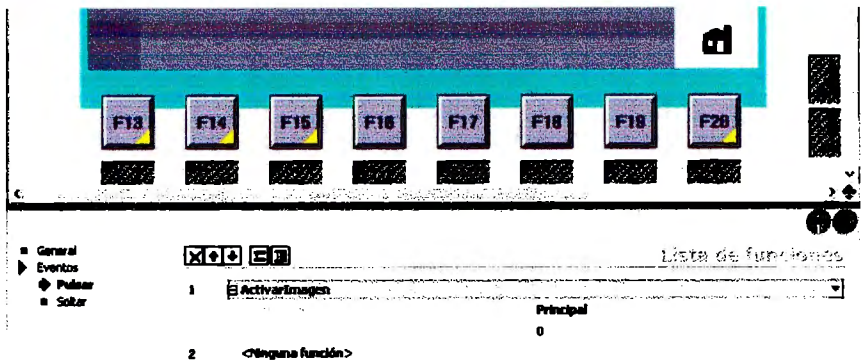


Figura 7.1.6.2. e. Configurando la opción Home

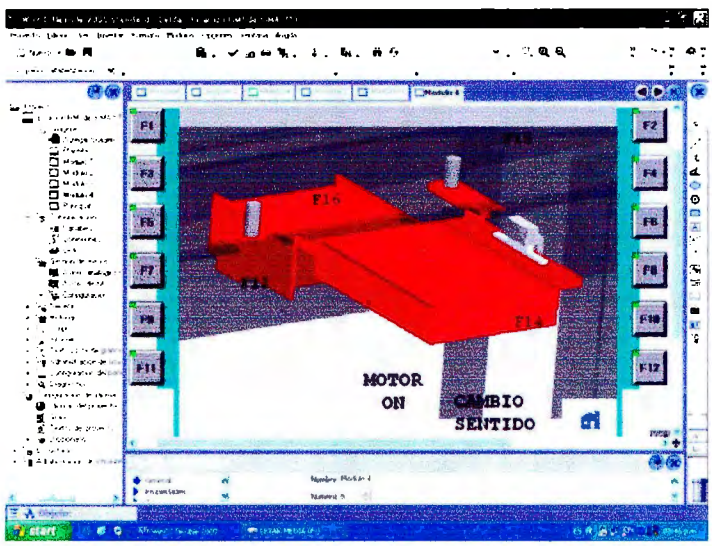


Figura 7.1.6.2. f. Imagen Módulo 4 y sus componentes

7.1.7 ¿Cómo programar los robots?

Existe un panel de trabajo en el cual se crea la secuencia que el robot va a seguir, eligiendo los puntos a los que irá y los procesos del mismo. Sobre este panel de trabajo siempre que se crea un nuevo archivo aparece el icono de "Inicio" ya que al correr una aplicación el robot debe ir a su posición de inicialización para después seguir una secuencia. Las operaciones se unen por medio de flechas que establecerán la secuencia de los mismos. Un panel ubicado a la derecha de la ventana de usuario es el que despliega los puntos existentes.

El panel de trabajo incluye operaciones comunes de Windows tales como el arrastrado de un objeto dentro del área de trabajo, copiar un objeto por medio del arrastrado y la tecla CTRL, eliminar objetos con la tecla Suprimir, aceptar una operación con el botón Enter y cancelar una operación con la tecla Esc.

Para el brazo robótico Puma los puntos se deben crear a partir de la ventana de usuario, por medio de un botón llamado "Crear punto", por medio de el cual se puede enumerar a los diferentes puntos que se vayan creando para después bajarlos, de forma que éstos aparezcan en pantalla para su uso. Para el brazo robótico Mitsubishi los puntos pueden crearse directamente desde el controlador del robot y después en la ventana de usuario se bajan los puntos para trabajar con los mismos.

7.1.7.1 Descripción del botones del programa

Manejo de archivos

Nuevo programa 	Abre un nuevo panel de trabajo en el que en un principio solo aparece el botón de Inicio
Abrir programa 	Permite al usuario abrir un archivo .pan o .pum y desplegarlo en el panel de trabajo.
Guardar programa 	Crea un archivo con la extensión .mit para el Mitsubishi o .pum para el Puma. Al presionar este botón aparece la venta de Windows de "Guardar como" y se debe proceder a elegir la carpeta para guardar y escribir el nombre con el que se desee guardar el archivo. Si se requiere guardar cambios hechos a un archivo se debe sobrescribir en el archivo ya existente.

Controles

Inicio 	
Mover a Punto 	Permite al robot ir a uno de los puntos existentes, al dar doble clic sobre el icono de Punto en el Panel de trabajo aparece una ventana que permite teclear puntos entre el 1 y el 200.
Abrir Gripper 	Como su nombre lo indica, manda al robot la instrucción de abrir su gripper y mantenerlo así hasta nuevo aviso.

Cerrar Gripper 	Hace que el robot cierre el gripper.
Retardo 	Permite crear un retardo en el programa de 1 a 999 segundos. Para elegir el retardo que se desee se debe dar doble clic sobre el icono que aparece en el Panel de trabajo y en la ventana que aparece teclear el número deseado. Existe un retardo por default de 3 segundos.
Velocidad 	Define una velocidad de acuerdo al parámetro del robot en un rango de 1 a 9. Para elegir la velocidad se realiza el mismo procedimiento que para el Retardo. La velocidad de es 5 por default.
Home 	Indica al robot que debe moverse a su posición inicial, que también se conoce como "nest".

Controles de señal

Señal de entrada 	Es mandada por el PLC para indicar que una pieza puede empezar a trabajarse porque ha llegado a la estación.
Señal de salida 	Es mandada por el robot para indicar que una pieza ha sido trabajada y se encuentra ahora en el palet para continuar su camino por la banda.

Controles de flujo

Esperar por señal 	Espera por una señal para continuar con el proceso.
Partir programa 	Si se tienen dos "subsecuencias" dentro de una secuencia principal, este botón manda la secuencia principal del robot a una subsecuencia o a otra dependiendo de la condición que se haya cumplido
Unir programa 	Une una bifurcación de secuencias, es decir, es el punto en el que dos subsecuencias se unen para seguir ejecutando la secuencia principal.

Manejo de puntos

Bajar Puntos 	Permite al usuario comunicarse con el robot para almacenar en un archivo las coordenadas de los puntos que tiene en la memoria.
Subir Puntos 	Permite al usuario subir puntos ya guardados a la memoria.
Borrar Puntos 	Borra los puntos existentes.

El objetivo de esta barra de herramientas es el de permitir al usuario descartar los puntos que no son necesarios. Si el robot se usa para diferentes tareas se recomienda borrar los puntos para evitar confusiones.

Reproducción

Ejecutar desde inicio 	Correr todo el programa de principio a fin.
Desde posición actual 	Corre el programa a partir de un paso seleccionado hasta el fin.
Solo paso seleccionado 	Corre solo el paso seleccionado
Paro 	Se activa únicamente una vez que el programa está corriendo, sirve para emergencias.

7.1.7.2 Secuencia general para un programa del robot Mitsubishi:

- ✓ Se abre un nuevo programa
- ✓ Se revisa si hay puntos guardados con el botón Bajar Puntos. En caso de haberlos se despliegan en el panel derecho.
  - ➢ Si hay puntos y éstos no sirven para el programa en turno, se deben borrar con el botón Borrar Puntos.
  - ➢ Posteriormente se programan los puntos necesarios con el control del robot y se presiona el botón Bajar Puntos.
- ✓ Se crea la secuencia necesaria, estableciendo velocidades, puntos, retardos (de ser necesarios) y señales de espera o de fin de secuencia para el PLC.
- ✓ Se presiona el botón Ejecutar para correr el programa.
- ✓ Si se desea se puede guardar el programa presionando el botón Guardar.

7.1.7.3 Secuencia general para un programa del robot Puma:

- ✓ Se abre un nuevo programa
- ✓ Se revisa si hay puntos guardados con el botón Bajar Puntos. En caso de haberlos se despliegan en el panel derecho.
  - ➢ Si hay puntos y éstos no sirven para el programa en turno, se deben borrar con el botón Borrar Puntos.
  - ➢ Posteriormente se programan los puntos necesarios con el control del robot y se crean en la ventana de usuario presionando el botón Crear Puntos. Al aparecer la ventana de puntos a crear se da el número de punto deseado y se presiona el botón Bajar Puntos.
- ✓ Se crea la secuencia necesaria, estableciendo velocidades, puntos, retardos (de ser necesarios) y señales de espera o de fin de secuencia para el PLC.
- ✓ Se presiona el botón Ejecutar para correr el programa.
- ✓ Si se desea se puede guardar el programa presionando el botón Guardar.