PROGRAMACIÓN DE UN ROBOT CARTESIANO

TIPO PÓRTICO

BRAYAN ELIHU REY ACEVEDO

UNIVERSIDAD PONTIFICIA BOLIVARIANA

FACULTAD DE INGENIERIA

ESCUELA DE INGENIERÍA MECÁNICA

BUCARAMANGA

2013

PROGRAMACIÓN DE UN ROBOT CARTESIANO

TIPO PÓRTICO

BRAYAN ELIHU REY ACEVEDO

PROYECTO DE GRADO

EDWIN CÓRDOBA TUTA
DIRECTOR

UNIVERSIDAD PONTIFICIA BOLIVARIANA
FACULTAD DE INGENIERIA
ESCUELA DE INGENIERÍA MECÁNICA
BUCARAMANGA
2013

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

_____

_____

Firma de Presidente del Jurado

_____

Firma del Jurado

_____

Firma del Jurado

Bucaramanga, 13 de septiembre de 2013

CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

RESUMEN GENERAL DE TRABAJO DE GRADO

TÍTULO: **PROGRAMACIÓN DE UN ROBOT CARTESIANO TIPO PÓRTICO**

AUTOR: **BRAYAN ELIHU REY ACEVEDO**

FACULTAD: **FACULTAD DE INGENIERÍA MECÁNICA**

DIRECTOR: **EDWIN CÓRDOBA TUTA**

RESUMEN

Con este proyecto se implementa el código y configuración de un robot cartesiano tipo pórtico, parte de una línea de producción orientada en el desarrollo, selección y empaque de piezas de diferentes alturas. Para que el robot cartesiano cumpla con las secuencias y funciones, como los modos de operación y la transferencia de datos de éste a la empacadora, la calle de selección y la base de datos, se desarrollaron rutinas, sub rutinas y configuraciones para cada elemento requerido en el sistema cartesiano. Estas fueron realizadas mediante el software Step 7, Step7 Microwin y WinCC Flexible. Este proyecto será una herramienta útil para fortalecer el conocimiento de los alumnos de ingeniería mecánica, electrónica e industrial en el área de automatización debido a la documentación resultante, en donde se explica con profundidad la creación del código y configuración, este sistema cartesiano también implementa una línea de producción la cual será usada como herramienta para reforzar el conocimiento práctico de los estudiantes. El robot cartesiano es comandado por un PLC siemens S7-300 con CPU 315F-2PN/DP 6ES7 315-2FH13-0AB0 y modulo digital SM 323 16DI/16DO, el cual se comunica con dos tarjetas de control CMMS-ST-C8-7 para motores paso a paso, una tarjeta de control SEC-AC-305-PB-P01, un

variador de frecuencia Micro master 440, un módulo de comunicación profibus EM277 para PLC S7-200 y una CPU 315F-2PN/DP 6ES7 315-2FJ14-0AB0 la cual comanda la empacadora. Toda la comunicación que existe entre los dispositivos se realizó mediante los protocolos de comunicación Profibus y Ethernet.

PALABRAS CLAVES: Robot cartesiano, PLC, Ethernet, Profibus, Automatización.

V° B° DIRECTOR DE TRABAJO DE GRADO

GENERAL SUMMARY OF WORK OF GRADE

TITLE: PROGRAMMING A ROBOT CARTESIANO PORTICO TYPE

AUTHOR(S): BRAYAN ELIHU REY ACEVEDO

FACULTY: FACULTAD DE INGENIERÍA MECÁNICA

DIRECTOR: EDWIN CÓRDOBA TUTA

ABSTRACT

The aim of this proyect is to create a code and to configurate a cartesian robot that is part of a production line to select and pack some pieces with different high. There are routines and subroutines generated in order to allow the robot to do the job expected, it is to work in diferent operation modes, and to share data between the selection machine and the packages machine it was made in STEP 7, STEP 7 MICROWIN and WINCC FLEXIBLE.

Also this proyect will be an important tool for the mechanical, electronic and industrial engineering students when they take the automatization subject, because there is documentation where there is explained the creation and configuration of the code in the same way a production line is used to make easy the learning process in the students.

The robot is controled by a PLC Siemens S7-300 with CPU 315F-2PN/DN 6ES7 315-2FH13-0AB0 and a digital module SM 323 16 DI/16DO, it comunicates in to drivers CMMS-ST-C8-/ to engines. A driver SEC-AC-305-PB-P01, a variable frequency drive Micro Master 440, a profibus EM 277 for the PLC S7-200 and a CPU 315F-2PN/DP 6ES7 315-2Fj14-0AB0 the last one is to control the package machine; the comunication between the devices were made in Profibus and Ethernet.

KEYWORDS: Robot Cartesian, PLC, Ethernet, Profibus, Automation.

V° B° DIRECTOR OF GRADUATE WORK

INTRODUCCIÓN

Actualmente el sector industrial requiere sistemas de producción mucho más rápidos y eficientes, debido al consumismo y la competencia del medio. Tal necesidad, se suple mediante ingenieros capacitados para manipular, diseñar y crear sistemas de producción que cubran la demanda actual.

Por lo anterior es indispensable que los estudiantes de ingeniería industrial y afines, interactúen con sistemas de producción reales y así adquieran durante su proceso de formación académicas experiencias útiles para su vida laboral. Con este fin las facultades de INGENIERÍA INDUSTRIAL, MECANICA Y ELECTRONICA DE LA UNIVERSIDAD PONTIFICIA BOLIVARIANA han ido desarrollando proyectos como la CALLE DE SELECCIÓN, LA EMPACADORA, y un sistema robótico (ROBOT CARTESIANO TIPO PORTICO) el cual inicialmente se programó para recrear parte del proceso de ensamble de un vehículo de juguete, previamente seleccionado para una línea de producción del mismo.

En la actualidad esta idea se ha reestructurado para llevar a cabo una línea de producción orientada en el desarrollo, selección y empaque de piezas de diferentes alturas. De tal forma, la programación y configuración inicial del robot no es completamente útil para la nueva línea de producción, debido al cambio de tareas que debe efectuar el robot cartesiano en este nuevo proyecto.

Para el desarrollo de las tareas del robot se programó y configuro los dispositivos de control y comunicación requeridos. En la programación, se crearon rutinas y subrutinas con las cuales se ejecutan los movimientos y enlaces desde el PLC maestro con los dispositivos esclavos. En la configuración, se definieron los elementos conectados y protocolos de comunicación, utilizados para intercambiar los datos. Durante el desarrollo del proyecto se adquirió los conocimientos necesarios para programar y configurar el sistema robótico. Los cuales quedan

plasmados en este documento, de tal manera sirvan de base para modificaciones futuras.

OBJETIVOS

OBJETIVO GENERAL:

Realizar la programación del robot cartesiano y la configuración de los elementos en comunicación, mediante el uso del software Step 7, Step7 Microwin y WinCC Flexible; para cerrar la línea de producción que se implementará en el laboratorio de automatización.

OBJETIVOS ESPECÍFICOS:

- Realizar la documentación sobre la programación y configuración de los dispositivos, mediante manuales de usuario y manuales de programación. Resultado: Documento con resumen de la programación y configuración. Indicador: Con el documento se debe ser capaz de programar y configurar el robot cartesiano y los elementos con que se comunica.

- Crear el código, mediante el software Step 7, usando el lenguaje de escritura avanzada. Resultado: Código. Indicador: El código debe ser creado en AWL, con Step 7.

- Configurar y programar los dispositivos conectados en la red Profibus y Ethernet, mediante el software WinCC Flexible, Step 7 microwin y el módulo hardware en Step 7. Resultado: Dispositivos comunicando. Indicador: La comunicación debe ser mediante los protocolos, Profibus y Ethernet.

- Realizar pruebas al sistema. Resultado: Documento con procedimiento para puesta en marcha. Indicador: Con el documento se debe ser capaz de simular todas las secuencias y funciones del robot cartesiano.

1. GENERALIDADES

Hoy en día la industria se hace más grande, generando así la demanda de sistemas de producción mucho más rápidos y precisos. Para cubrir tal necesidad, una de las soluciones generadas, ha sido el uso del robot. El robot industrial se puede desempeñar en sectores como montaje, embalaje, plástico, electrónica entre otros [1].

Los robots pueden ser clasificados de varias formas; según geometría del brazo, aplicación, fuente de potencia y sistema de control.

Según la aplicación [2]:

Manipulación de materiales: básicamente consiste en desplazar materiales o piezas de un lugar a otro. Esto puede ser usado para distribuir piezas de diferentes características en puntos establecidos (clasificación). También es muy usado para la carga o des carga de una máquina, es decir el robot carga la pieza de trabajo en la maquina o descarga la pieza acabada de esta.

Procesos: en esta aplicación el robot realiza un trabajo sobre la pieza, necesitando así el robot operar mediante una herramienta y no mediante una pinza. Esta aplicación incluye soldadura por puntos, por arco, pintura por pulverización, entre otros.

Montaje e inspección: en esta aplicación el robot genera un producto final, mediante el ensamble de las diferentes partes que conforman el producto o mide y calibra características de calidad del producto, utilizando sensores como herramienta.

Según el método de control [3]:

Robot no servo-controlado: este tipo de robot tiene un número fijo de posiciones en cada articulación, de finidas ya sea por topes o sensores como finales de carrera, de tal manera que se mueve solo para posicionarse en estos puntos y no

en alguna posición intermedia. Suelen ser controlados mediante neumática o hidráulica.

Robot servo-controlado: este tipo de robot tiene un sensor de posición lineal o angular en cada articulación. La señal del sensor de posición, es enviada a un controlador el cual procesa esta señal y define en donde será posicionado el robot, mediante un algoritmo establecido en el controlador que compara la señal del setpoint con la señal del sensor. De esta forma todo el rango de posiciones puede ser utilizado.

Robots servo-controlado punto a punto: a diferencia del sistema anterior, el controlador no establece el movimiento a el punto exacto de posicionamiento (setpoint), si no que calcula la trayectoria a partir del punto final e inicial, de manera que el robot se mueve a través de un vector calculado y no hacia un punto.

Según la geometría de su estructura [3], [4], [5]:

Cartesiano: El posicionamiento es llevado a cabo, mediante el movimiento sobre tres elementos deslizantes, perpendiculares entre sí. De tal forma que para posicionarse, las coordenadas cartesianas (x, y, z) deben ser definidas. Este tipo de robot es rápido, preciso, capaz de trabajar con altas cargas, su elemento terminal (herramienta) generalmente no es móvil y su estructura puede ser, tipo cantiléver o pórtico.

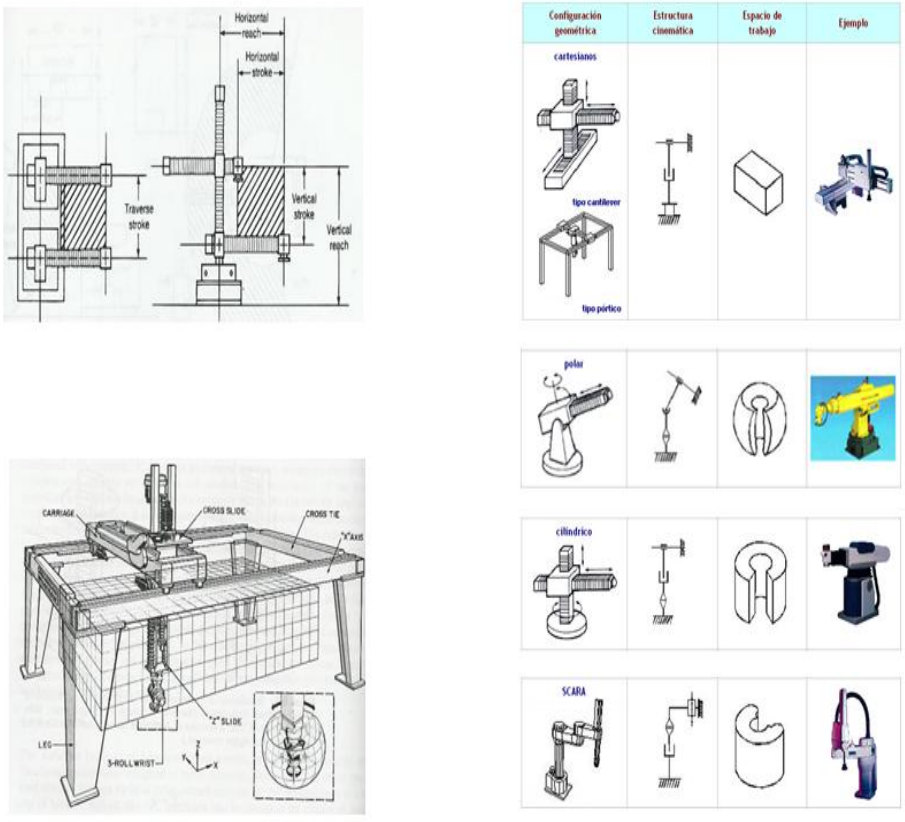
Cilíndrico: El posicionamiento es llevado a cabo, mediante tres articulaciones, una rotacional en la base (coordenada acimutal φ), una perpendicular a la base (coordenada en el eje z, define altura) y una perpendicular al eje z (define el radio R). De tal manera que para posicionarse, las coordenadas cilíndricas (r, z, φ) deben ser definidas.

El área de trabajo de este tipo de robot es equivalente al espacio entre dos cilindros concéntricos de la misma altura.

Polar o esférico: el posicionamiento es llevado a cabo, mediante tres articulaciones, dos rotacionales (Angulo polar θ y azimut φ) y una lineal (radio de la esfera R). De tal manera que para posicionarse, las coordenadas esféricas (r, θ, φ) deben ser definidas.

Scara: el posicionamiento es llevado a cabo, mediante tres articulaciones, dos angulares, que definen la posición en el plano XY y una lineal que define la posición en el eje Z. Es rápido, barato y preciso, pero solo puede acceder en áreas de trabajo que sean perpendiculares a su eje vertical. Se utiliza generalmente en operaciones de ensamble o inserción de componentes electrónicos. En la figura 1 se muestran las configuraciones geométricas de los robots anteriormente mencionados.

Figura 1: Clasificasion de robots industriales según su geometria.



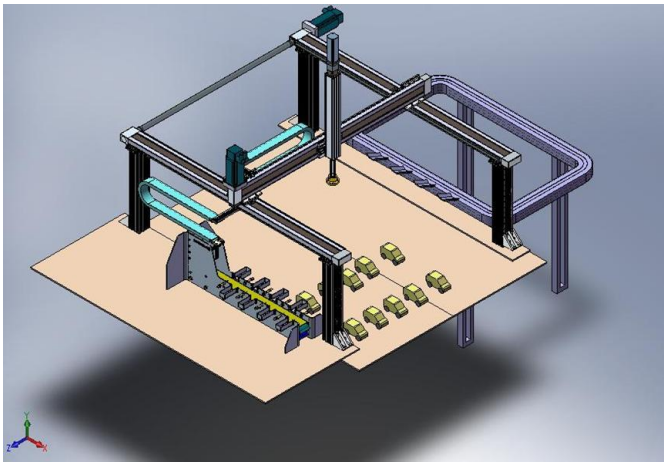
Fuente: [2], [5].

Hoy en dia los robots industriales estan siendo implementados en algunas universidades para poder reforsar a los estudiantes en el area de la automatizacion, por ejemplo en la universidad andres bello en chile, tienen un robot Fanuc LR Mate 200iC, el cual es robot tipo antropomorfo y la universidad autonoma del caribe en barranquilla colombia, tiene un robot cartesiano en el laboratorio de robotica [6], [7], [8].

El robot cartesiano en el laboratorio de automatizacion, se utilizara con el fin de cerrar la linea de produccion a implimentar, en esta el robot desempeñara la labor de transportar el numero de piezas especificadas en cada peticion de la empacadora. Esta linea de produccion tiene como objetivo ser un proceso practico donde se puedan analizar variables comunes de una linea de produccion; tal como las paradas del proceso y sus causas , tiempos de operario, tiempo de ciclo, entre otras. para tal analizis, se utilizara una base de datos en la cual todas las variables del proceso seran cargadas.

Inicialmente el Robot Cartesiano hacia parte de un macro proyecto de la Universidad Pontifica Bolivariana, el cual se decidió no realizar, de tal manera que la programación inicial tenía un objetivo diferente. Este proyecto básicamente consistió en la construcción de un proceso conformado por la calle de selección (la fase uno de este proyecto), el robot cartesiano (fase dos) y una calle de ensamblaje, que habría sido la fase tres. Al Macro Proyecto se le asignó el nombre CIM (Centro Integrado de Manufactura) el cual fue visualizado como se muestra en la figura 2.

Figura 2: Modelo inicial del proceso.



Fuente: [9].

Debido a el cambio de proceso fueron agregados varios elementos al robot. Inicialmente estaba compuesto por los sigientes dispositivos de control : CPU 315F-2PN/DP de referencia 6ES7 315-2FH13-0AB0, dos tarjetas electrónicas CMMS-ST-C8-7, para los ejes X-Y, una tarjeta SEC-AC-305-PB para el eje Z, un panel táctil OP177B PN-DP, un módulo de entradas y salidas DI16/DO16x24V/0.5A, de referencia 6ES7 323-1BL00-0AA0 y una fuente de poder PS 307 5A, de referencia 6ES7 307-1EA01-0AA0 [9].

En el nuevo sistema el robot necesita de tres dispositivos mas, estos son: una CPU 315F-2PN/DP de referencia 6ES7 315-2FJ14-0AB0 la cual es la que controla la empacadora, un variador de frecuencia Micro Master 440 de referencia 6SE6400-1PB00-0AA0. y un modulo de comunicación EM277, el cual se encarga de comunicar dos plc S7200 Y S7300.

2. METODOLOGIA

Para el desarrollo de este proyecto se llevaron acabo las siguientes tareas.

Tarea 1: Revisar manuales de usuario y documentos relacionados con la configuración y programación de los dispositivos a utilizar.

Tarea 2: Realizar un documento con la información resumida de la programación y configuración del sistema.

Tarea 3: Crear el código de las rutinas y subrutinas para la ejecución de cada modo de operación, aplicando las funciones anteriormente consultadas.

Tarea 4: Realizar la configuración inicial, agregando la CPU, La fuente de poder y los módulos utilizados, en el módulo hardware del Step 7.

Tarea 5: Configurar las líneas de comunicación Profibus, Ethernet y agregar los dispositivos a estas, mediante el modulo hardware.

Tarea 6: Realizar la configuración física del módulo EM277 Y el variador de frecuencia Micro Master 440.

Tarea 7: Realizar el código de comunicación entre el PLC Maestro y Los PLC S7300, S7200 y el variador de frecuencia, mediante el software Step7 y Step7 MicroWin.

Tarea 8: Configurar el panel táctil mediante el software WinCC flexible.

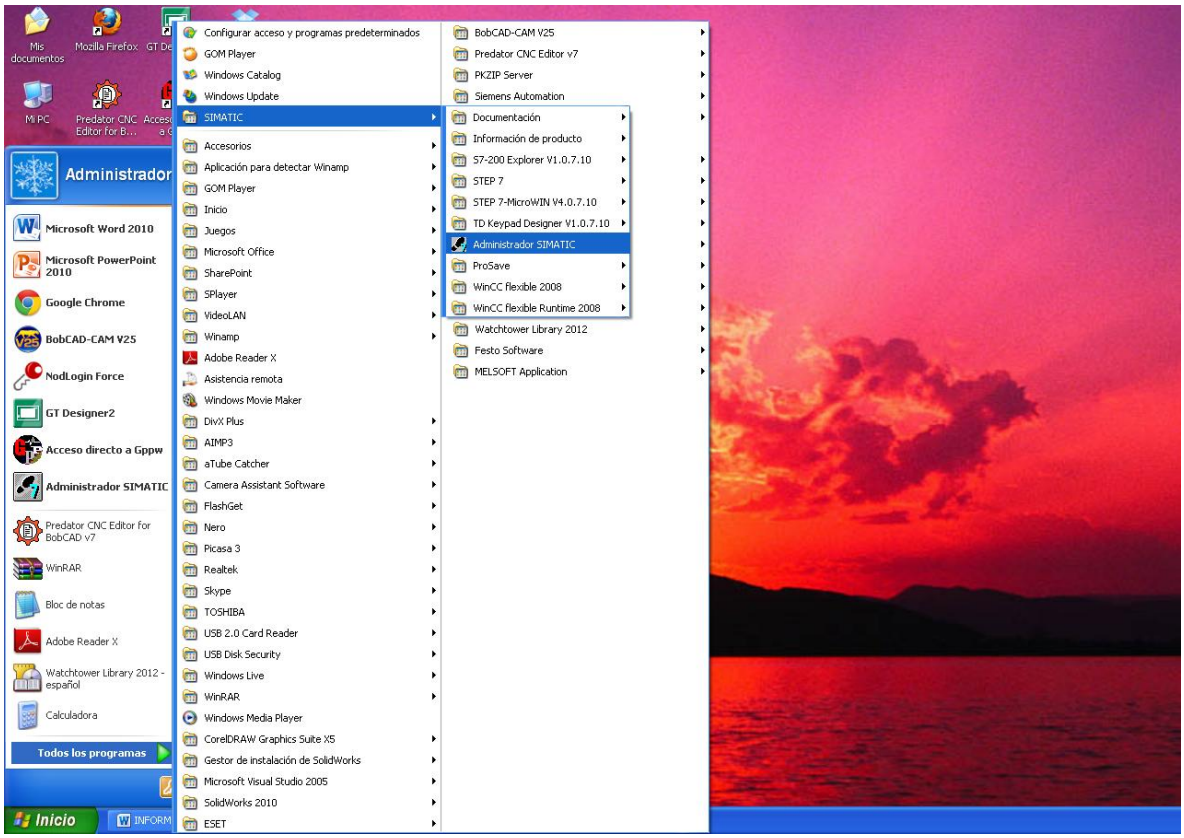
Tarea 9: Probar cada secuencia y función programada, y documentar los pasos necesarios para que sean recreadas.

3. CONFIGURACIÓN DE HARDWARE

Una vez realizada la parametrizacion de cada una de las tarjetas mediante el software *Festo Configuration Tool* para los motores pasó a paso y *Wmemoc* para el servomotor, el primer paso a realizar es la configuración de Hardware dentro del ambiente de Simatic Step 7. A continuación se mostrara como configurar la comunicación del PLC con cada uno de los dispositivos.

El primer paso es abrir el *Administrador SIMATIC* (ver figura 3).

Figura 3: Administrador SIMATIC.

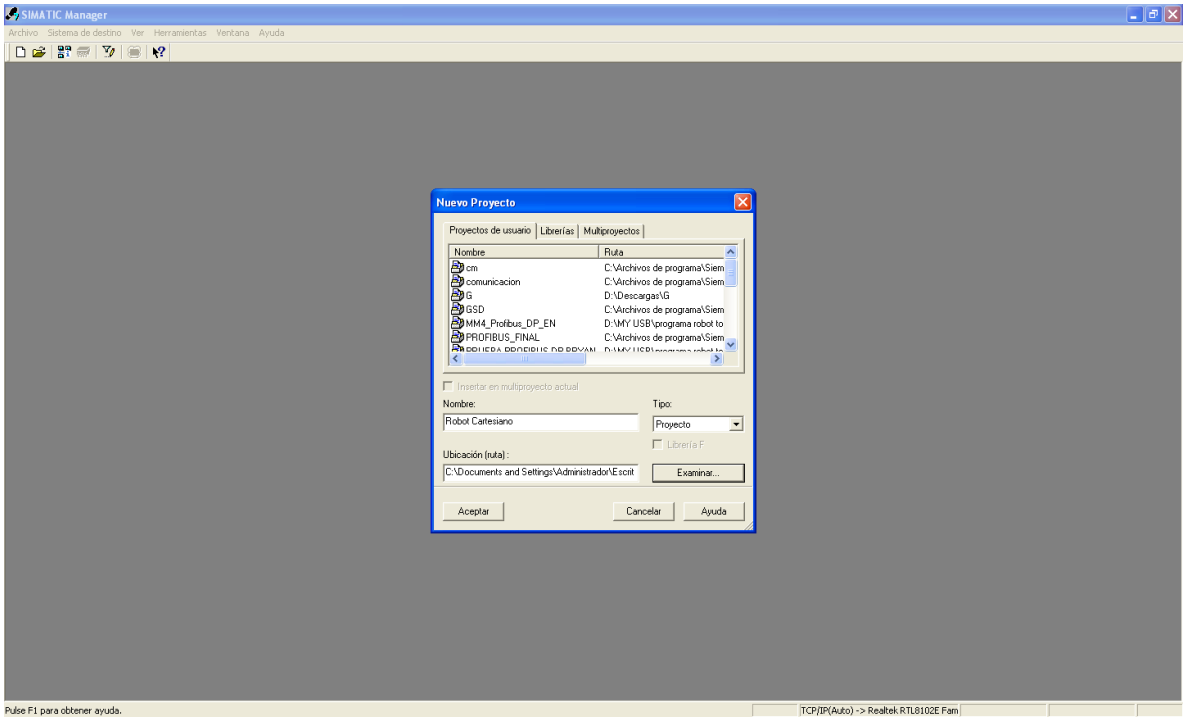


Fuente: Autor.

Una vez abierto el Administrador SIMATIC, dirigirse a *Archivo, Nuevo*, o simplemente precionar las teclas Ctrl y N simultaneamente para abrir la ventana

de Nuevo Proyecto, luego que se halla abierto la ventana se puede dar un nombre, en este caso sera *Robot cartesiano* (ver figura 4).

Figura 4: Nuevo proyecto.



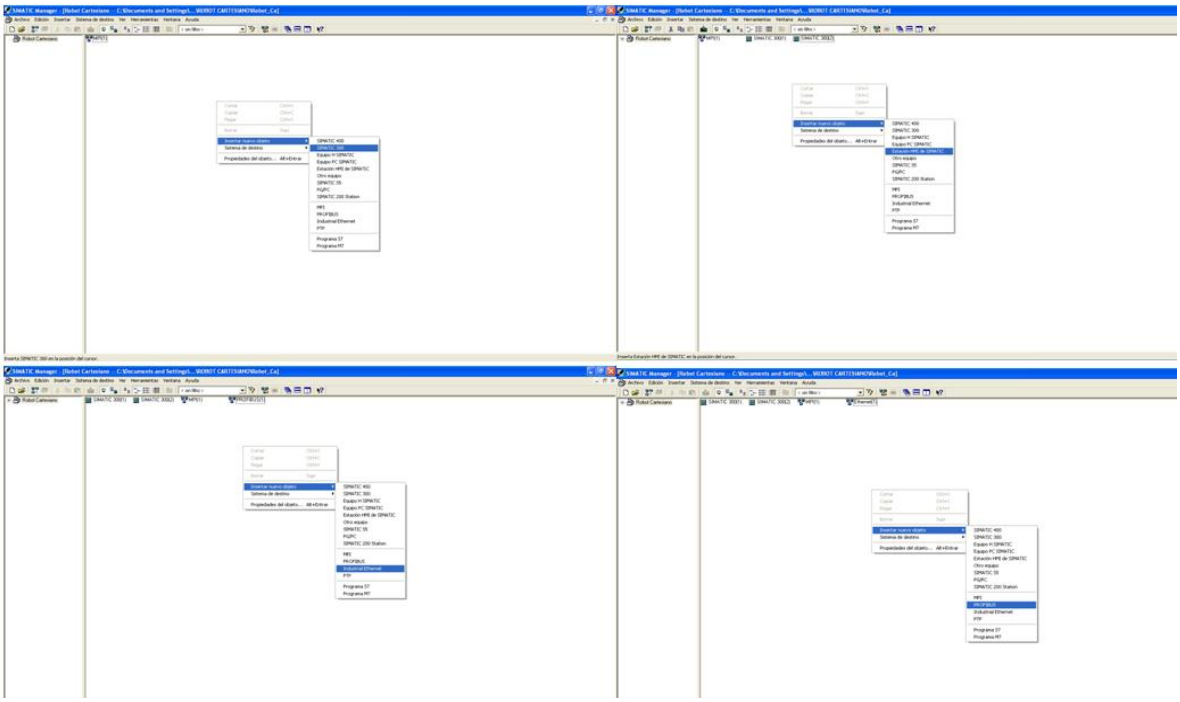
Fuente: Autor.

El proyecto nuevo se creara vacio y con una red MPI, entonces se procede a ingresar los dispositivos neceserarios. Para este proyecto se crearan 2 equipos SIMATIC 300, una estacion HMI de SIMATIC, una red Profubus y una red industrial Ethernet. Para agregar estos dispositivos, dar click derecho sobre la pantalla, elegir *Insertar nuevo objeto* y seleccionar cada uno de los dispositivos (ver figura 5). Al insertar el panel tactil (estacion HMI de SIMATIC) saldra un menu donde se debe indicar el nombre y versión del equipo, para este caso el panel usado es el *OP 177B 6" color PN/DP* el cual se encuentra en *Panels-170* (ver figura 6). Tambien puede modificarse el nombre a los dispositivos, en este caso se modifica el nombre de los dos equipos SIMATIC 300 para identificar el esclavo y

el maestro (ver figura 7). Una vez agregados los dispositivos se ingresa al equipo SIMATIC *MAESTRO* y luego a *Hardware* (ver figura 8).

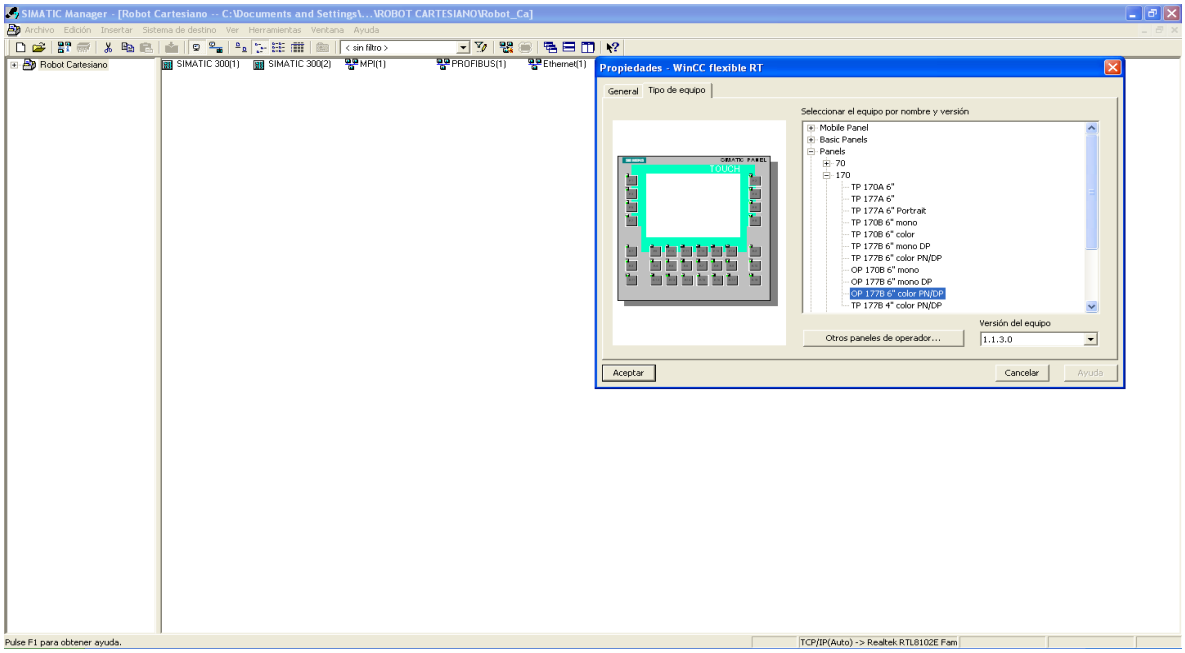
Dentro de la pantalla de hardware se configuran las lineas de comunicación, se establece el enlace entre los dispositivos en la red y el plc y se definen los modulos autilizar, en pocas palabras a qui se realiza la configuracion de los dispositivos. Para iniciar la configuracion se empieza agregando un bastidor, que es el perfil soporte donde se anclan los elementos del PLC, para agregarlo se da click derecho, *Insertar objeto*, *SIMATIC 300*, *BASTIDOR 300* y *Perfil soporte*, este tambien puede agregarse direntamente desde la ventana derecha en *SIMATIC 300, BASTIDOR300* y *Perfil soporte* (ver figura 9).

Figura 5: Agregando dispositivos.



Fuente: Autor.

Figura 6: Agregando HMI.

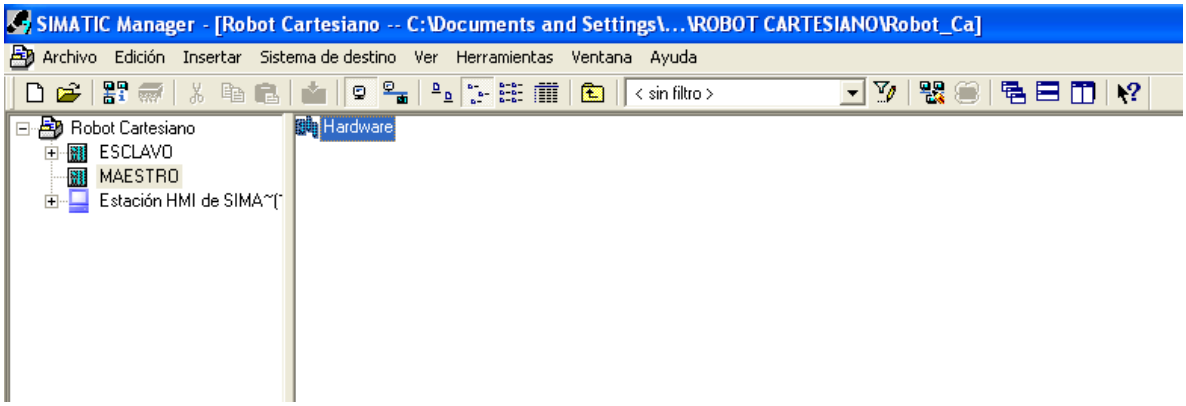


Fuente: Autor.

Figura 7: Maestro y Esclavo.

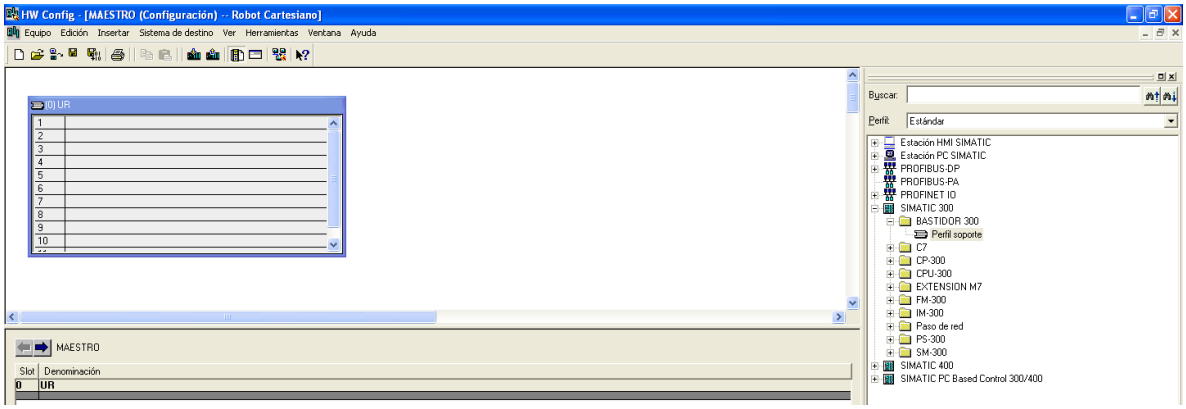
Fuente: Autor.

Figura 8: Hardware.



Fuente: Autor.

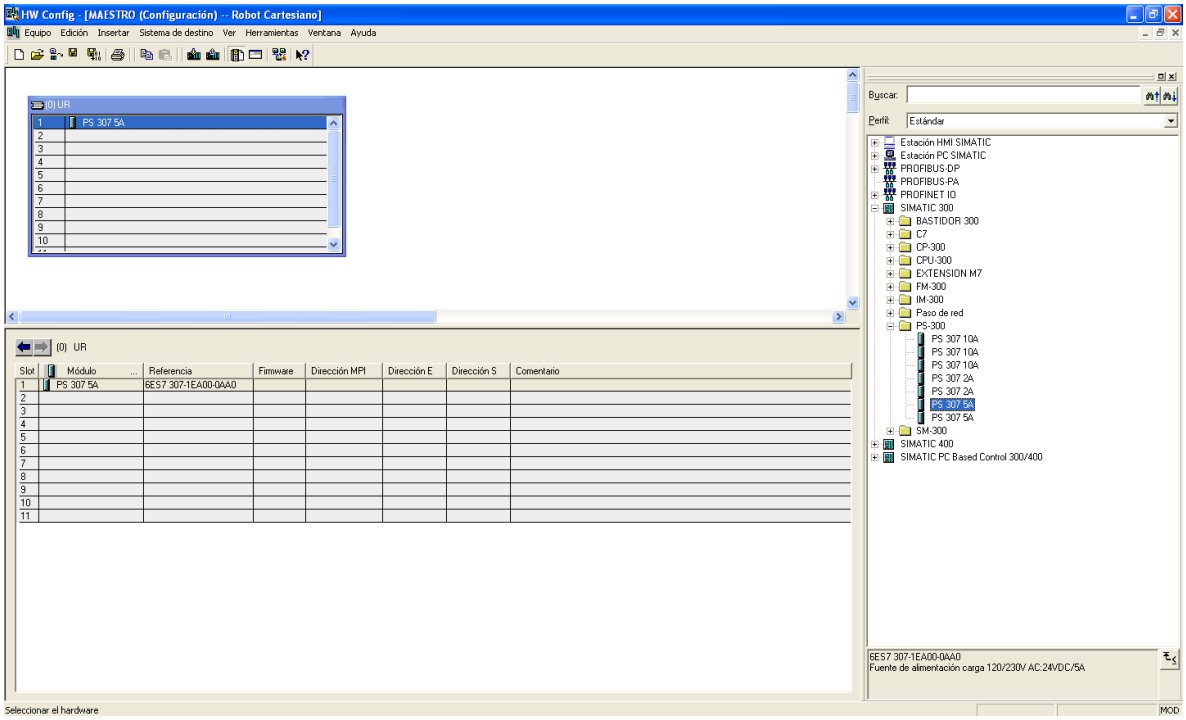
Figura 9: Perfil soporte.



Fuente: Autor.

El primer slot del bastidor corresponde a la fuente de alimentación la cual en este caso es el modulo PS 307 5A con referencia 6ES7 307-1EA01-0AA0, esta se encuentra en la ventana derecha, *SIMATIC 300*, *PS 300* y *PS 300 5A*. Se selecciona la que tenga la referencia anterior (ver figura 10).

Figura 10: Fuente PS 307 5A.



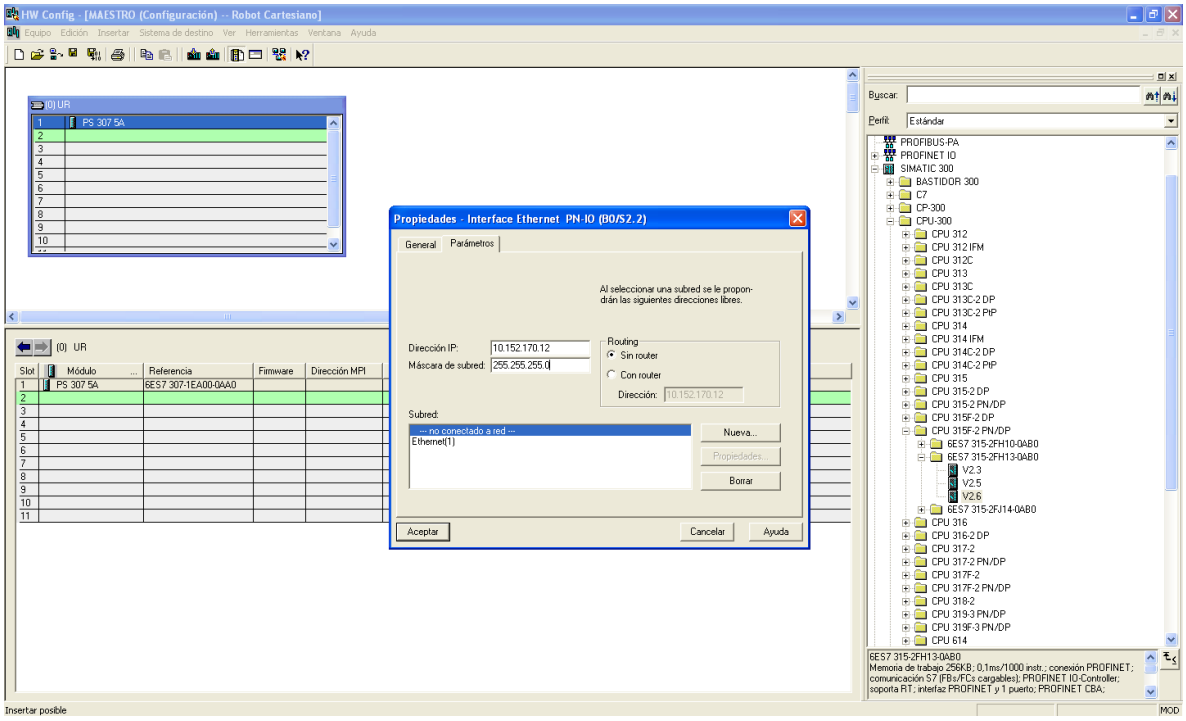
Fuente: Autor.

El segundo slot corresponde a la CPU que es el modulo principal (PLC), que en este caso es el modulo CPU 315F-2PN/DP con referencia 6ES7 315-2FH13-0AB0, esta se encuentra en la ventana derecha, *SIMATIC 300, CPU 300, CPU 315F-2 PN/DP y 6ES7 315-2FH13-0AB0*, se selecciona la versión 2.6 (ver figura 11).

Para llevar los dispositivos hasta el bastidor hay que seleccionar el dispositivo manteniendo click izquierdo oprimido y arrastrar hasta la posición que le corresponde. Una vez llevada la CPU, se abrirá una ventana en donde se puede configurar la dirección IP a utilizar, esta se debe a que la CPU viene con puerto Ethernet, el cual será utilizado para establecer la comunicación Ethernet entre las dos CPU S7 300 y además para cargar y descargar el programa. En el laboratorio de automatización se definió para cada dispositivo una dirección IP específica, en

lo que respecta al PLC del robot cartesiano le correspondió la dirección IP **10.152.170.12**, la máscara de subred es la **255.255.255.0** (ver figura 11).

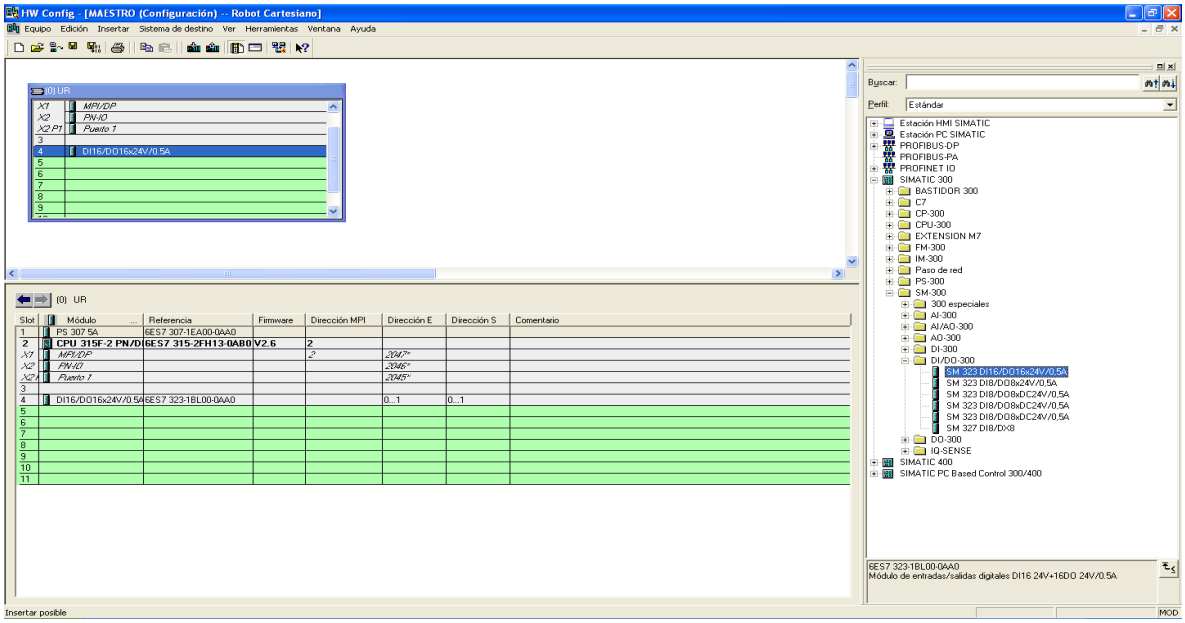
Figura 11: CPU 315F-2PN/DP.



Fuente: Autor.

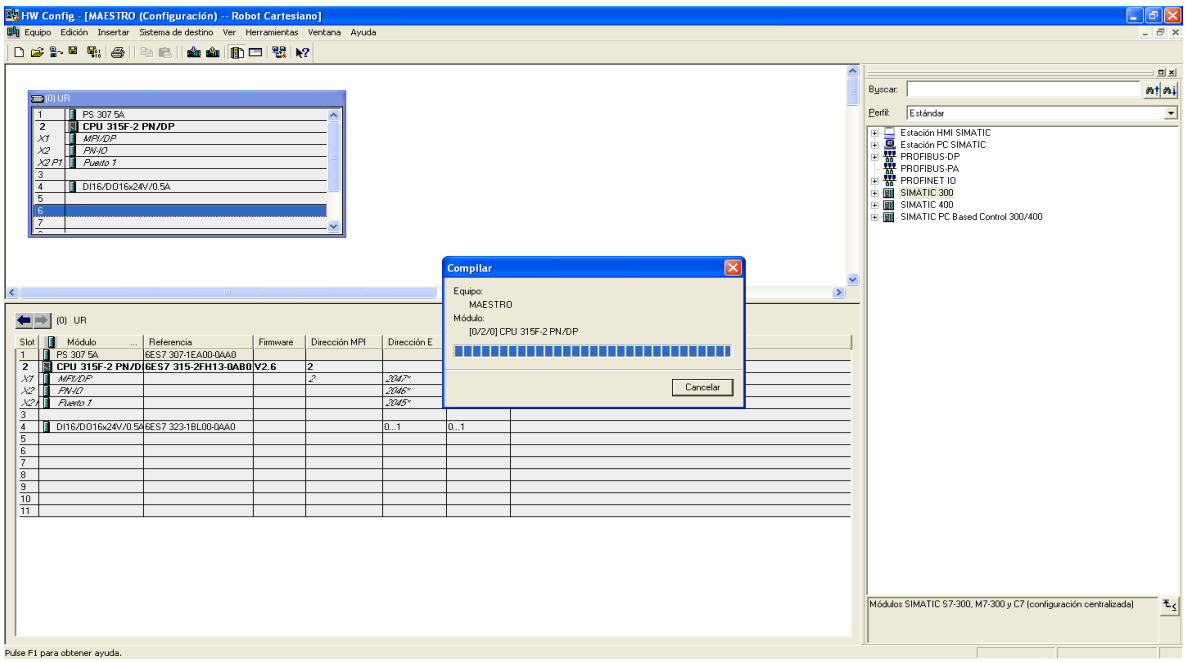
El slot número 3 es para los módulos de interfaz (módulos IM) y del 4 para abajo es para los demás. Entonces el siguiente dispositivo se ubicara en el slot número 4 ya que es un módulo digital de dos bytes de entrada y dos bytes de salida, esto quiere decir que se tiene a disposición 16 bits de entradas y 16 bits de salida, en este caso el modulo usado es el SM323 DI16/DO16xDC24V de referencia 6ES7 323-1BL00-0AA0, este se encuentra ubicado en la ventana derecha, *SIMATIC 300, SM-300, DI/DO-300 y SM323 DI16/DO16xDC24V* (ver figura 12). Una vez agregados estos dispositivos, se guarda y compila la configuracion para que step 7 asimile los cambios (ver figura 13). A continuacion se procedera a configurar las lineas de comunicación y acoplar los dispositivos a dichas redes.

Figura 12: SM323 DI16/DO16xDC24V.



Fuente: Autor.

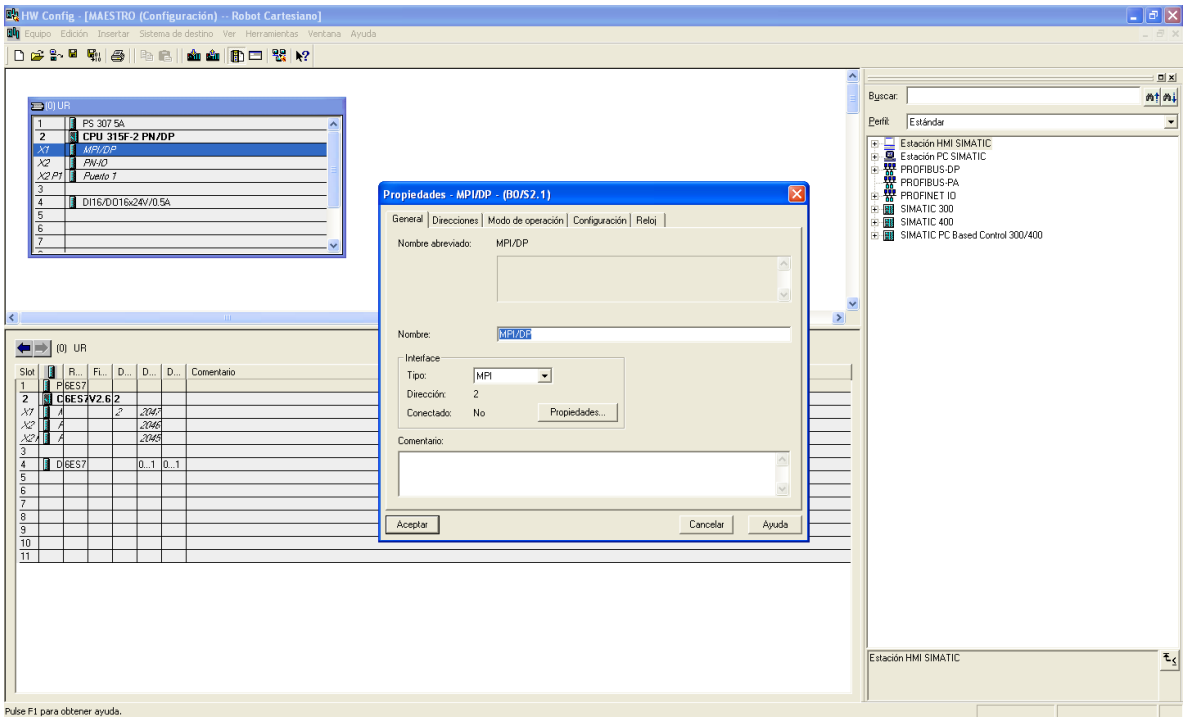
Figura 13: Guardar y compilar.



Fuente: Autor.

Para configurar la red Profibus hay que dirigirse al slot número 2 que corresponde a la CPU y dar doble click izquierdo en el puerto *X1*, este puerto es el puerto físico donde se conecta el cable PROFIBUS al PLC. Una vez hecho esto se abrirá una pantalla donde se puede configurar la red de comunicación a utilizar (ver figura 14), como se está trabajando el puerto X1 solo se puede seleccionar entre *MPI* y *PROFIBUS*, si se requiere utilizar una red Ethernet el puerto *X2* es el que se utiliza para configurarla, más adelante se mostrara como.

Figura 14: Inicio red Profibus.

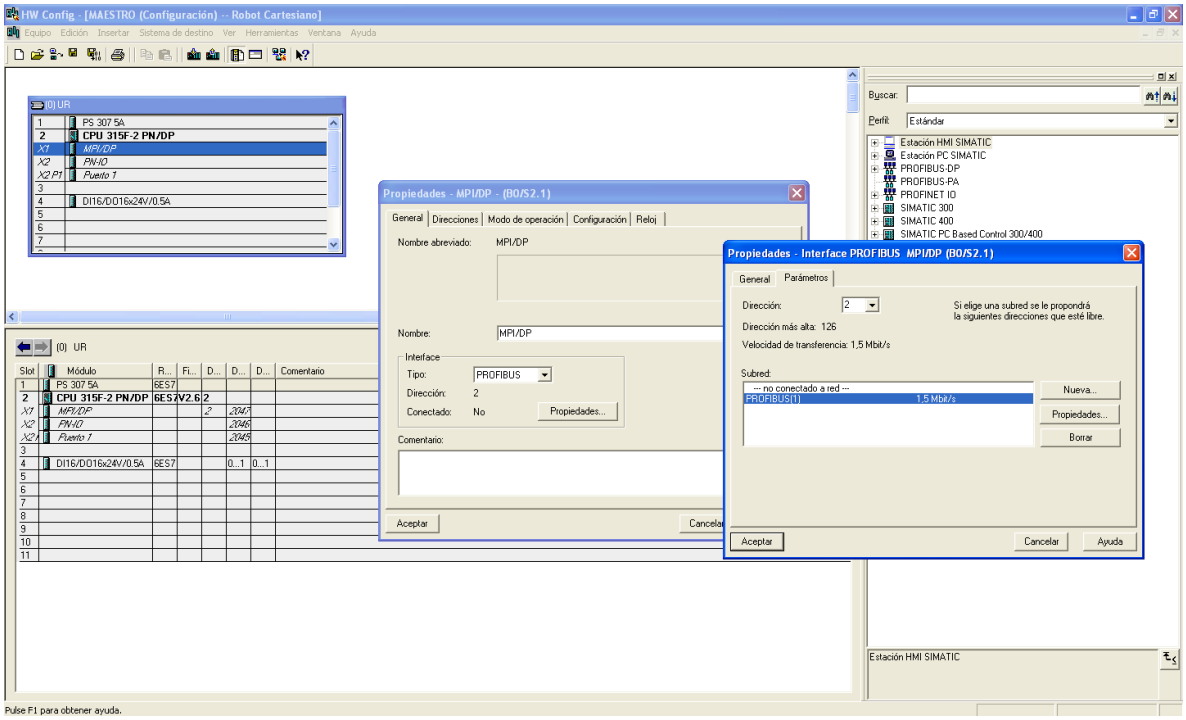


Fuente: Autor.

Luego hay que dirigirse a la pestaña que dice *Tipo* y seleccionar *PROFIBUS*, una vez seleccionado *PROFIBUS* se abrirá la ventana de propiedades en donde se especifica el número de la estación, en este caso se le asignara el número 2 el cual queda establecido para el PLC maestro, una vez definida la dirección Profibus hay que dirigirse a *Subred* y selecciona la red Profibus que inicialmente se había

creado, esta es *PROFIBUS (1)* (ver figura 15). Si no se crea desde el inicio la red a utilizar se puede crear en la misma pantalla de propiedades donde dice *Nueva.*

Figura 15: Numero de estacion, PLC maestro.

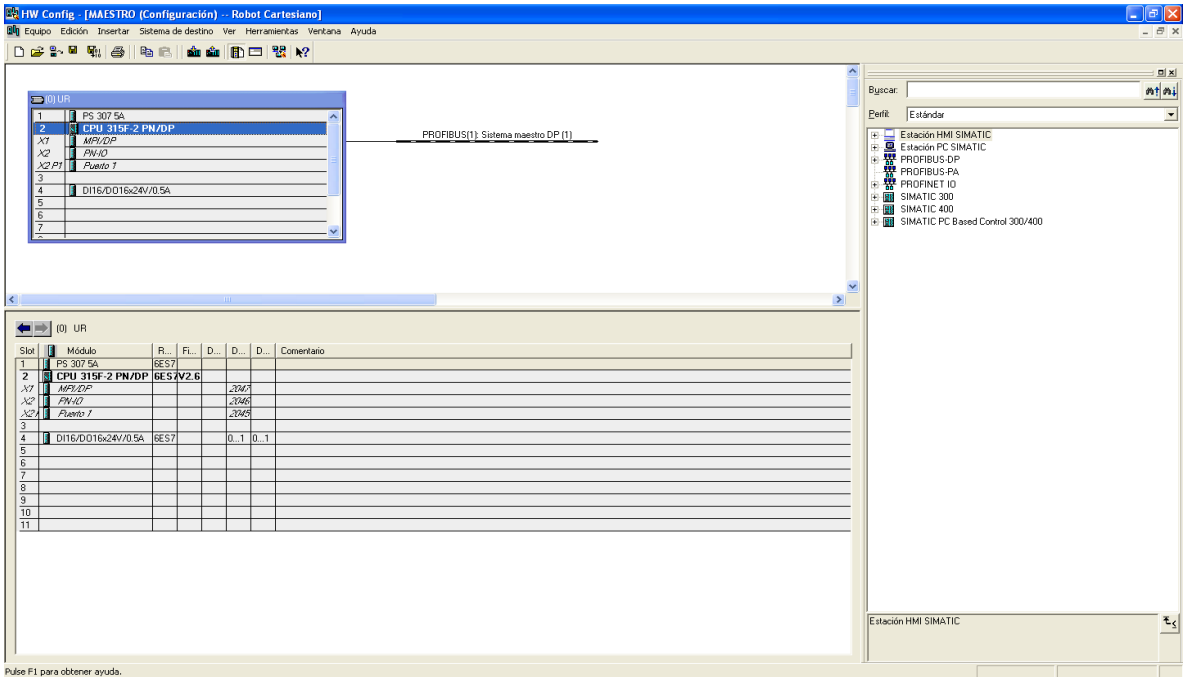


Fuente: Autor.

Una vez realizados los pasos anteriores, se termina la configuración dando en los botones *Aceptar*, entonces la red PROFIBUS será creada en la pantalla *HW Config* como una línea negra conectada en el puerto *X1* (ver figura 16). Otra forma para crear una red es desde *NetPro*, para crear la red hay que primordialmente guardar y compilar, y luego dirigirse al icono de *Configurar red* , este botón abrirá la pantalla de *NetPro* (ver figura 17), si la red no se creó inicialmente, se puede crear dirigiéndose a la carpeta *Subredes*, la cual está ubicada en la parte derecha de la ventana *NetPro*, en esta carpeta se da doble cilck izquierdo en la red a utilizar y esta se creara, pero en este caso no es necesario ya que las redes a utilizar se crearon des del inicio, así que solo hay que acoplar la CPU con la red

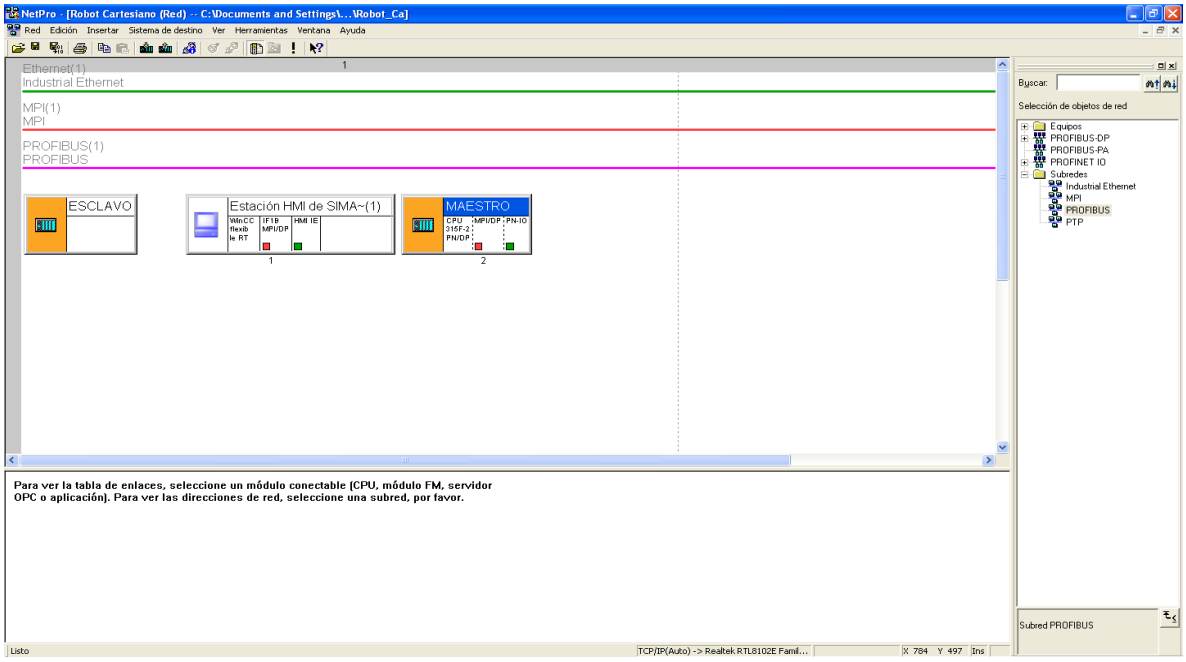
Profibus ya creada, para esto hay que dar doble click izquierdo en el recuadro *MPI/DP* de la CPU *MAESTRO*, y se aplica el mismo procedimiento realizado anteriormente (ver figura 18), una vez realizada la configuración se puede ver que el acople entre la CPU y la línea Profibus creada anteriormente fue realizado ya que aparecerá una línea que une la CPU con la línea de red (ver figura 19). Una vez configurada la red hay que guardar y compilar, pero como aún no se ha configurado el PLC esclavo, ni el panel táctil, la compilación generara errores (ver figura 20). Luego de guardar y compilar se regresa al ambiente *HW Config* para continuar con la configuración, para esto se da doble click sobre la CPU *MAESTRO* desde *NetPro* (ver figura 21).

Figura 16: Red creada.



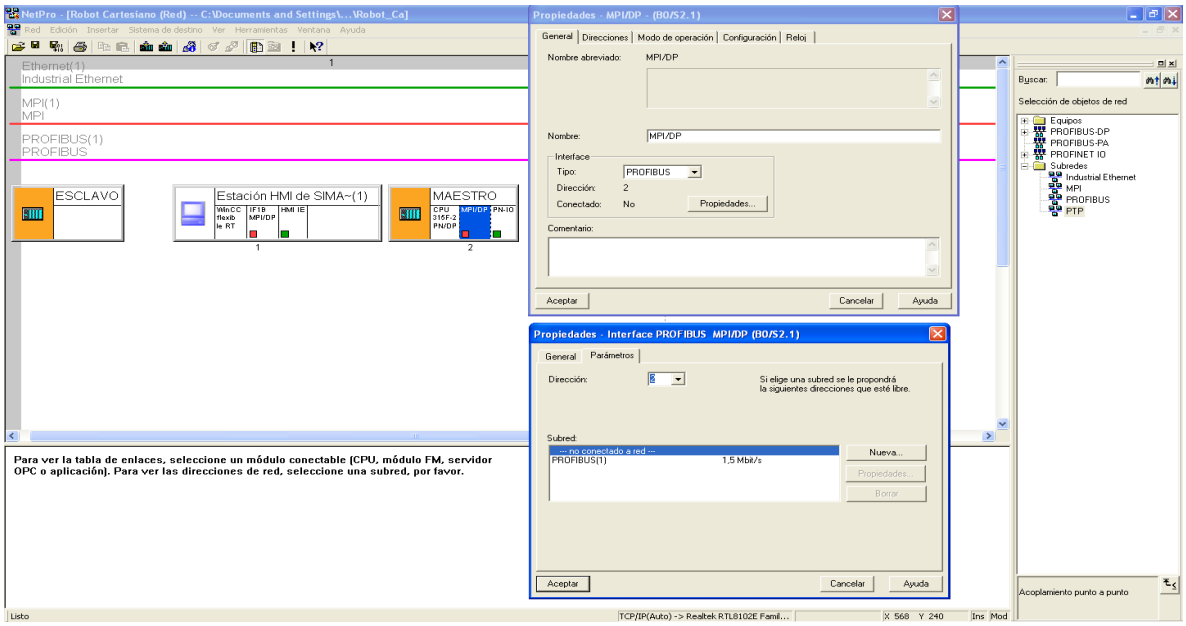
Fuente: Autor.

Figura 17: Creando red desde NetPro.



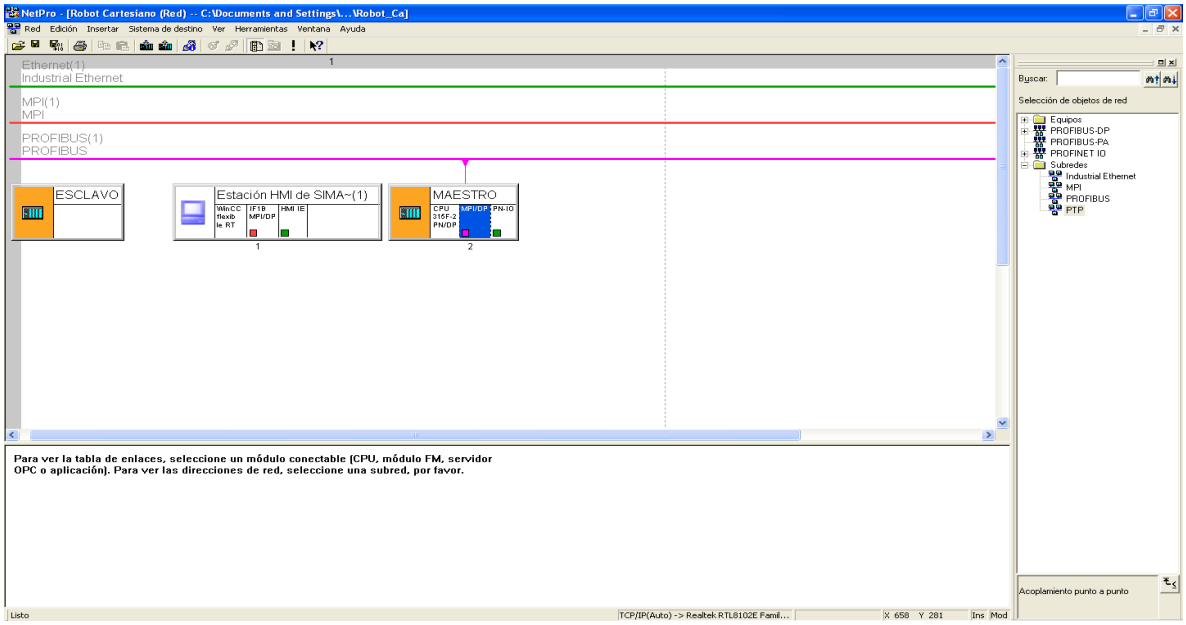
Fuente: Autor.

Figura 18: Configuracion desde NetPro.



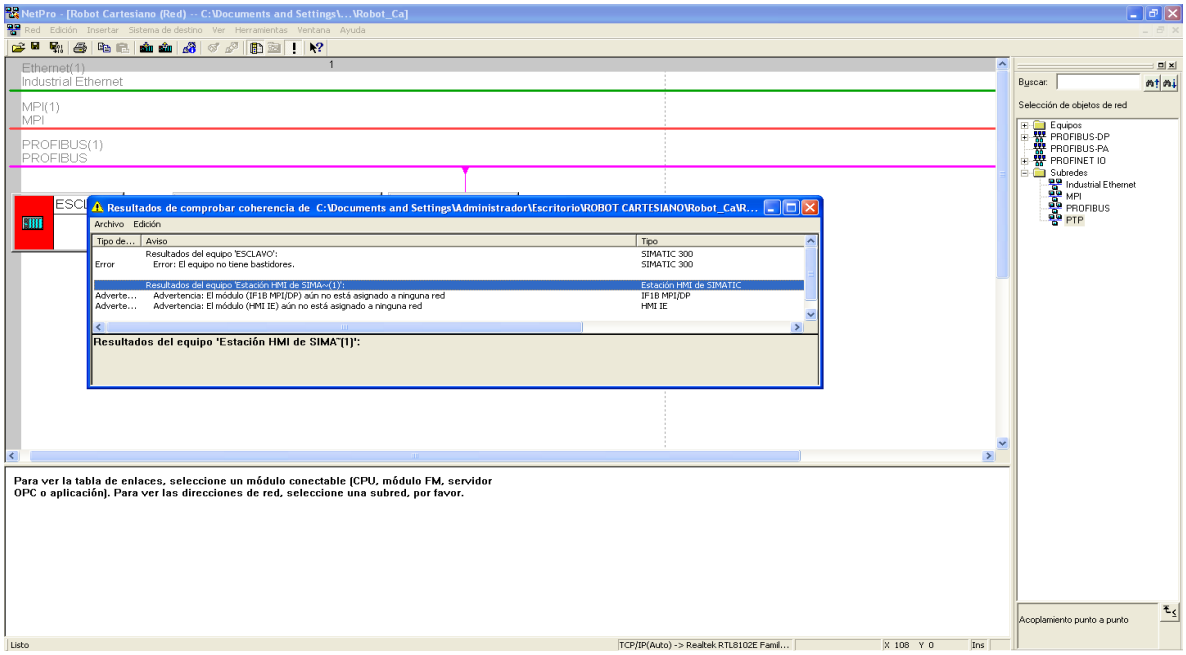
Fuente: Autor.

Figura 19: Red configurada desde NetPro.



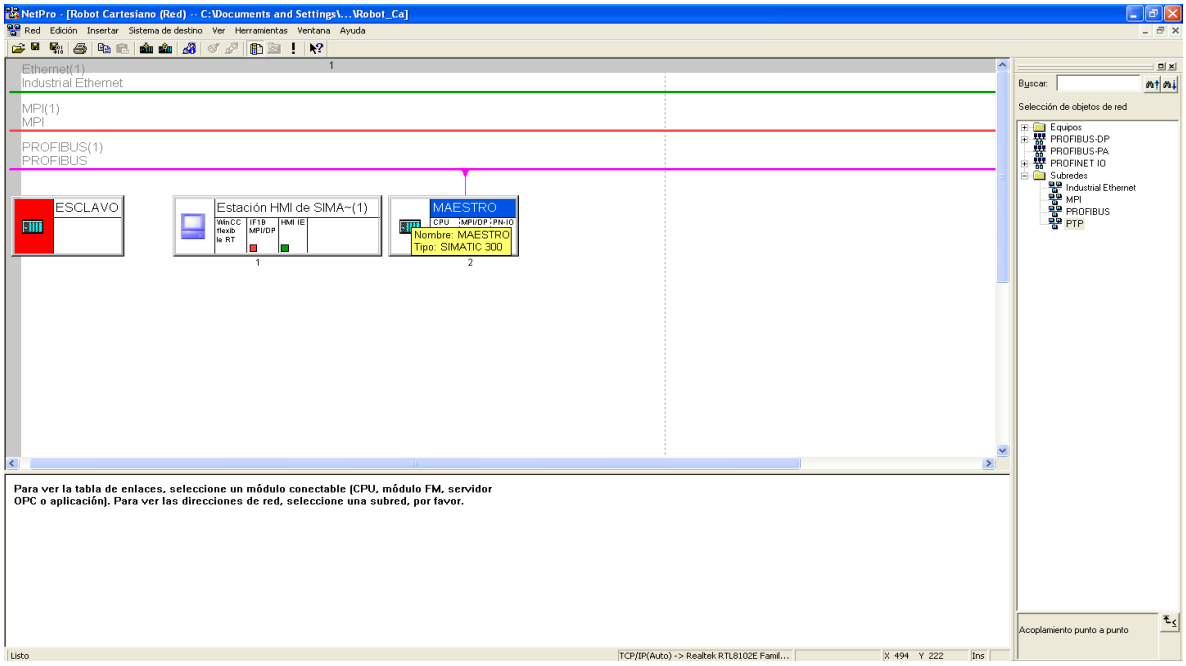
Fuente: Autor.

Figura 20: Errores, compilacion NetPro.



Fuente: Autor.

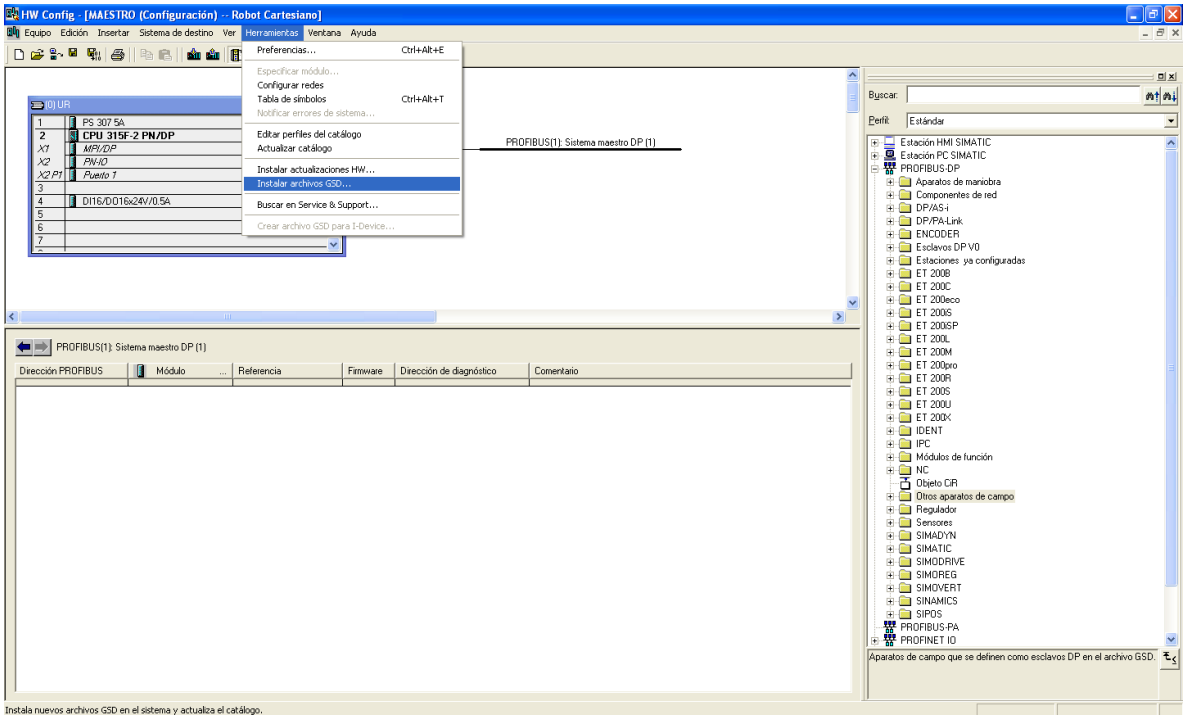
Figura 21: Ir a HW Config desde NetPro.



Fuente: Autor.

Luego de esto se procederá a agregar las tarjetas de los motores. Primero que todo hay que agregar los archivos GSD antes de agregar los dispositivos, ya que si estos no viene por defecto en la instalación del software, no aparecerán. Para instalar un archivo GSD hay que dirigirse a *Herramientas, Instalar archivos GSD* en la pantalla *HW Config* (ver figura 22).

Figura 22: Intalar archivos GSD.



Fuente: Autor.

Los archivos GSD de las tarjetas vienen en los CDS de instalación de FESTO, para las tarjetas CMMS-ST-C8-7 la carpeta GSD se encuentra en *FB-Tools, Profibus y GSD-Datei_070718*. Para la tarjeta SEC-AC-305-PB la carpeta GSD se encuentra en *Field bus y Profibus*. Para el variador de frecuencia MM440 6SE6440-2UC13-7AA1 la carpeta GSD también se encuentra en el CD de instalación la cual se encuentra directamente en el CD como *GSD File*. Para el módulo EM277 6ES5 277-0AA22-0XA0 la carpeta GSD puede ser descargada desde la página de Siemens[1].

Los archivos GSD son para agregar dispositivos Profibus, que son los que se encuentran en el catálogo de *HW Config* en la pestaña *PROFIBUS-DP*, pero si se necesita agregar una CPU o algún modulo para realizar la configuración sobre el

[1]http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&objId=113652&nodeid0=10805317&lang=en&siteid=cseus&aktprim=0&objaction=csview&extranet=standard&viewreg=WW.

bastidor, se tiene que ejecutar la opción *Instalar actualizaciones HW*, la cual buscara desde la página de siemens el dispositivo que se necesite.

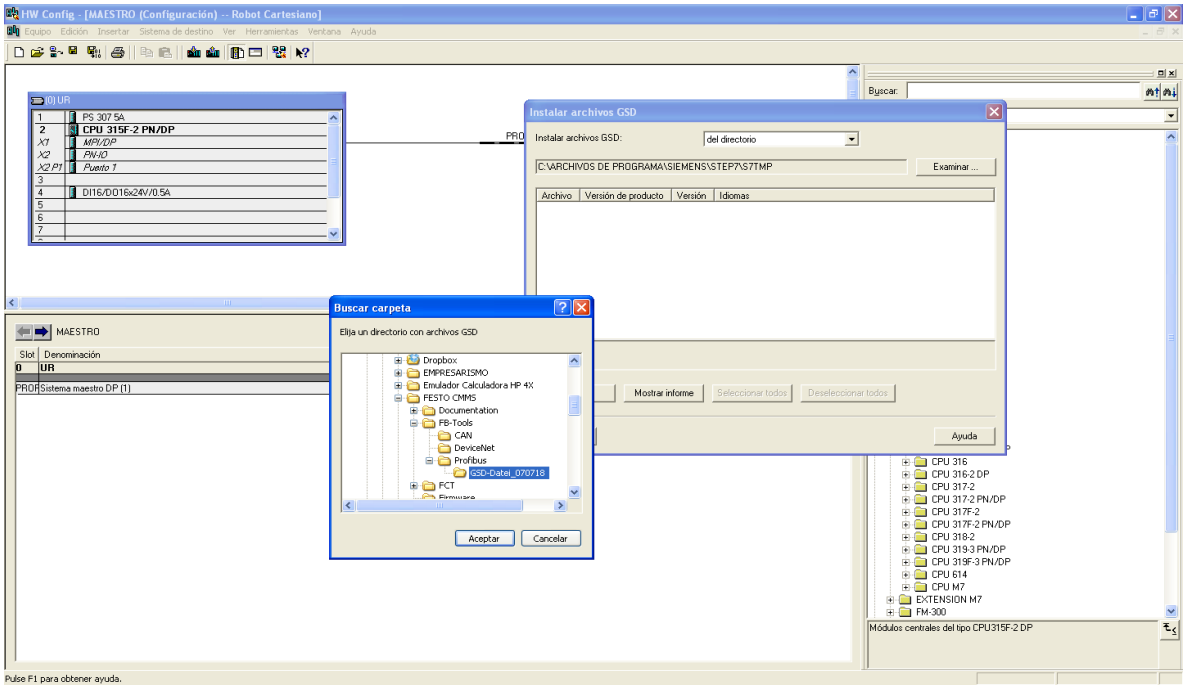
Una vez se tengan los archivos, el siguiente paso es dirigirse a *Herramientas, Instalar archivos GSD*, luego de esto se abrirá una ventana donde se especifica la dirección en donde se encuentra la carpeta GSD del dispositivo a instalar (ver figura 23), cuando se halla seleccionado la dirección de la carpeta, aparecerá el nombre del archivo GSD del dispositivo y más abajo el nombre del dispositivo. Seleccionando el nombre del archivo GSD, dando click en la opción *Instalar* y por último actualizando, se incluirá el dispositivo en el catálogo de *HW- Config* (ver figura 24), para actualizar el catalogo hay que dirigirse a *Herramientas, Actualizar catalogo* (ver figura 25). La carpeta GSD de un dispositivo contiene el archivo GSD y las imágenes de este, así que cuan el dispositivo no aparece en el catálogo aun habiendo sido instalado, la solución es pegar el archivo GSD en la carpeta *GSD* del STEP7[2] y las imágenes en la carpeta *NSBMP* del STEP7 [3].

La CPU del PLC esclavo, la cual es de referencia CPU 315F-2PN/DP 6ES7 315-2FJ14-0AB0, no viene por defecto en la instalación del software, entonces antes de continuar con la configuración es necesario descargar la CPU en el catálogo, con la finalidad de poder configurar todos los elementos en el bastidor. Para descargarla hay que dirigirse a *Instalar actualizaciones HW* en el menú *Herramientas* (ver figura 26), una vez seleccionada esta opción, aparecerá una ventana donde se mostraran todas las actualizaciones de los dispositivos en el catálogo. Para que las actualizaciones aparezcan hay que seleccionar la opción *Ejecutar* (ver figura 27), una vez hayan aparecido las actualizaciones, hay que buscar el dispositivo a descargar y seleccionarlo, en este caso es la CPU 315F-2PN/DP 6ES7 315-2FJ14-0AB0 la cual aparece en la lista de actualizaciones como *CPU 31X(F)-2 PN/DP V3.1* con referencia *6ES7 31X-2??14-0AB0*, luego dar *Descargar* y por ultimo actualizar el catalogo (ver figura 28).

[2] La carpeta GSD del STEP7 está ubicada en C:\Archivos de programa\Siemens\Step7\S7DATA\GSD.
[3] La carpeta NSBMP del STEP7 está ubicada en C:\Archivos de programa\Siemens\Step7\S7DATA\NSBMP.

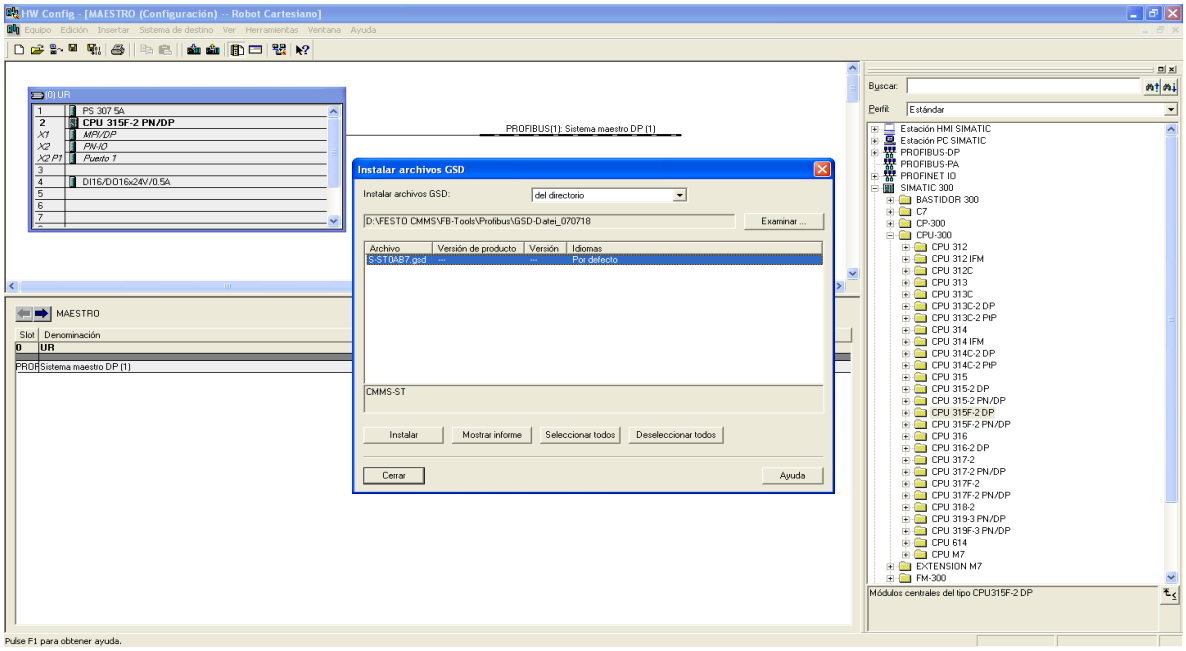
Con todos los dispositivos requeridos ya descargados se puede continuar con la configuración. En el catalogo en la opción *PROFIBUS-DP* se encontraran los dispositivos a anclar en la red, empezando con las tarjetas de los motores paso a paso (CMMS-ST-C8-7), hay que dirigirse a *Otros aparatos de campo, Accionamientos, Festo y CMMS-ST* , luego arrastrar el icono de CMMS-ST hacia la línea Profibus y entonces aparecerá una ventana donde se debe poner la dirección Profibus, iniciando por la tarjeta del motor del eje x, se ingresa la dirección 3, ya que esta dirección fue la que se le asigno en los D-switch así como en el FCT (ver figura 29), posteriormente, se selecciona el modulo a utilizar, hay dos módulos disponibles que definen la forma de transmisión de los datos.

Figura 23: Seleccionar direccion de la carpeta GSD.



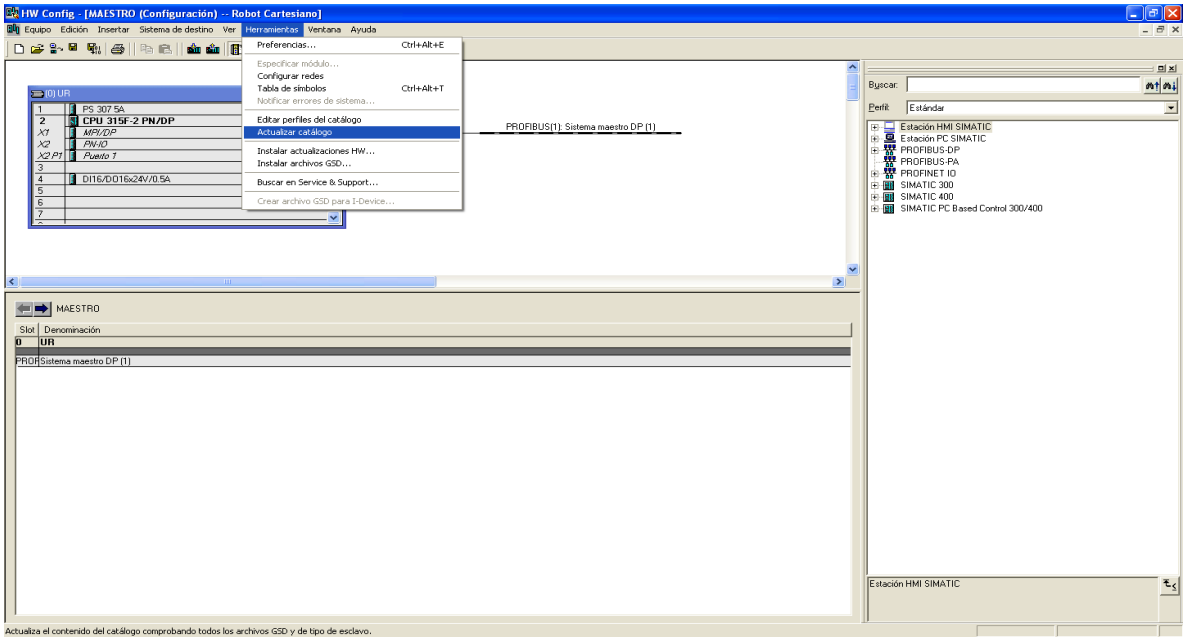
Fuente: Autor.

Figura 24: Seleccionar e instalar GSD.



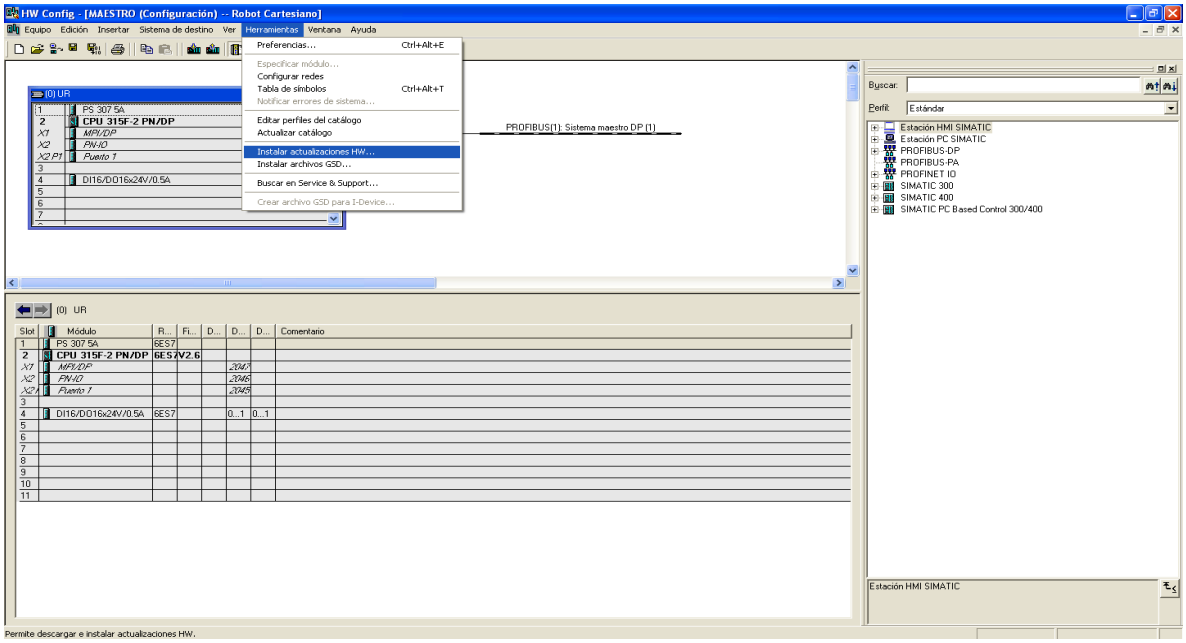
Fuente: Autor.

Figura 25: Actualizar catalogo.



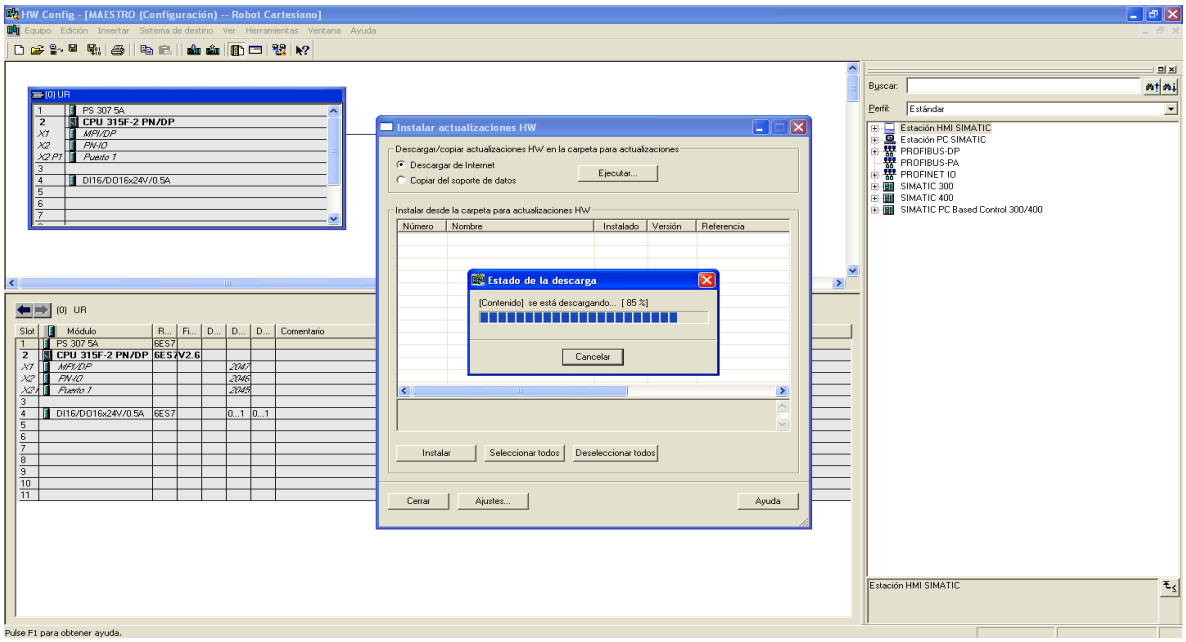
Fuente: Autor.

Figura 26: Instalar CPU.



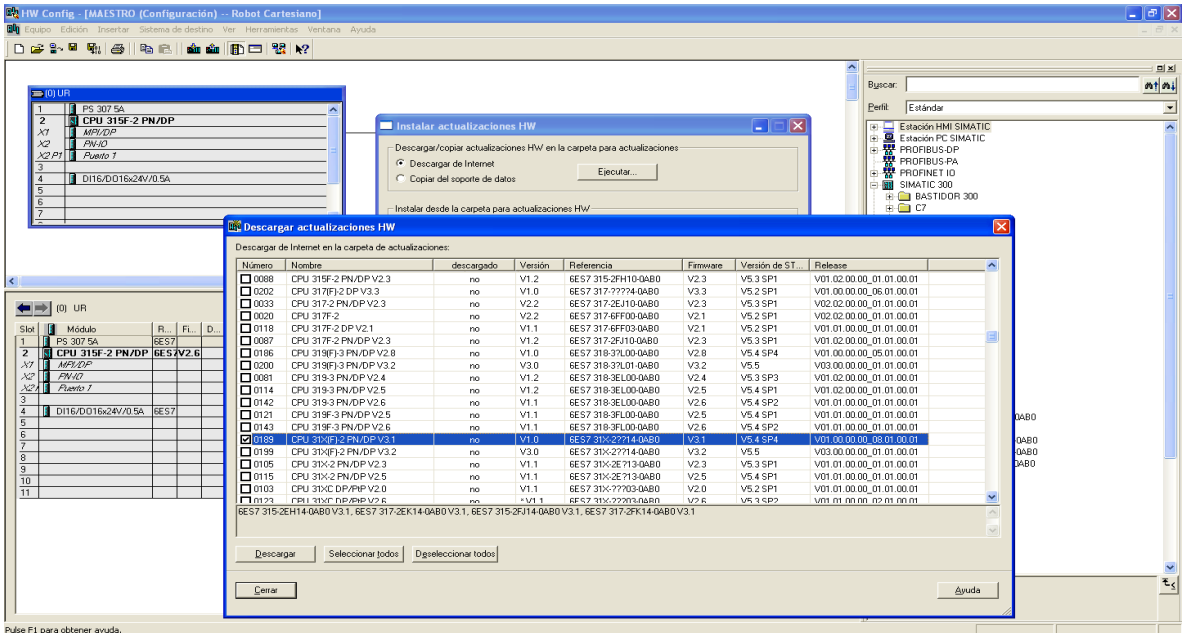
Fuente: Autor.

Figura 27: Ejecutando busqueda de actualizaciones.



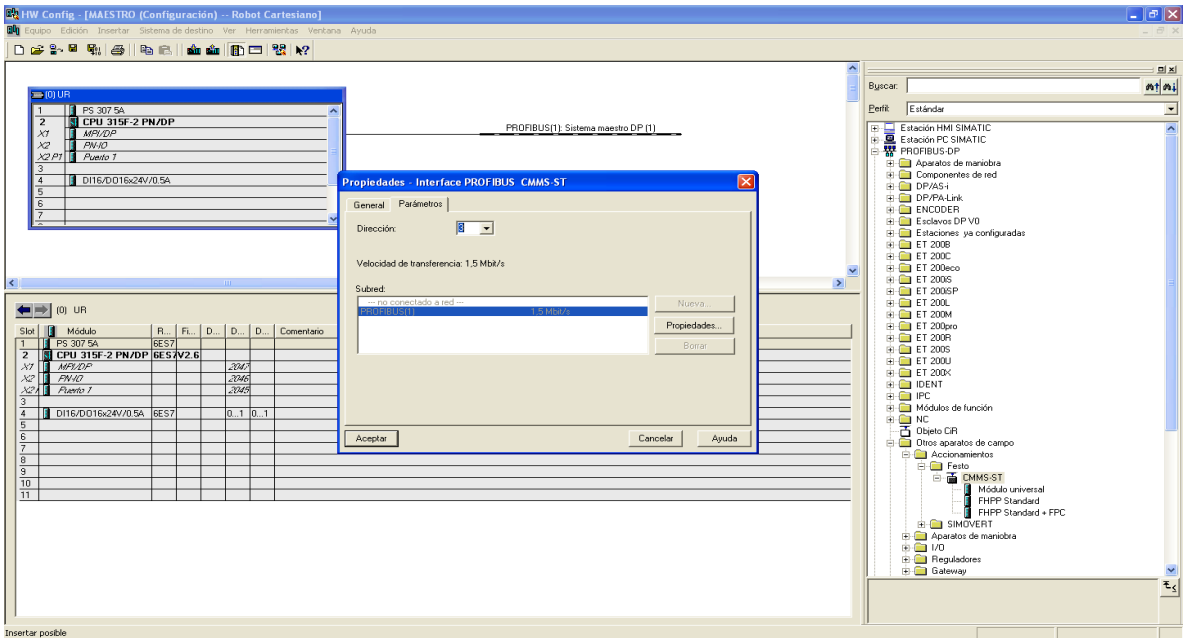
Fuente: Autor.

Figura 28: Selecion de CPU 31X (F)-2 PN/DP V3.1 con referencia 6ES7 31X-2??14-0AB0.



Fuente: Autor.

Figura 29: Dirección Profibus para tarjeta CMMS-ST eje-x .



Fuente: Autor.

El primer módulo es el *FHPP Standard* y el segundo es el *FHPP Standard* + *FPC*. El FHPP es un perfil de datos adaptado a tareas de manipulación y posicionamiento, sus siglas provienen de *Festo handling and positioning profile*. El perfil estándar tiene la capacidad de manejar 8 bytes de datos de entradas y salidas. El perfil con canal de parámetros a saber el *Festo handling and positioning profile with parameter channel* (*FHPP Standard* + *FPC*) tiene la capacidad de manejar 16 bytes de datos de entradas y salidas. Ahora para seleccionar la opción *FHPP Standard* + *FPC,* hay que dar click sobre el icono de la tarjeta para que aparezcan los dos espacios donde va la configuración de esta y luego dar doble click sobre la opción *FHPP Standard* + *FPC* (ver figura 30). Luego se definen las direcciones de entrada y salida, para el primer slot escribir la dirección 288 y para el segundo slot la 296 (ver figura 31). Estas direcciones son desde donde van a empezar los 16 bytes de entrada y salida mediante los que el PLC interactúa con la tarjeta, de esta forma el PLC identifica que bloque de función le corresponde a cada tarjeta.

De la misma forma se procede para agregar la tarjeta controladora del eje Y, pero ahora se dará la dirección 4 para Profibus, la dirección 304 para el slot 1 y la dirección 312 para el slot 2 (ver figura 32). El siguiente paso es agregar la tarjeta del servo motor la cual tiene referencia SEC-AC-305-PB, para agregarla hay que dirigirse a *PROFIBUS-DP* y desplegar la opción *Reguladores* en *Otros aparatos de campo,* allí aparecerá el módulo *FESTO SEC-AC* el cual se arrastra y pone en la línea Profibus. Las direcciones para este son, 5 para Profibus y 256 para entradas y salidas. La dirección 256 es la dirección inicial desde donde se establecerán los 16 bytes de entrada y salida (ver figura 33).

El siguiente elemento a anclar es el módulo EM277 6ES5 277-0AA22-0XA0, el cual es necesario para el intercambio de datos entre el PLC maestro S7-300 y el PLC S7-200. Para agregarlo hay que dirigirse a *Otros aparatos de campo, PLC* y *SIMATIC,* hay aparecerá el módulo *EM277 PROFIBUS-DP,* el cual hay que

arrastrar y poner en la línea Profibus (ver figura 34), a este módulo se le pondrá la dirección 6, la cual también se define físicamente en la perilla, el siguiente paso es elegir el número de bytes a intercambiar, en este caso se necesitaran 8 bytes de entrada y 8 bytes de salida, para esto se elige la opción *8 Bytes Out / 8 Bytes In* y se le asigna el numero 20 como dirección inicial de entrada y salida (ver figura 35), luego hay que asignar las memorias V que servirán como entrada y salida en el PLC S7-200, para esto hay que dar click sobre el icono del EM277 e ir a *Parametrizar,* luego ir a *parámetros del equipo* y *Parámetros específicos del aparato,* en esta ventana aparecerá una opción que dice *I/O Offset in the V-memory,* al frente de esta opción se pone el número 80 (ver figura 36), esto quiere decir que las memorias de la VB80 a la VB87 sirven como entrada en el S7-200 y las memorias de la VB88 a la VB95 sirven como salida del S7-200.

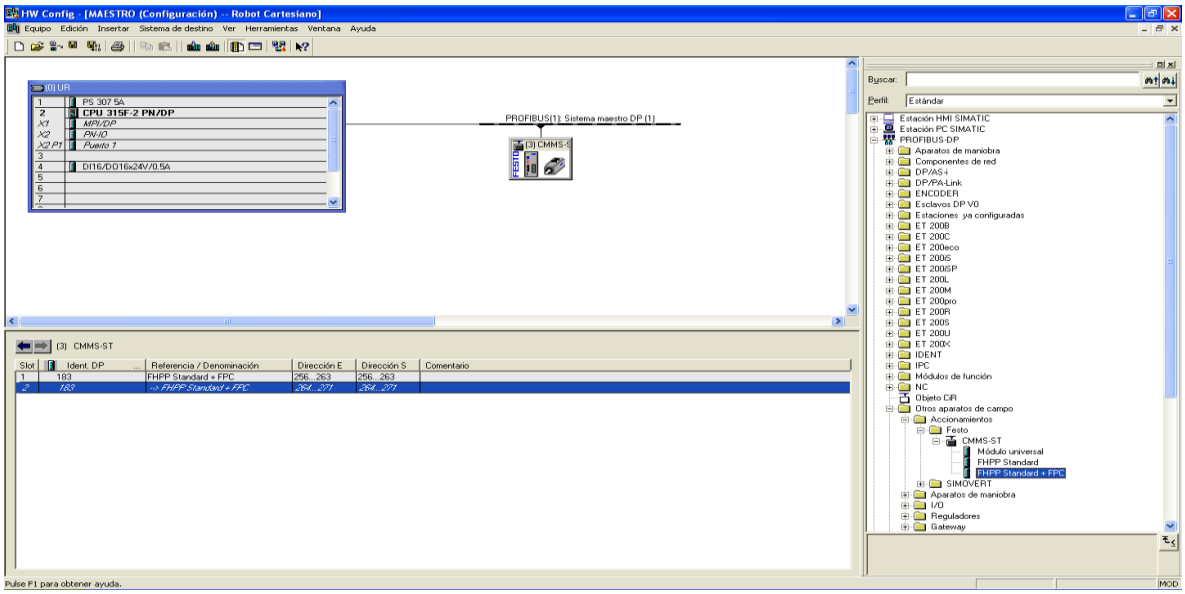
El intercambio de datos entre el S7-300 y el S7-200 queda definido de la siguiente forma, de la EB20 a la EB27 son las entradas en el PLC S7-300 que reciben los datos de las memorias V que sirven como salida del S7-200, es decir de la memoria VB88 a la VB95 y de la AB20 a la AB27 son las salidas en el PLC S7-300 que mandan los datos a las memorias que sirven como entrada del S7-200, es decir de la VB80 a la VB87.

El último elemento a anclar a la red Profibus es el variador de frecuencia Micro Master 440 el cual es utilizado para controlar el motor de la banda de la calle de selección. Este se encuentra ubicado en *PROFIBUS-DP, Otros aparatos de campo, Accionamientos, SIMOVERT y MICROMASTER 4,* al arrastrar el dispositivo a la línea Profibus aparecerá la ventana donde se ingresa la dirección Profibus, a este se le asignó la dirección 7[4] (ver figura 37). Luego hay que seleccionar modulo universal y luego el tipo de estructura de datos, en este caso se elige la *PPO3*, la cual da acceso solo al área de proceso, con los datos de

[4] La dirección Profibus también debe ser definida en el variador, para esto hay que digitarla en el parámetro P918.

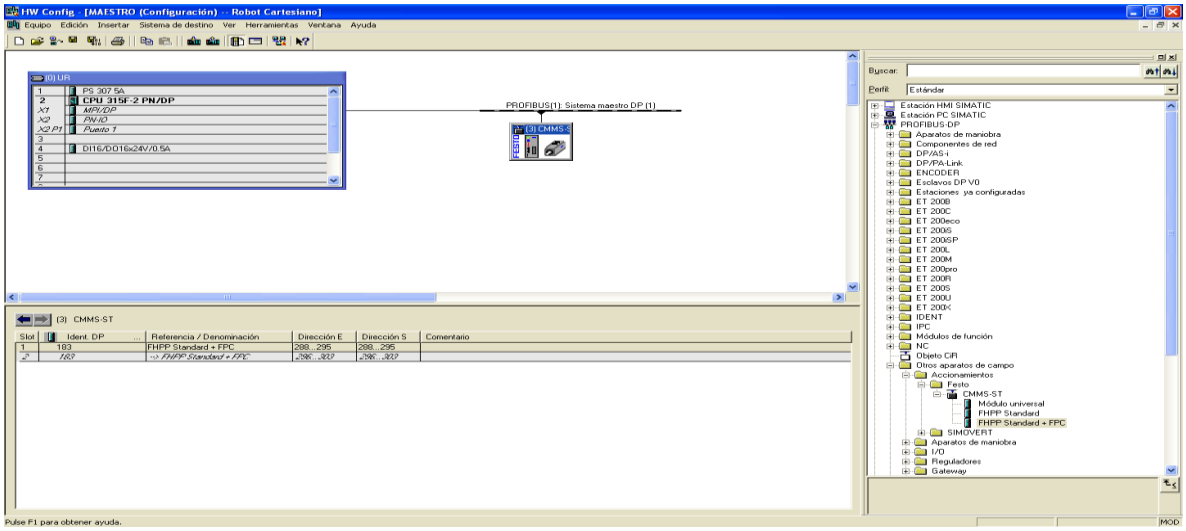
proceso se pueden transmitir palabras de mando y valores de consigna así como palabras de estado y valores reales. En el slot 3 se digita como dirección inicial de entrada y salida la 321 (ver figura 38).

Figura 30: *FHPP Standard + FPC*.



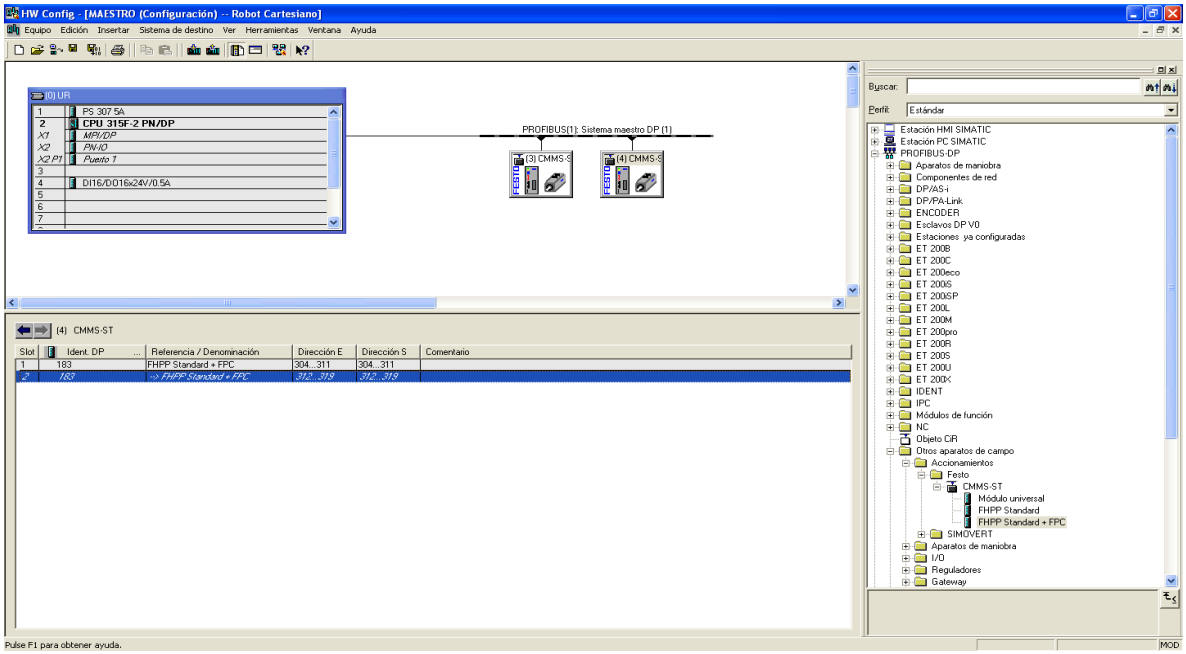
Fuente: Autor.

Figura 31: Direcciones entrada y salida.



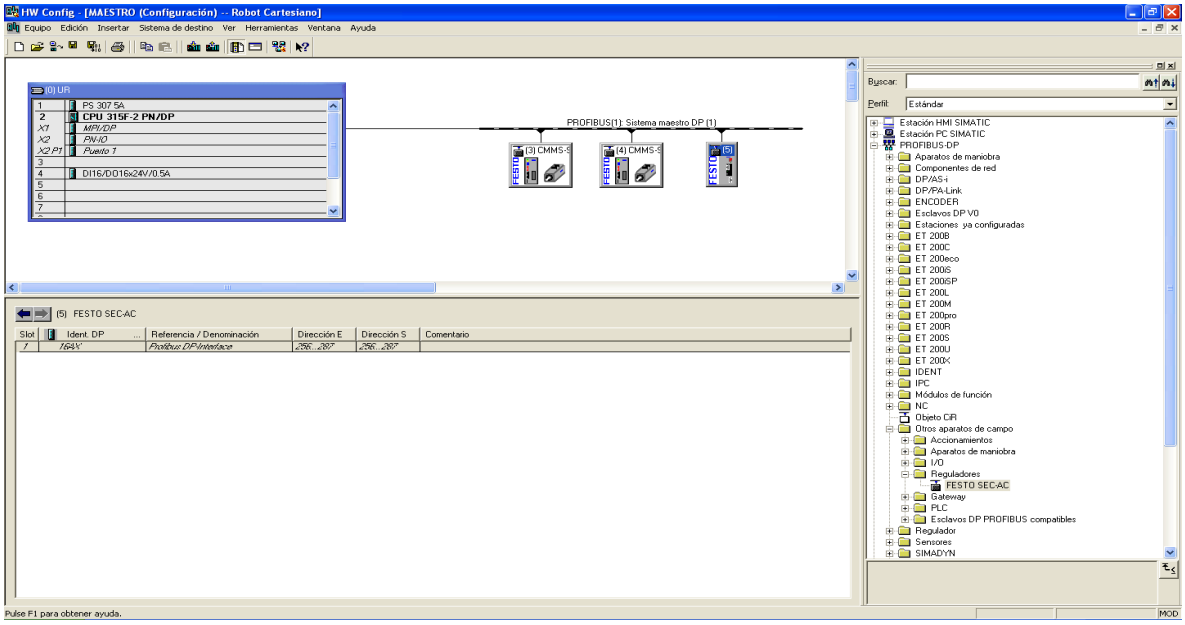
Fuente: Autor.

Figura 32: Tarjeta CMMS-ST eje y.



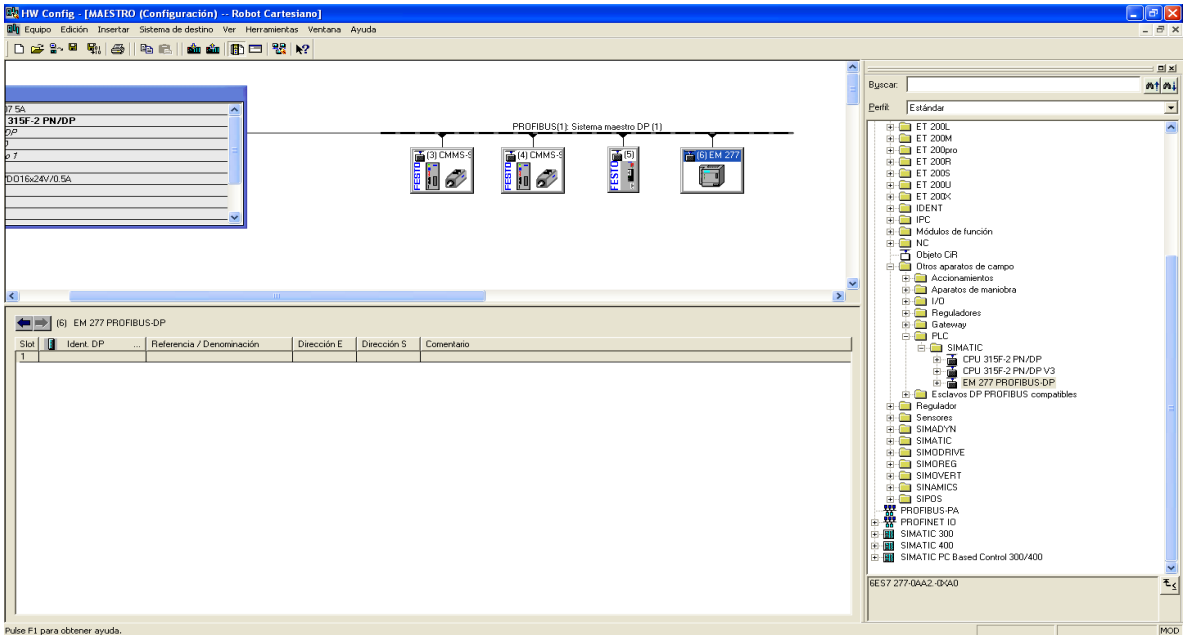
Fuente: Autor.

Figura 33: Tarjeta SEC-AC eje z.



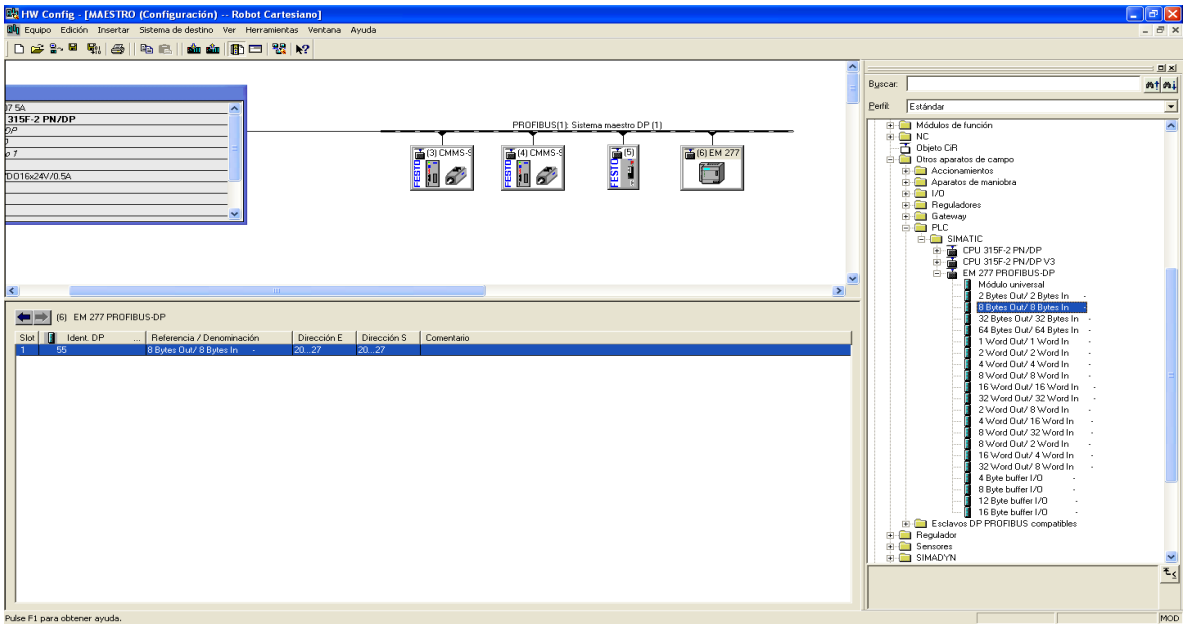
Fuente: Autor.

Figura 34: Modulo EM 277.



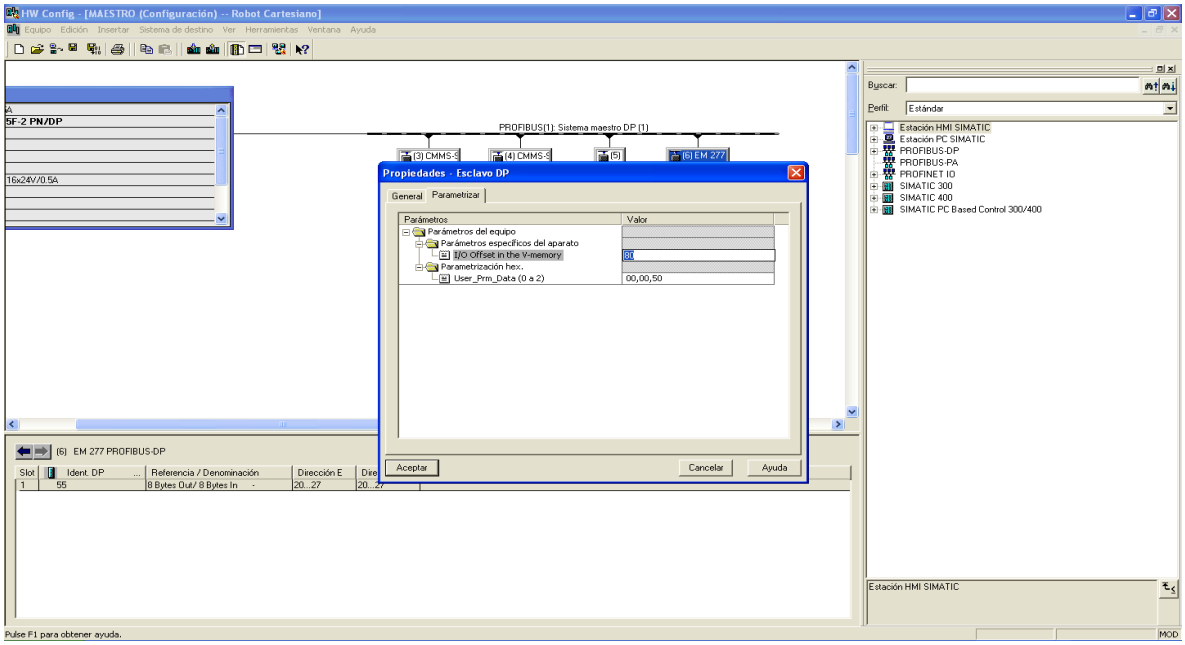
Fuente: Autor.

Figura 35: 8 Bytes Out / 8 Bytes In.



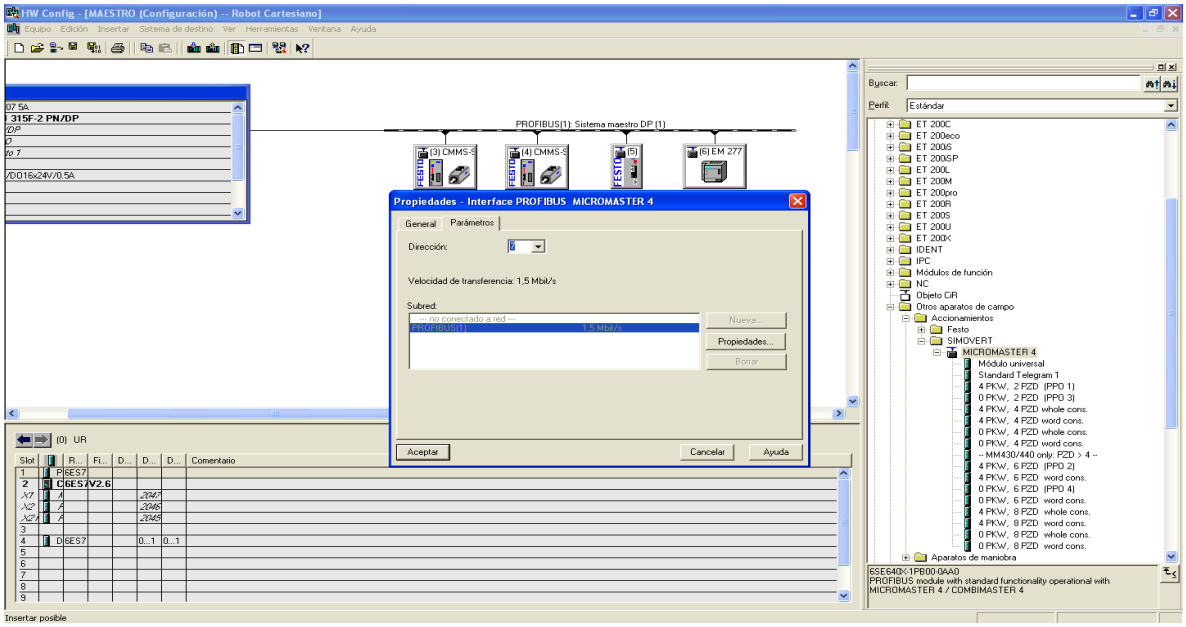
Fuente: Autor.

Figura 36: Área de memoria V para recibir y enviar datos desde la CPU S7-200.



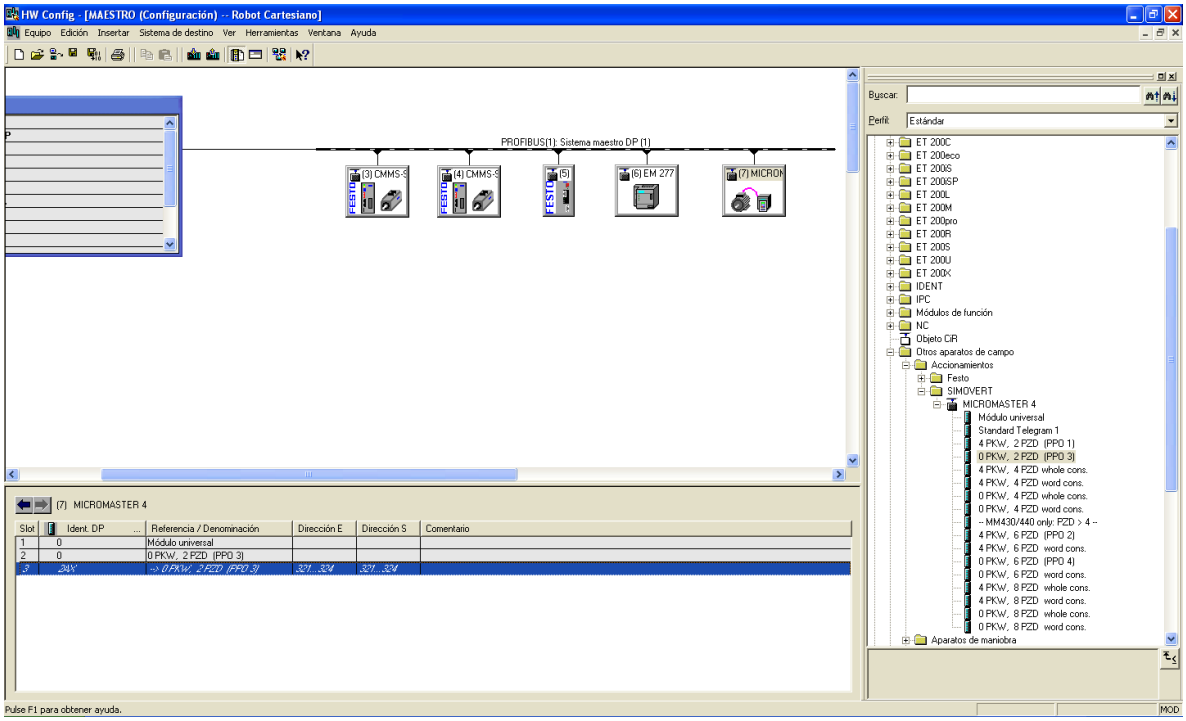
Fuente: Autor.

Figura 37: Dirección Profibus Micro Master 440.



Fuente: Autor.

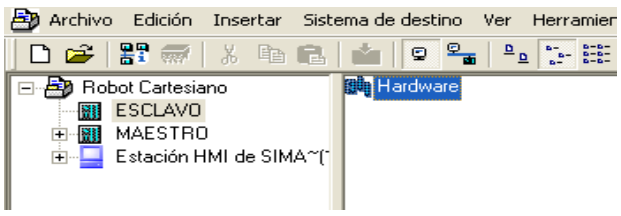
Figura 38: PPO3.



Fuente: Autor.

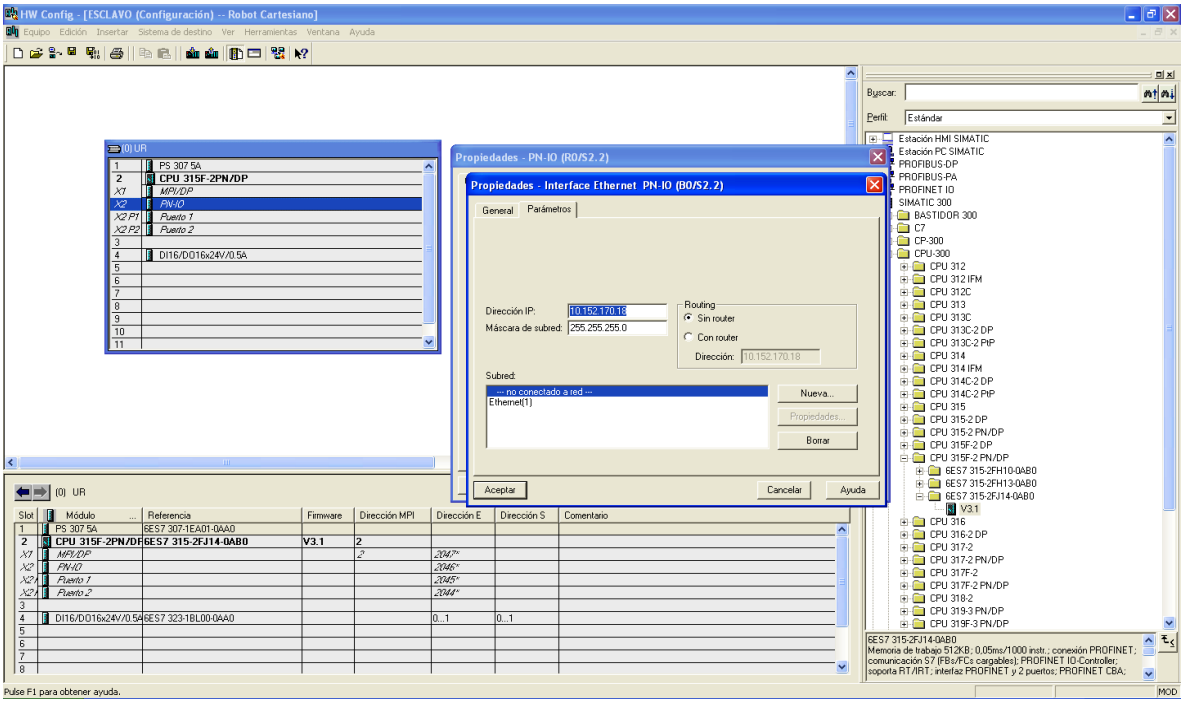
Como siguiente paso se procede a conectar el PLC *ESCLAVO* y la *Estación HMI* a la red Ethernet, para esto se empieza configurando el PLC *ESCLAVO* en *Hardware* (ver figura 39), los elementos a agregar en el bastidor son los mismo que se agregaron en la configuración del PLC *MAESTRO*, la diferencia es que la CPU cambia, ahora es la 6ES7 315-2FJ14-0AB0, al agregarla saldrá una ventana donde se debe definir la dirección IP la cual es **10.152.170.18** (ver figura 40).

Figura 39: Iniciando configuración PLC esclavo.



Fuente: Autor.

Figura 40: IP PLC esclavo.



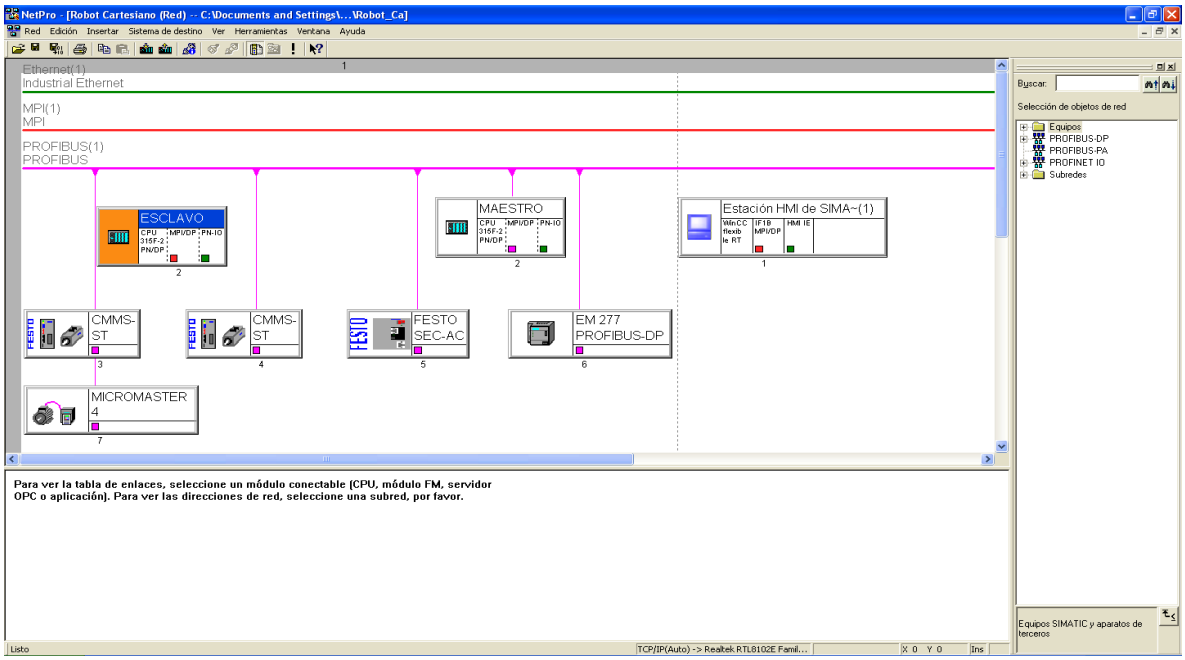
Fuente: Autor.

Una vez agregados todos los dispositivos se guarda y compila para que se asimilen los cambios y luego hay que dirigirse *NetPro* (ver figura 41).

Para unir los dispositivos a la red Ethernet solo hay que arrastrar el cuadro verde en el dispositivo hacia la red Ethernet ya creada (ver figura 42). Ahora hay que cambiar la dirección IP al panel táctil el cual tiene la dirección **10.152.170.13** (ver figura 43). Luego se debe crear el enlace entre los dos PLCs, para eso hay que dar click en *CPU 315F-2PN/DP* del *MAESTRO*, luego dar click derecho en la primera fila debajo de *ID local* y seleccionar *Insertar nuevo enlace* (ver figura #44), luego de esto se abre una ventana en donde se elige la estación con la que se quiere crear el enlace y el tipo de enlace, en este caso por defecto aparece seleccionado la CPU *ESCLAVO* y el *Enlace S7* que son los que se necesitan, así que se acepta y en la siguiente ventana se deja la dirección ID en 1 la cual

permite la comunicación entre los PLCs (ver figura 45), para finalizar se acepta y compila. Con esto la configuración de *Hardware* que da concluida, ahora el siguiente paso es crear el código de comunicación para cada dispositivo (ver figura 46)[5].

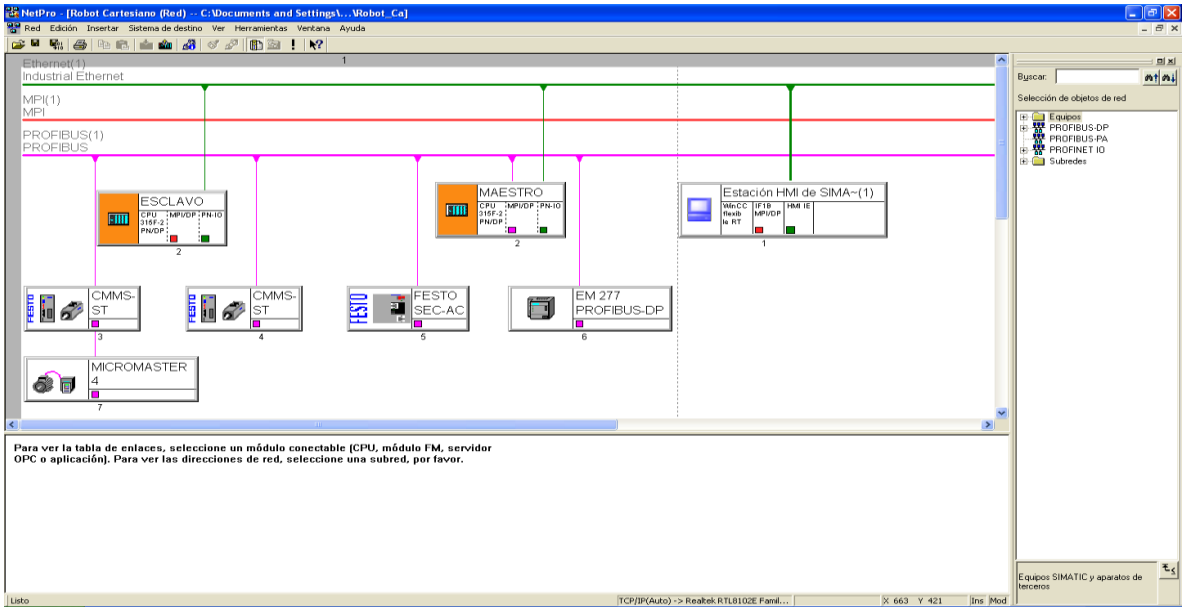
Figura 41: Conectando dispositivos desde NetPro.



Fuente: Autor.

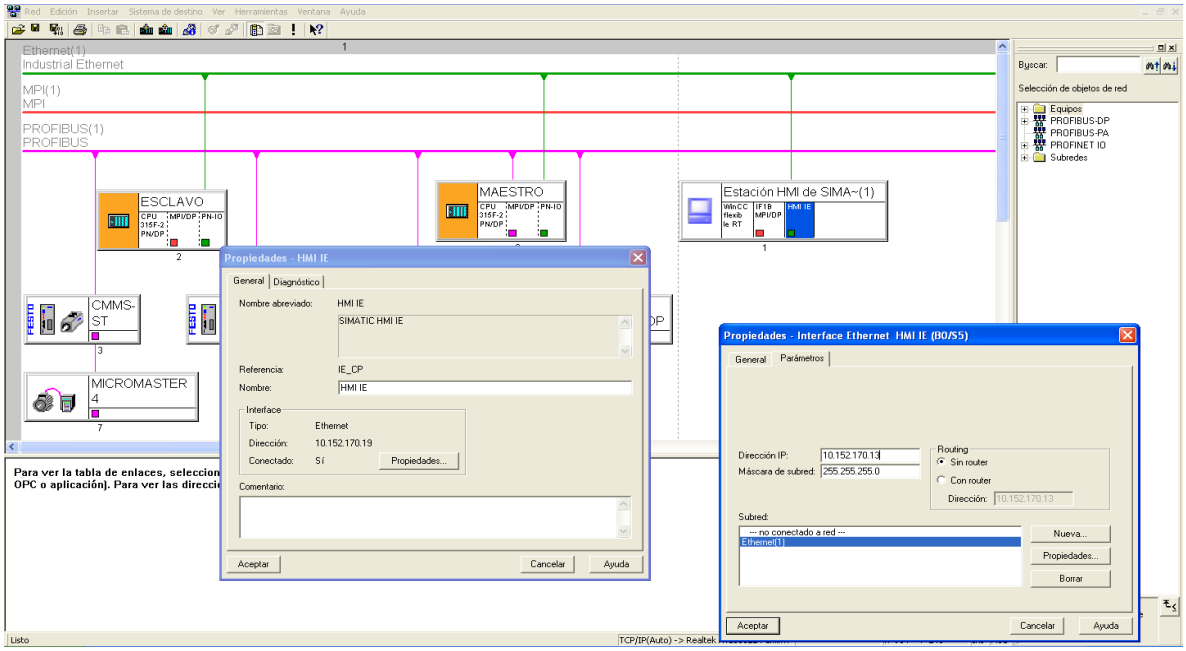
[5] Los documentos consultados para realizar la configuración de Hardware son las referencias [10], [11], [12], [13], [14], [15].

Figura 42: Dispositivos anclados.



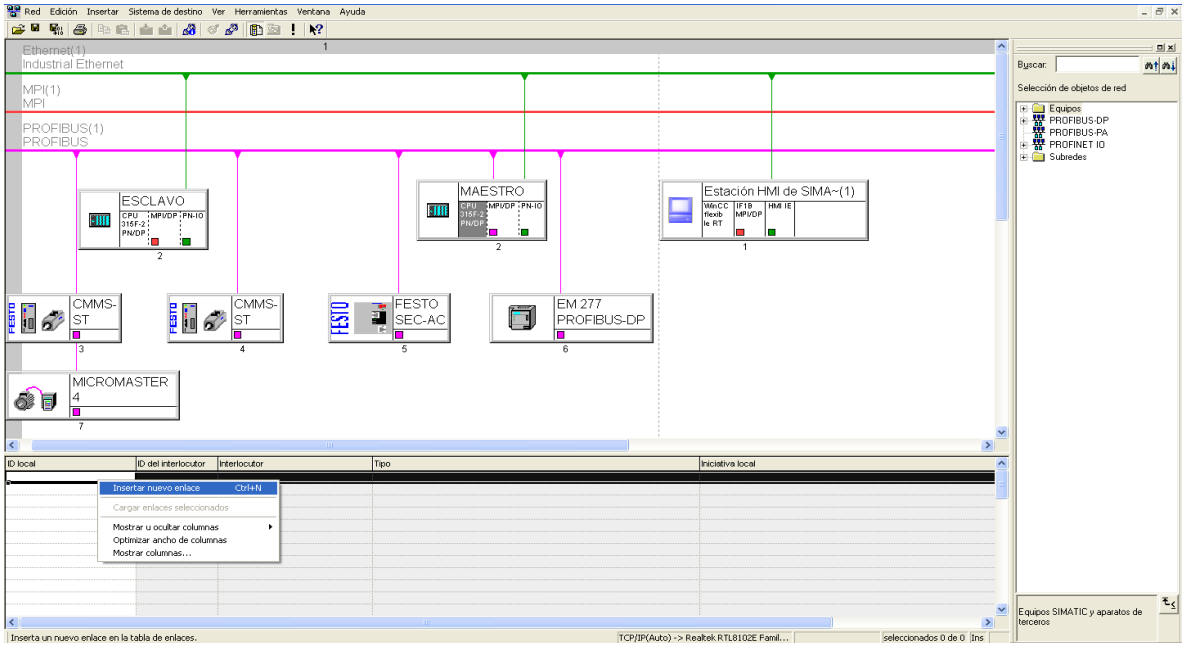
Fuente: Autor.

Figura 43: IP panel táctil.



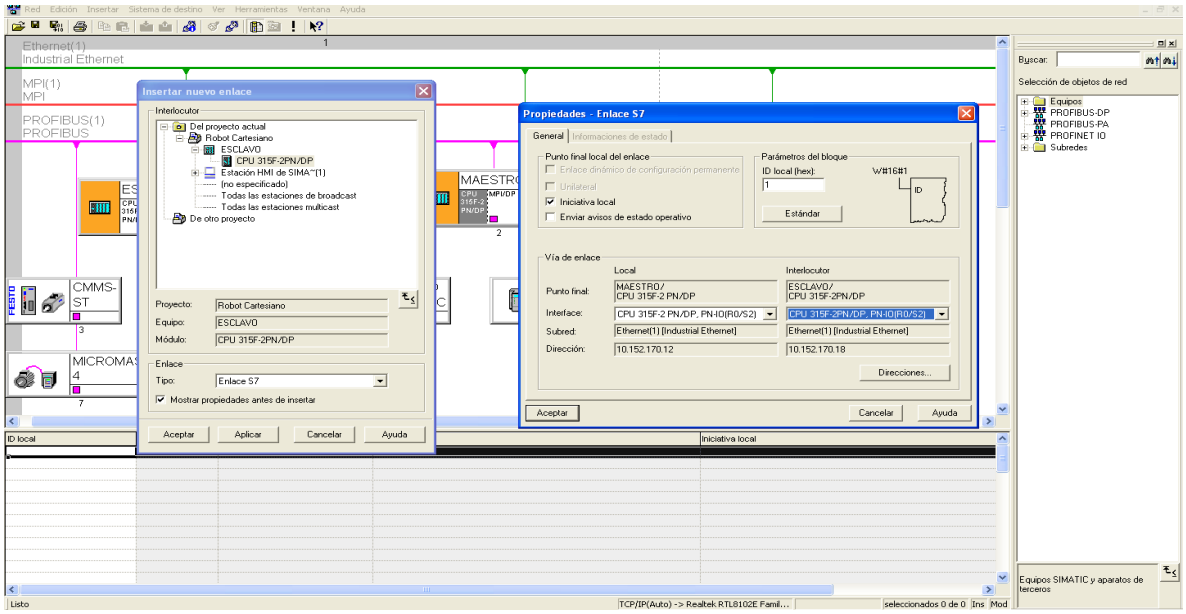
Fuente: Autor.

Figura 44: Enlace.



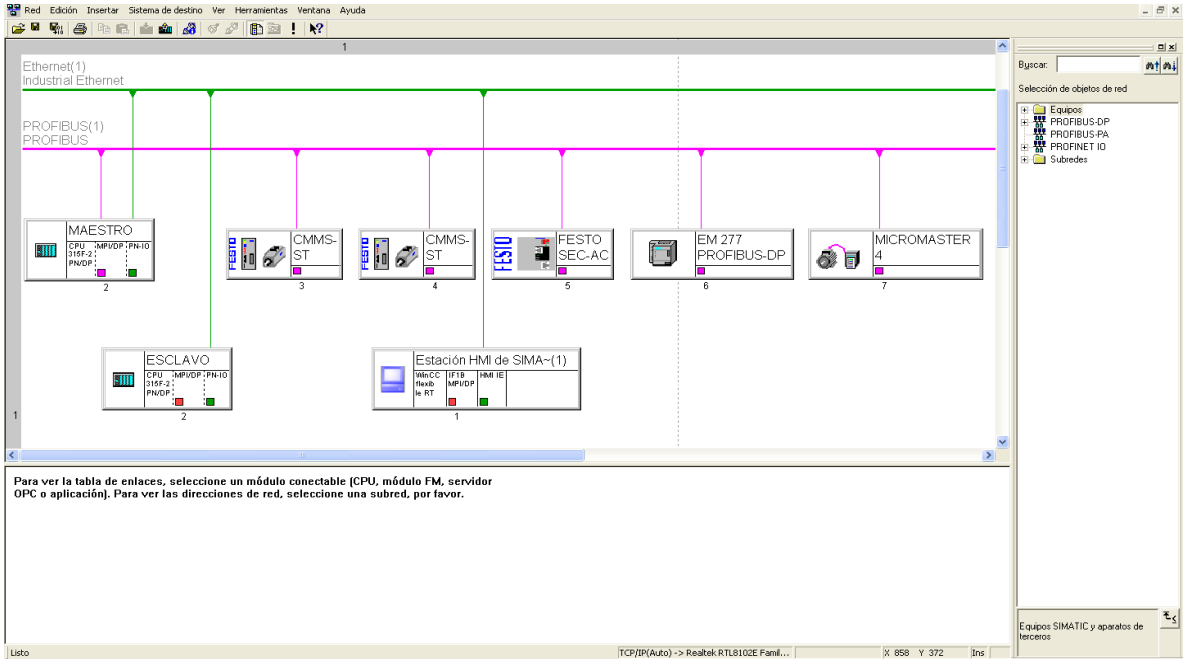
Fuente: Autor.

Figura 45: ID 1.



Fuente: Autor.

Figura 46: Configuración de Hardware concluida.



Fuente: Autor.

4. CÓDIGO DE COMUNICACIÓN

El código se realiza siguiendo el orden en que los dispositivos fueron agregados, entonces se comienza con la tarjeta CMMS-ST del eje-x.

Nos dirigimos al CD de FESTO a la carpeta *FB-Tools,* luego a *Profibus* y luego se descomprime el archivo *CMMS-ST_V1_0* el cual contiene el programa ejemplo de donde se sacaran los bloques. Una vez se haya abierto el programa de ejemplo se copian los bloques de la carpeta *Bausteine* en *_CTRL* y se pegan en el proyecto.

El siguiente paso es duplicar el bloque DB90 cambiándolo a DB91, luego el bloque DB10 cambiándolo a DB11, luego la tabla de variables *CMMS_ST_CONTROL* cambiándola primero de nombre a *CMMS_ST_CTR_DRIVE-EJE-X* y luego el duplicado a *CMMS_ST_CTR_DRIVE-EJE-Y*, por último la tabla de variables *CMMS_ST_OBSERVE* cambiándola primero de nombre a *CMMS_ST_OB-DRIVE-EJE-X* y el duplicado a *CMMS_ST_OB-DRIVE-EJE-Y.*

Los duplicados se deben a que se va a manejar dos motores paso a paso. El bloque FB10 es el bloque de función que se encarga de controlar la tarjeta del motor paso a paso del eje x, el bloque DB10 es el bloque de datos de instancia, el cual actúa como memoria del bloque FB10 guardando los parámetros formales, el bloque DB90 es el bloque de datos global el cual tiene como función guardar los parámetros actuales.

Es necesario que se haga coincidir los duplicados de las tablas de variables con los duplicados de los bloques de datos, para esto hay que cambiar las direcciones en las tablas de variables de DB90 a DB91 (ver figura 47).

El siguiente paso es abrir el OB1 y configurar los bloques de control, el primero a configurar es el que aparece por defecto, este se va a usar para el eje x y como ya viene configurado simplemente hay que asignarle la dirección inicial de entrada y de salida, esta es la 288, pero hay que introducirla en formato hexadecimal que

seria 120, así que se introduce el número 120 de la forma *W#16#120* en los parámetros formales I_ADDRESS y O_ADDRESS (ver figura 48).

El siguiente paso es sacar del menú de funciones de la carpeta *Bloques FB* el bloque de función 10 y arrastrarlo a un nuevo segmento, este es el bloque de control para el eje y, así que como tiene dirección inicial de entrada y salida 304, daría en hexadecimal 130 la cual se introduce en la forma *W#16#130,* es necesario cambiar las direcciones de los parámetros actuales de D90 a D91 y además la dirección del bloque de instancia de D10 a D11 (ver figura 49).

Ahora se procede de la misma forma a configurar la tarjeta del eje z, pasando el bloque de control el cual se cambia de nombre a FB12, el bloque de datos global el cual se cambia de nombre a DB92, el bloque de datos de instancia DB14 y la tabla de variable *Z control.* Hay que asegurarse que todos los parámetros que están en *Z control* empiezan por DB92, luego de esto nos dirigimos al OB1 y en un nuevo segmento arrastramos el bloque de función FB12 (ver figura 50), este se procede a configurar de la misma forma, se asigna la dirección del bloque de instancia que le corresponde, la cual es el D14, la dirección de entrada y salida que es la 256 la cual se escribe de la forma *W#16#100,* los parámetros actuales que son los que se encuentran en la tabla de variables *Z control,* el método de homing que es el 6 y el cual se escribe de la forma B#16#6, la constante de avance L#10000, entre otros (ver figura 51). Ahora se procede a realizar el código de comunicación para el módulo EM277.

Como sabemos, se definió como dirección inicial de entrada y salida en el S7-300 la numero 20 y como se dieron 8 bytes de entrada y salida se tienen disponibles las direcciones EB20-EB27 y AB20-AB27, ahora la necesidad es que las señales de los sensores en las bandas de la calle de selección sean reconocidas en el PLC maestro, estas señales son en el PLC S7-200 I5.0, I5.1, I5.2, I5.3, I5.4 e I5.5, así que hay que transferir el byte IB5 del S7-200 a las memorias MB95 del S7-300 que son las que se utilizaran como señales equivalente a estas entradas.

Primero nos dirigimos al OB1 y cambiamos de formato a AWL que es con el cual se realizara todo el programa y luego en un nuevo segmento escribimos el código para la dirección EB20 que es el primer byte por donde entraran los datos del primer byte de salida del S7-200, es decir de la memoria VB88 (ver figura 52). Esta dirección se debe a que en la configuración de *hardware* se definió como dirección inicial de memorias de almacenamiento la V80 y como se eligieron 8 bytes de entrada y 8 bytes de salida entonces las memorias de la VB80 a la VB87 son los 8 bytes de entrada y las memorias de la VB88 a la VB95 son los 8 bytes de salida.

El siguiente paso es realizar el movimiento del byte IB5 a la memoria de salida VB88, para esto abrimos el *STEP 7-MICRO/WIN* y escribimos el siguiente código[6] (ver figura 53), es importante usar la memoria especial SM0.0 como accionamiento para que la transmisión sea en todo ciclo de programa.

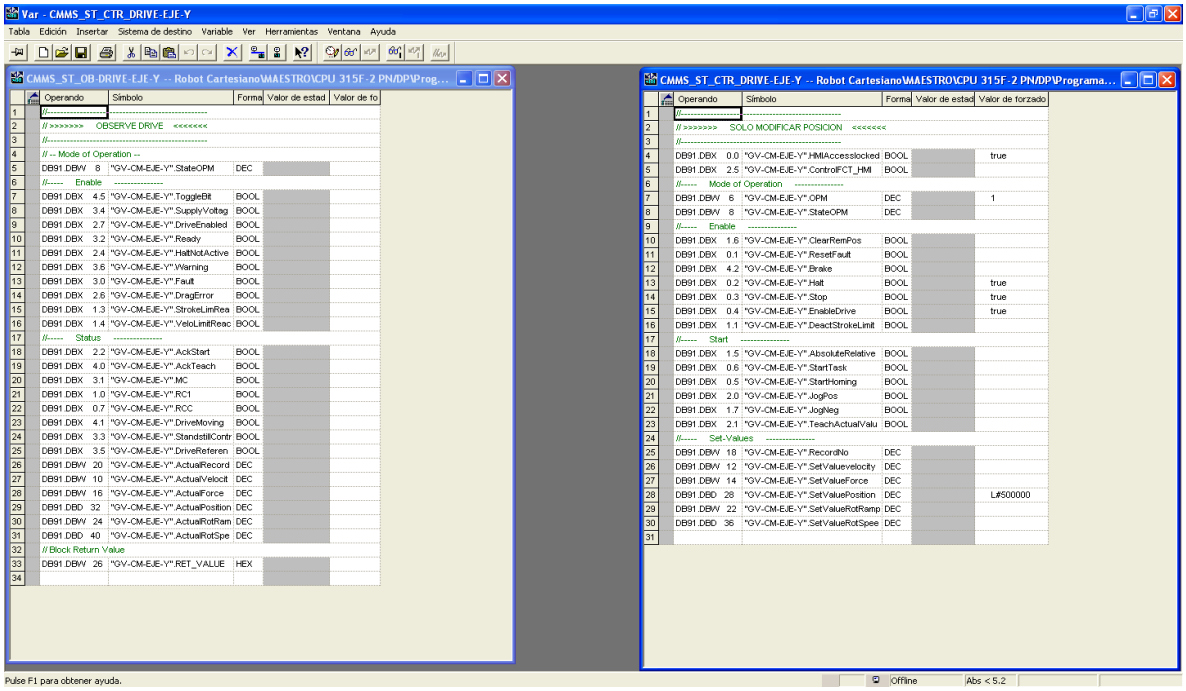
El siguiente elemento es el variador de frecuencia, el primer paso es extraer el programa de ejemplo que se encuentra en la carpeta *EFB* del CD que viene con el módulo MICROMASTER PROFIBUS (ver figura 54), luego se copian todos los bloques excepto el OB1, del OB1 se copia el código pero antes cambiamos las direcciones de los parámetros en la tabla de datos (ver figura 55) y luego se pega el código[7] (ver figura 56).

La dirección MD998 es la memoria que recibe el valor de frecuencia desde la HMI (ver figura 57), el parámetro actual *"drive_f_sent"* que es la memoria MD990 es el que define el valor de la frecuencia, pero este debe estar dado en real, así que se aplica el comando DTR que transforma un entero doble a real.

[6] Hay que tener en cuenta que este código debe ser agregado al programa de la calle de selección que es el que se le carga a la CPU S7-200.

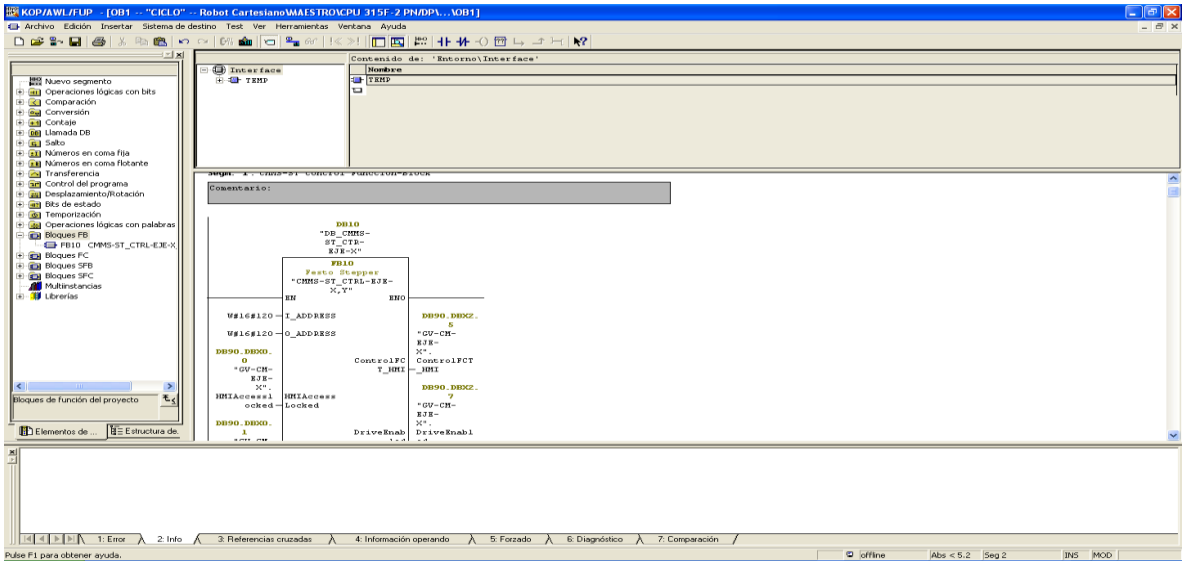
[7] Hay que tener en cuenta que en la tabla de símbolos deben estar asignados los nombre que aparecen en la tabla de variables del programa de ejemplo a las direcciones de las memorias cambiadas para que al pegar el código del bloque del MM440 no salgan errores, una solución rápida es copiar y pegar la tabla de símbolos y cambiar las direcciones.

Figura 47: Cambio de variables.



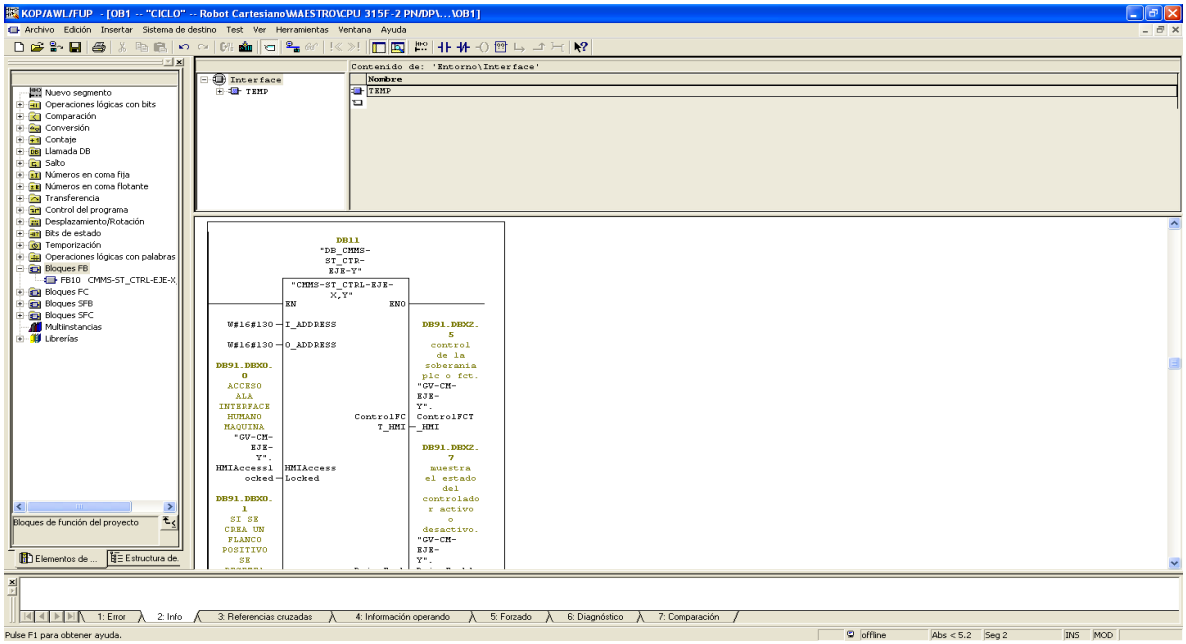
Fuente: Autor.

Figura 48: Dirección entrada y salida.



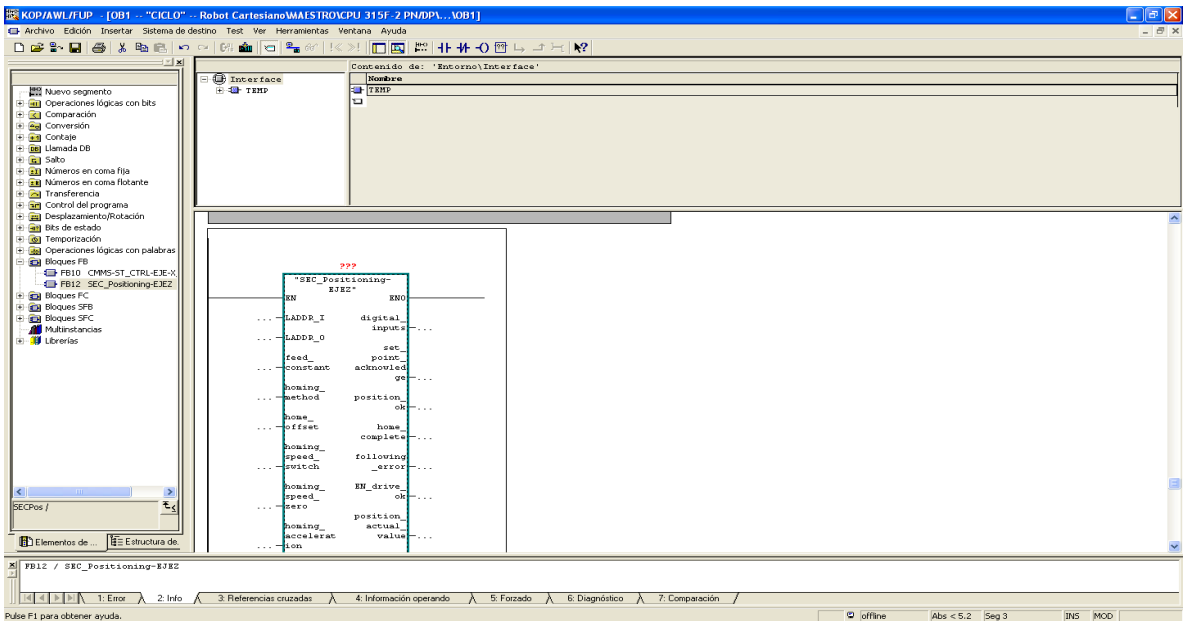
Fuente: Autor.

Figura 49: Configuración tarjeta eje-y.



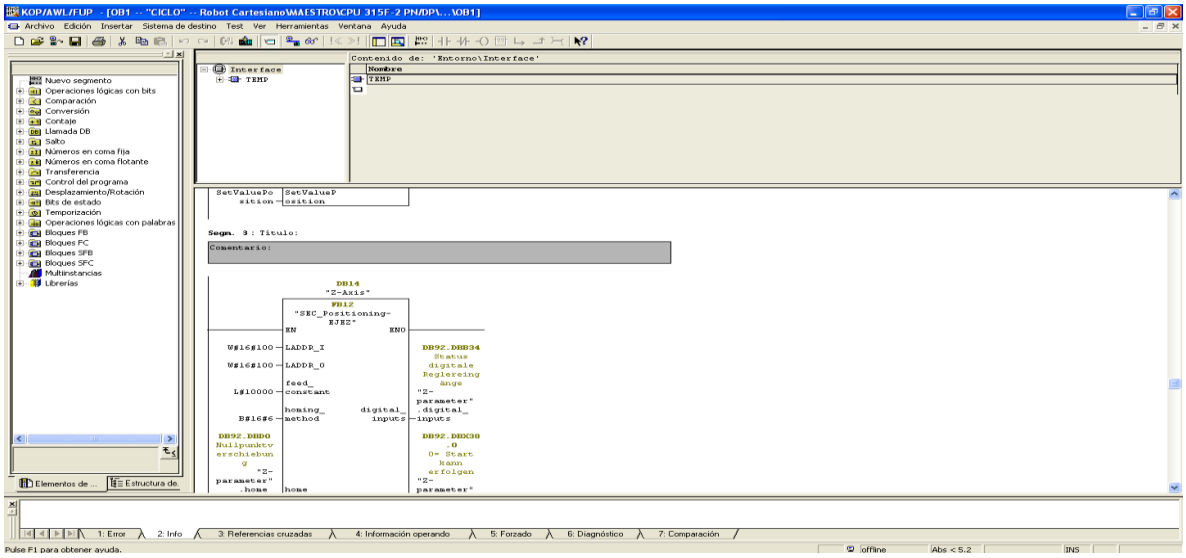
Fuente: Autor.

Figura 50: Configuración tarjeta eje-z.



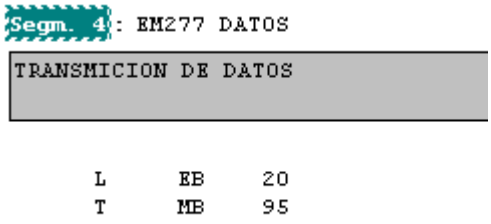
Fuente: Autor.

Figura 51: Configuración tarjeta eje-z terminada.



Fuente: Autor.

Figura 52: Código en Step7.

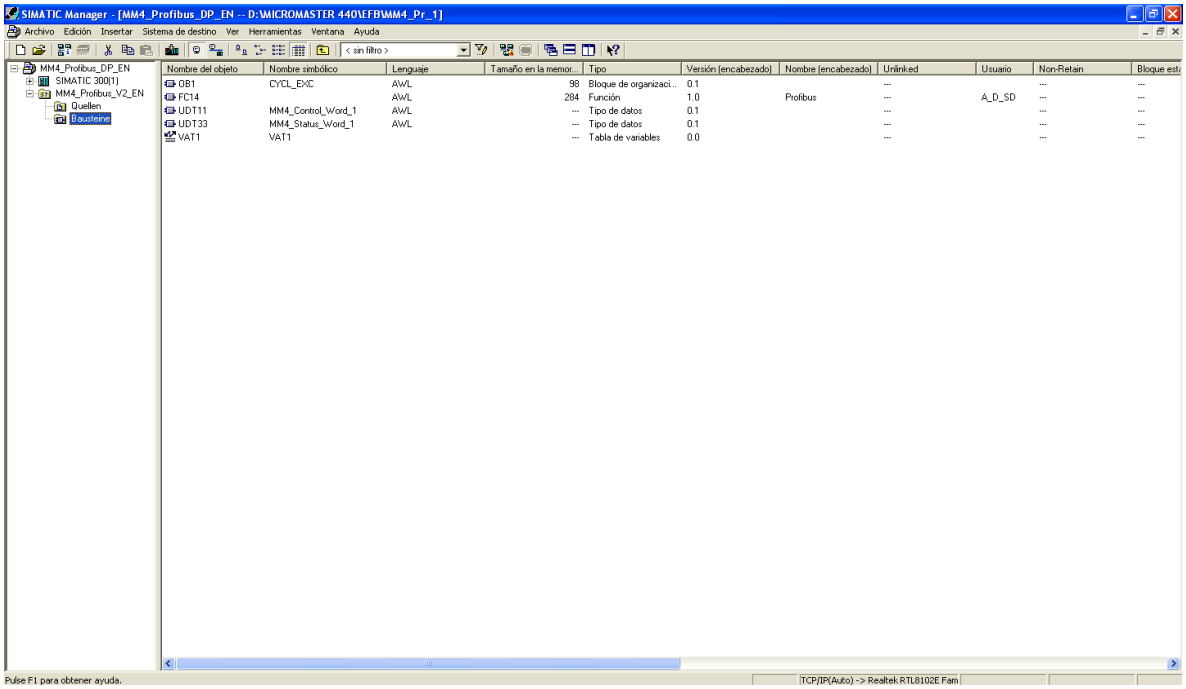


Fuente: Autor.

Figura 53: Código en STEP 7-MICRO/WIN.

Fuente: Autor.

Figura 54: Proyecto de ejemplo MM440.



Fuente: Autor.

Figura 55: Cambio de direcciones de la tabla de variables MM440.

Tabla Edición Insertar Sistema de destino Variable Ver Herramientas Ventana Ayuda

	Operando	Símbolo	Formato de visualización	Valor de estado	Valor de forzado
1	// Control bits				
2	M 15.0	"drive_on_off"	BIN		2#0
3	M 15.1	"drive_reversing"	BIN		2#0
4	M 15.2	"drive_fault_ack"	BIN		2#0
5	// Status bits				
6	M 15.3	"drive_in_operation"	BIN		
7	M 15.4	"drive_fault"	BIN		
8	M 15.5	"drive_alarm"	BIN		
9					
10	MD 990	"drive_f_sent"	REAL		0.0
11	MD 994	"drive_f_act"	REAL		
12					
13					
14					
15					
16					
17					

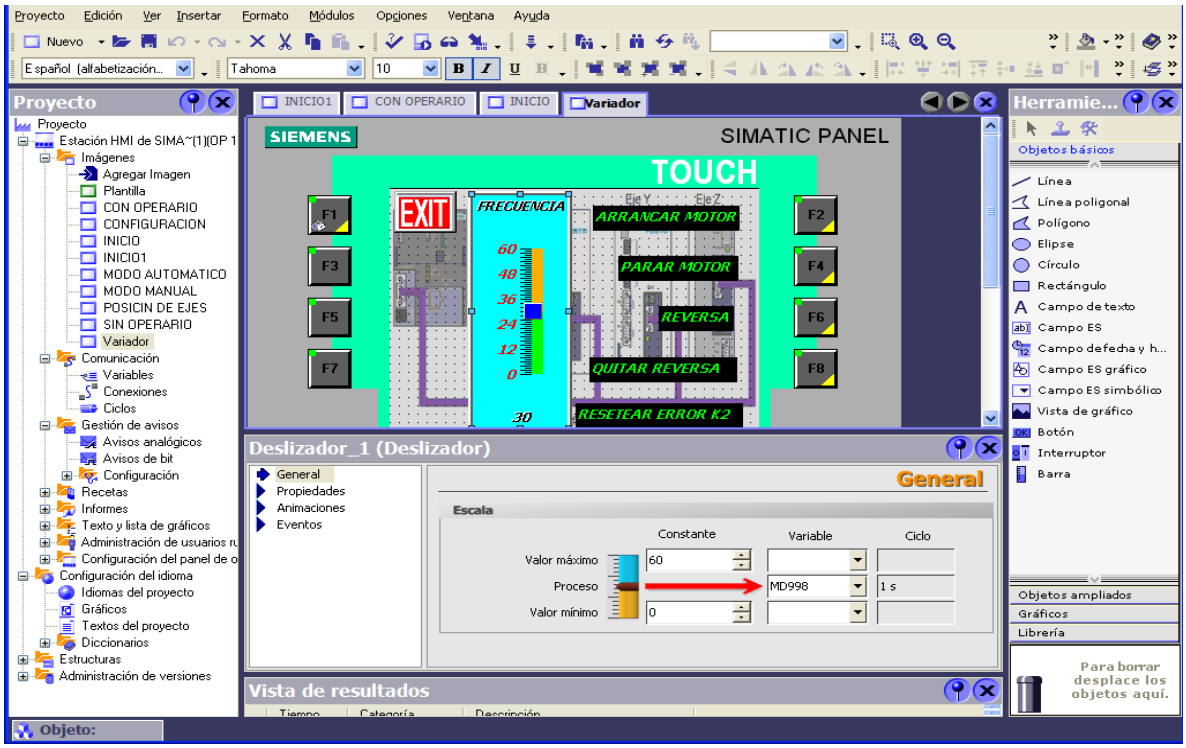
Fuente: Autor.

Figura 56: Código de comunicación MM440.

```
Segm. 5: LLAMAR FUNCION PARA MICROMASTER 440.
LLAMAR FUNCION PARA MICROMASTER 440.

      CALL   "FUNCION PARA MM440"                          FC14
       hw_config_I_O_address:=321
       drive_on_off          :="drive_on_off"              M15.0
       drive_reversing       :="drive_reversing"           M15.1
       drive_fault_ackn      :="drive_fault_ack"           M15.2
       drive_f_setpoint      :="drive_f_sent"              MD990
       drive_in_operation    :="drive_in_operation"        M15.3
       drive_fault_active    :="drive_fault"               M15.4
       drive_alarm_active    :="drive_alarm"               M15.5
       drive_f_act           :="drive_f_act"               MD994


      L      "MD998"                                       MD998
      DTR
      T      "drive_f_sent"                                MD990
```

Fuente: Autor.

Figura 57: Memoria MD998.



Fuente: Autor.

El último elemento es el PLC esclavo. Para comunicar este con el PLC maestro se usó el protocolo de comunicación Ethernet.

El primer paso es dirigirse al bloque de programa del PLC esclavo, luego nos dirigimos a mano izquierda a *librerías, Standard Library* y por ultimo a *Communicatio Blocks.* Ahora como el PLC esclavo que es el de la empacadora va a enviar los datos, entonces se utiliza la función poner o *PUT,* la cual aparece en la librería como *FB15 PUT CPU_300,* así que la arrastramos a la línea de programa que aparecerá por defecto, inmediatamente después de esto nos pedirá asignarle una memoria de almacenamiento, es decir un bloque de datos el cual en este caso se le asignó el DB1, luego pasamos el lenguaje de programación a AWL el cual es el que se utilizara para la creación del código y por último se le asignan las variables a utilizar.

El primer parámetro es *REQ* el cual es el bit que habilita la transmisión, a este se le asigna el bit *EXTRA2* el cual es una variable temporal creada en *TEMP* que se ubica en la lista superior de variables, este bit esta siempre activado debido a la negación de la variable temporal *EXTRA1*, la cual permanece siempre en cero, garantizando de esta forma mantener la transmisión de datos en todo ciclo de programa, el siguiente parámetro es ID que es la identificación del enlace, al principio en la configuración de hardware se declaró la dirección del enlace como la 1, así que en este espacio se asigna la dirección ya definida anteriormente como *W#16#1*, a continuación esta *DONE* el cual es el bit que indica si la transferencia de datos fue realizada de manera exitosa, en esta casilla le asignamos la variable temporal *HECHO,* el siguiente parámetro es *ERROR* el cual es el bit que indica si ha sucedido un error, a este se le asigna la variable temporal *ERROR,* el siguiente parámetro es *STATUS* el cual es una palabra en donde se muestra el número que identifica el tipo de error, a continuación encontramos el parámetro *ADDR_1* el cual indica la zona de datos de la CPU remota donde se escribirán los datos, como no se requiere esta retro alimentación entonces dejamos este parámetro abierto, el siguiente y ultimo parámetro de esta

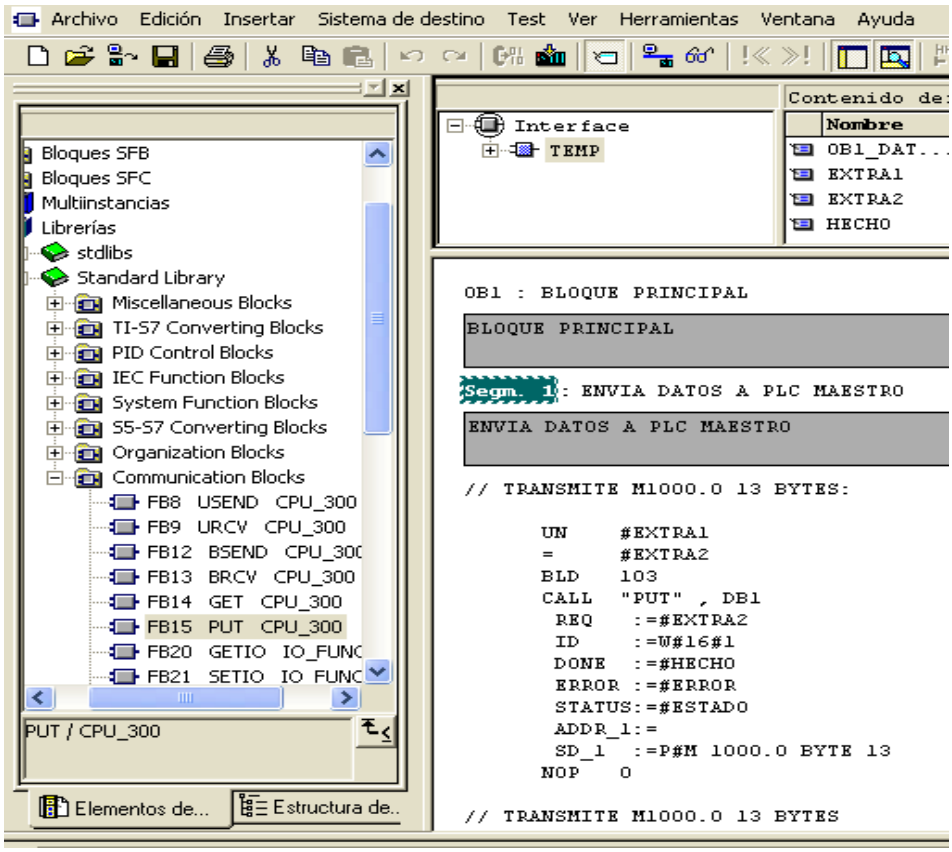
función es *SD_1,* este parámetro es donde se indica la dirección de los datos a enviar a la CPU remota, en este caso se requiere que el PLC esclavo envié el número de piezas de cada banda necesitadas en la empacadora, entonces se requiere una palabra para transmitir los datos de cada banda, así que como son 6 bandas esto daría 12 bytes y además también se requiere de un bit que indique la petición de la empacadora, de tal forma que se requieren 12 bytes y 1 bit, pero como solo se pueden transmitir bytes entonces se requiere 13 bytes, los cuales van a iniciar desde la memoria M1000.0. El parámetro *SD_1* está en formato ANY de tal forma que los 13 bytes se escriben de la siguiente forma: P#M 1000.0 BYTE 13 (ver figura 58).

El siguiente paso es configurar la función que recibe los datos en el PLC maestro, para esto se ejecuta el mismo procedimiento pero ahora se selecciona la función *FB14 GET CPU_300*, este bloque de función a diferencia del bloque *PUT* se ejecuta (es decir lee nuevos datos) solo si se presenta un flanco positivo en el parámetro *REQ,* entonces para esto se crea un tren de pulsos con las variables temporales *ACCESO, PULSOS Y ACCIONAMIENTO.* La variable acceso es accionada cuando se elige desde el panel táctil el modo real sin operario, de tal forma que el tren de pulsos solo se activa en este modo, la variable pulsos es la que se encarga de generar los pulsos para mantener la variable *ACCIONAMIENTO* prendiendo y apagando, de tal forma que se generan flancos positivos constantemente en el parámetro REQ. El siguiente parámetro es *NDR*, el cual indica el estado de la petición, 0 significa petición aún no iniciada o en curso y 1 significa petición realizada sin errores, a este parámetro se le asignó la variable temporal *PETICION,* a continuación encontramos el parámetro *ADDR_1* el cual indica aquellas áreas de la CPU remota que deben leerse, en este caso si se requiere por qué se necesita leer los datos enviados por la CPU esclavo, es decir que introducimos la misma área de memoria asignada en el parámetro *SD_1* de la función *PUT* (P#M 1000.0 BYTE 13), el siguiente y ultimo parámetro es el *RD_1* el cual indica las áreas de memoria de la CPU propia en donde se depositan los

datos leídos, en este caso se depositaran desde la memoria M18.0, así que escribimos el área de memoria como P#M 18.0 BYTE 13 [8].

Para finalizar el código de comunicación se realiza la transferencia de datos de las memorias que recibieron los datos de la CPU esclavo (desde la M18.0 hasta la M30.7) a las memorias creadas en la base de datos general para almacenar el número de piezas necesarias de cada banda y la señal de petición, es decir de la memoria DBW268 a la DBX280.0 del DB80, esto se realiza con el comando *L* para cargar la variable y el comando *T* para descargar el dato de la variable cargada a la variable que almacena (ver figura 59).

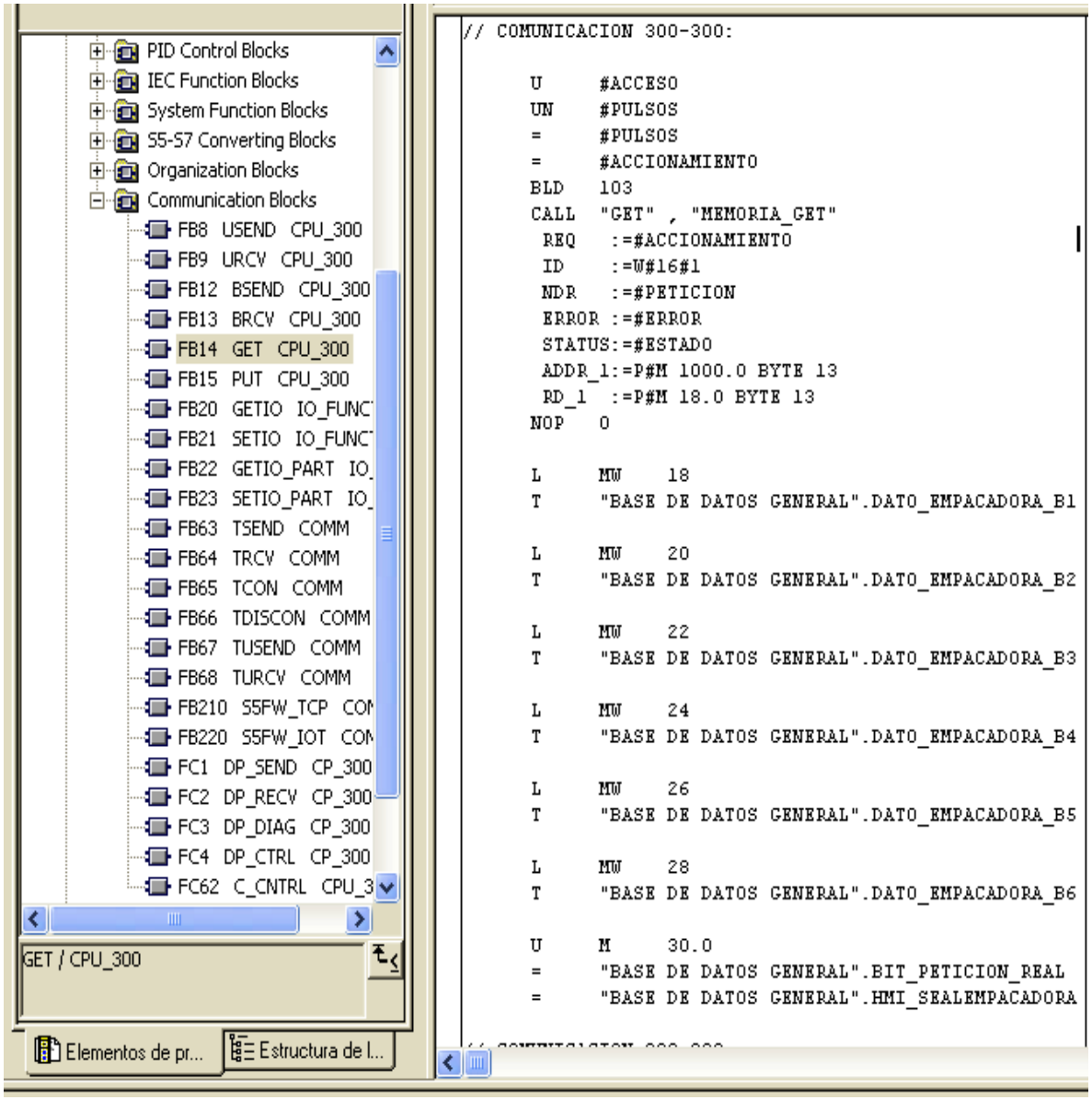
Figura 58: Configuración FB15 PUT CPU_300.



Fuente: Autor.

[8] Los demás parámetros no se explican debido a que son los mismos de la función *PUT*.

Figura 59: Configuración FB14 GET CPU_300.



Fuente: Autor.

Con esto se concluyen los códigos de comunicación. El siguiente paso es realizar el programa el cual desarrolla las rutinas y sub rutinas requeridas[9].

[9] Los documentos consultados para realizar el código de comunicación son las referencias [12], [13], [14], [15], [16].

5. CÓDIGO DE TRABAJO

El código de trabajo es el que manipula los movimientos del robot, en este punto se describirá cada función que conforma la programación del robot cartesiano.

Después de los códigos de comunicación, en el mismo OB1 continuamos con la conversión de parámetros, en este punto la idea es que las velocidades y aceleraciones puedan ser introducidas desde el panel táctil en unidades de cm/s y cm/s^2 respectivamente.

Entonces básicamente las palabras y palabras dobles que obtienen los datos del panel táctil, son multiplicadas por un factor con el cual se llega a las unidades originales que los bloques de función de las tarjetas aceptan. Los factores son 10 para las velocidades de los ejes y, x y 10000 para la velocidad, aceleración y desaceleración del eje z. estos valores se deben a que la conversión de las velocidades de los ejes x, y es de cm/s a mm/s y la conversión de la velocidad, aceleración y desaceleración del eje z es de cm/s a µm/s y de cm/s^2 a $\mu m/s^2$.

Las variables que reciben del panel táctil y las que cargan a los bloques de función de las tarjetas se crearon en el bloque de datos DB80, que es la base de datos general.

Para llevar a cabo estas conversiones se utilizaron los comandos cargar *L,* multiplicar entero **I,* multiplicar doble entero **D* y transferir *T.*

Estos comandos se utilizaron de la siguiente manera, primero se carga la variable que obtiene el dato de velocidad o aceleración, luego se carga el factor de conversión, luego se escribe la operación (**D o *I* dependiendo de los datos) y por último se transfiere este resultado a la variable que carga el dato al bloque de función de la tarjeta (ver figura 60).

El siguiente segmento contiene el código de inicio. Como primer paso se verifica un error que suele suceder en el robot cartesiano, este ocurre cuando se conmuta la parada de emergencia física mientras se encuentra en movimiento.

El error consiste en que cuando se inicia de nuevo el recorrido de referencia después de haber conmutado por alguna razón la parada de emergencia, la señal de recorrido hecho que da anclada como si ya se hubiera realizado, entonces si el eje z quedo extendido después de esta parada, el sistema cartesiano realizaría el recorrido de referencia con el eje z extendido, de esta manera causaría la destrucción total de la máquina. Entonces la primera parte del código de inicio se creó con la finalidad de evitar justamente ese daño

Figura 60: Conversión de unidades.

```
CONVERCION DE PARAMETROS

// VELOCIDADES EJES CONVERSION:

      L     "BASE DE DATOS GENERAL".VELOCIDAD_EJE_X_CM_S       DB80.DBW170
      L     10
      *I
      T     "BASE DE DATOS GENERAL".VELOCIDAD_EJE_X_MM_S       DB80.DBW172

      L     "BASE DE DATOS GENERAL".VELOCIDAD_EJE_Y_CM_S       DB80.DBW174
      L     10
      *I
      T     "BASE DE DATOS GENERAL".VELOCIDAD_EJE_Y_MM_S       DB80.DBW176

      L     "BASE DE DATOS GENERAL".VELOCIDAD_EJE_Z_CM_S       DB80.DBD178
      L     10000
      *D
      T     "BASE DE DATOS GENERAL".VELOCIDAD_EJE_Z_MIC_S      DB80.DBD182

      L     "BASE DE DATOS GENERAL".ACELERACION_EJE_Z_CM_SC    DB80.DBD186
      L     10000
      *D
      T     "BASE DE DATOS GENERAL".ACELERACION_EJE_Z_MIC_SC   DB80.DBD190

      L     "BASE DE DATOS GENERAL".DESACELERACION_EJEZ_CMSC   DB80.DBD194
      L     10000
      *D
      T     "BASE DE DATOS GENERAL".DESACELERACION_EJEZMICSC   DB80.DBD198

// VELOCIDADES EJES CONVERSION.
```

Fuente: Autor.

El código de seguridad dentro del código de inicio funciona de la siguiente manera. Primero se compara la posición actual del eje z, si este está por encima del punto de seguridad entonces la variable temporal *EXTRA1* permanece en 0 lógico, pero si este se encuentra por debajo del punto de seguridad entonces la variable temporal *EXTRA1* cambia a 1 lógico, al mismo tiempo se pregunta el estado de la variable *"Z-parameter".homing_attained* si está en 1 quiere decir que el recorrido de referencia ya se realizó, cosa tal que causaría el daño si es justo antes de dar inicio y si está en 0 quiere decir que no se ha realizado de tal forma que subiría automáticamente el eje con el recorrido de referencia y no habría problema. El 0 y 1 lógico de esta señal se reflejan en las variables temporales *EXTRA2* y *EXTRA8.* La variable *EXTRA2* es para la parte de seguridad y la variable *EXTRA8* es para habilitar el modo automático.

Si las dos condiciones están activas, es decir si el eje z esta abajo y además la señal de recorrido de referencia está en 1 entonces se encenderán dos bits los cuales son las variables temporales *EXTRA3* y *EXTRA4* (ver figura 61).

Figura 61: Código de seguridad en inicio.

```
// COMPARACION DE ERROR EJE Z:

      L     "Z-parameter".position_actual_value
      L     L#-10187
      <=D
      =     #EXTRA1

      U     "Z-parameter".homing_attained
      =     #EXTRA2
      =     #EXTRA8
                                  TEMP: BOOL
      U     #EXTRA1
      U     #EXTRA2
      =     #EXTRA3
      =     #EXTRA4

// COMPARACION DE ERROR EJE Z.
```

Fuente: Autor.

Para realizar la comparación se utilizan los comandos cargar *L,* comparación de entero doble en *ACU2* es menor que entero doble en *ACU1 <=D,* resultado = y compuerta lógica AND *U.*

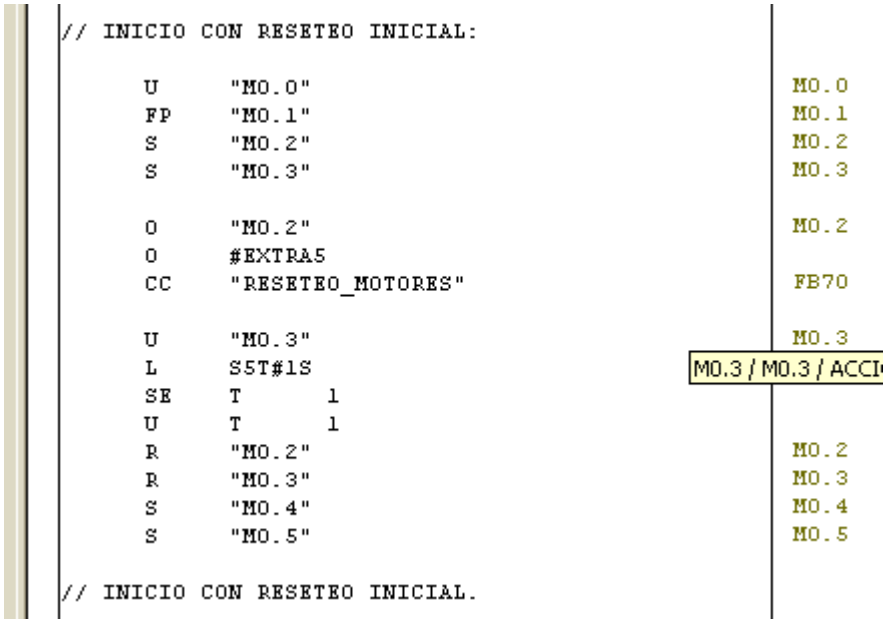
La variable temporal *EXTRA3* bloquea la llamada a la función *"INICIO MOTORES"* de tal forma que si está presente este error los motores no iniciaran. La variable temporal *EXTRA4* acciona la función *"SUBIDA_DE_HEMERGENCIA"* la cual corrige este error arrancando únicamente el eje z y subiéndolo un poco por encima del punto de seguridad.

Después de esto en el mismo segmento de inicio se continúa con el código de inicio con reseteo inicial, el cual funciona de la siguiente manera, primero desde el panel táctil se acciona la memoria M0.0 la cual da inicio a todo el programa, la memoria M0.1detecta el flanco positivo de M0.0, al detectarse el flanco positivos se ancla la memoria M0.2 y la memoria M0.3. La memoria M0.2 activa la llamada de la función *"RESETEO_MOTORES",* la memoria M0.3 activa el temporizador *T1* el cual trabaja en modo con retardo a la conexión, de esta forma el tiempo en que se mantiene la función reseteo activada es 1s pues es el tiempo de conteo del temporizador.

Para llevar a cabo este tramo de código se utilizan los comandos compuerta lógica AND *U,* detectar flanco positivo *FP,* anclar bit *S,* compuerta lógica OR *0,* llamada condicional de función *CC* y temporización, teniendo primero un accionamiento del temporizador *U "M0.3",* luego la carga del tiempo *L S5T#1S,* luego el modo de funcionamiento del temporizador *SE T1* y por último la salida después del conteo *U T1* (ver figura 62).

La memoria M0.0 se deja anclada de manera que no se genere una parada inesperada a causa de que se pulse de nuevo inicio, así que se puede comenzar de nuevo si primero se resetea todo desde el panel táctil.

Figura 62: Código de inicio con reseteo inicial.



Fuente: Autor.

Una vez terminado el conteo de 1s se resetean las memorias M0.2 y M0.3, de esta forma la función reseteo se deshabilita y además mediante el flanco se hace posible el reseteo de estas ya que este solo da un pulso en vez de mantener la señal constante como pasaría con una compuerta lógica AND.

Al terminar el conteo de 1s se anclan las memorias M0.4 y M0.5. La memoria M0.4 activa el temporizador *T2* si el error del eje z no está presente, si así es entonces el temporizador contara 500ms y luego se resetearan las memorias M0.4 y M0.5 y además se anclara la memoria M0.6, la cual llama la función *"INICIO MOTORES"* , de tal forma que se inicia el recorrido de referencia y luego se habilita el modo automático, pero si el error del eje z está presente entonces la memoria M0.5 accionara la llamada a la función *"SUBIDA_DE_HEMERGENCIA",* de tal forma que el eje subirá por encima del punto de seguridad y entonces se deshabilitara la llamada de esta función y se habilitara la temporización de *T2,* esto se debe a que

al subir, las memorias EXTRA3 y EXTR4 cambian a cero ya que el eje z estaría por encima del punto de seguridad (ver figura 63).

Figura 63: Verifica error eje z.

```
// VERIFICA ERROR EJE Z ( INICIA O SUBE ) :

      U     "M0.4"                                  M0.4
      UN    #EXTRA3
      L     S5T#500MS
      SE    T       2
      U     T       2
      R     "M0.4"                                  M0.4
      R     "M0.5"                                  M0.5
      S     "M0.6"                                  M0.6

      U     "M0.6"                                  M0.6
      CC    "INICIO MOTORES"                        FB71

      U     "M0.5"                                  M0.5
      U     #EXTRA4
      CC    "SUBIDA_DE_HEMERGENCIA"                 FB72

// VERIFICA ERROR EJE Z ( INICIA O SUBE ).
```

Fuente: Autor.

Para finalizar el segmento inicio, seguimos con reseteo total, en esta parte del código se pretende que mediante un botón en el panel táctil se pueda resetear los motores y algunas memorias fundamentales.

El bit que será accionado desde el panel táctil es la memoria M0.7, esta a su vez acciona tres variables temporales *EXTRA5, EXTRA6* y *EXTRA7.* La variable *EXTRA5* llama la función *"RESETEO_MOTORES",* la variable *EXTRA6* llama la función *"RESETEO_TOTAL"* y por último la variable *EXTRA7* acciona el temporizador *T7* el cual define el tiempo en que las funciones reseteo estarán activas, en este caso 1 s (ver figura 64).

El siguiente segmento contiene el código para el modo automático. Para que el modo automático esté disponible el robot cartesiano tiene que haber realizado el recorrido de referencia y cuando esto sucede se accionan las memorias *"GV-CM-EJE-X".MC, "GV-CM-EJE-Y".MC* y *"Z-parameter".homing_attained,* de modo que

cuando estos tres bits están en 1 lógico el modo automático se puede habilitar, así que para lograr esto se realiza una compuerta lógica AND con estas tres señales, solo que la señal del eje z se cambia por la variable temporal *EXTRA8* la cual anteriormente se definió como accionada por la variable *"Z-parameter".homing_attained.* Si estas tres señales están en 1 lógico entonces la memoria M4.0 se ancla y da señal a las variables temporales *CONOPREAL, CONOPSIMU, SINOPREAL y SINOPSIMU,* de esta forma se habilitan los cuatro modos de funcionamiento que desde ese momento quedan dependientes del bit asignado para cada modo de operación el cual es activado desde el panel táctil. La memoria M3.6 corresponde al modo real con operario, la memoria M3.7 corresponde al modo de simulación con operario, la memoria M4.1 corresponde al modo real sin operario y la memoria M4.2 corresponde al modo de simulación sin operario. Una vez seleccionado un modo se activaran dos variables; una memoria y una variable temporal, la variable temporal llama la función *"SELECCIONAR_MODO"* en donde mediante la memoria activada se selecciona la meta correspondiente al modo de funcionamiento (ver figura 65).

El siguiente segmento está hecho para la carga de posiciones en los puntos usados para el movimiento en los diferentes modos de operación. Como son 6 bandas, 6 recepciones y punto común, entonces en total hay 13 puntos modificables. La idea de esta parte del código es poder cambiar las coordenadas de los puntos de trabajo, ya que puede presentarse en un futuro que la estructura de la calle de selección tenga cambios o simplemente que esta sea movida por alguna razón, entonces mediante este código y la función *"CARGA_POCISIONES"* estos puntos se pueden modificar.

Figura 64: Reseteo total.

```
// RESETEO TOTAL:

      U     "M0.7"
      S     #EXTRA5
      S     #EXTRA6
      S     #EXTRA7

      U     #EXTRA6
      CC    "RESETEO_TOTAL"

      U     #EXTRA7
      L     S5T#1S
      SE    T       6
      U     T       6
      R     #EXTRA5
      R     #EXTRA6
      R     #EXTRA7

// RESETEO TOTAL.
```

Fuente: Autor.

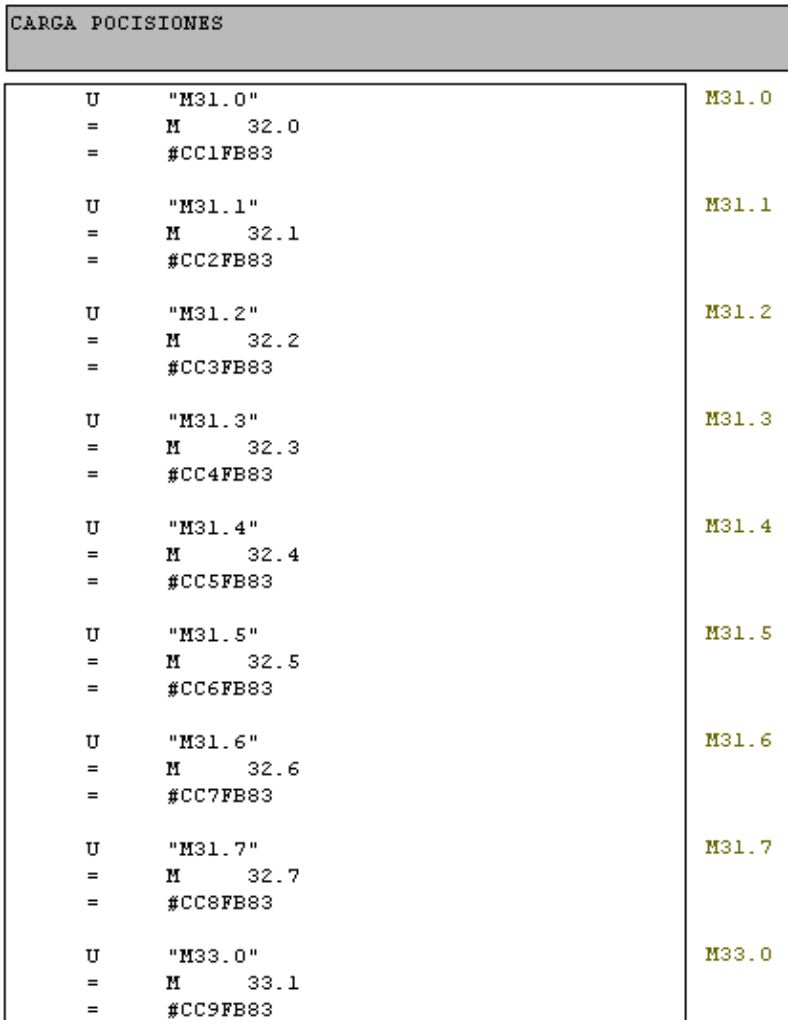
Figura 65: Modo automático.

```
// MODO AUTOMATICO:

      U     #EXTRA8
      U     "GV-CM-EJE-Y".MC                        DB91.DBX3.1
      U     "GV-CM-EJE-X".MC                        DB90.DBX3.1
      S     "M4.0"                                  M4.0
      S     "BASE DE DATOS GENERAL".HOMING_HECHO    DB80.DBX280.2

      U     "M4.0"                                  M4.0
      =     #CONOPREAL
      =     #CONOPSIMU
      =     #SINOPREAL
      =     #SINOPSIMU

      U     #CONOPREAL
      U     "M3.6"                                  M3.6
      =     "M3.2"                                  M3.2
      =     #SELECCION_MODO1

      U     #CONOPSIMU
      U     "M3.7"                                  M3.7
      =     "M3.3"                                  M3.3
      =     #SELECCION_MODO2

      U     #SINOPREAL
      U     "M4.1"                                  M4.1
      =     "M3.4"                                  M3.4
      =     #ACCESO
      =     #SELECCION_MODO3

      U     #SINOPSIMU
      U     "M4.2"                                  M4.2
      =     "M3.5"                                  M3.5
      =     #SELECCION_MODO4

      O     #SELECCION_MODO1
      O     #SELECCION_MODO2
      O     #SELECCION_MODO3
      O     #SELECCION_MODO4
      CC    "SELECCIONAR_MODO"                      FB81
```

Fuente: Autor.

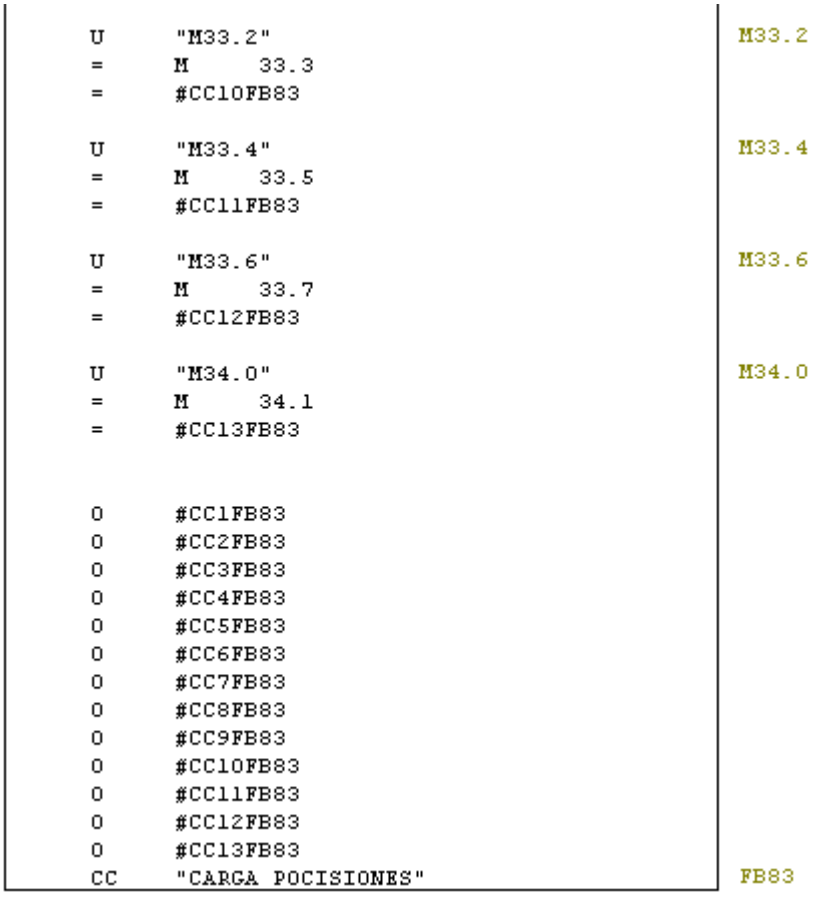
El funcionamiento de esta parte del código es el siguiente, primero mediante el panel táctil el robot se mueve hasta las nuevas coordenadas del punto que se desea modificar, luego de esto desde el mismo panel se activa un bit el cual a su vez activara 2 variables, estas son una memoria y una variable temporal, las variables temporales definidas en esta parte del código tienen la función de llamar el bloque o subrutina *"CARGA_POCISIONES"* mediante una compuerta lógica OR, la memoria efectúa su trabajo dentro de esta subrutina en donde básicamente genera un salto asía la meta que corresponde al punto elegido (ver figura 66, 67).

Figura 66: Cargar posiciones.



Fuente: Autor.

Figura 67: Cargar posiciones continúa.



Fuente: Autor.

Con esto se termina el código realizado en el OB1. El siguiente paso es revisar con de talle las funciones que dentro del OB1 son llamadas, para empezar nos dirigimos al bloque de función FB70 que es el mismo *RESETEO_MOTORES.*

Con esta subrutina se pretende básicamente poner en cero todas las áreas de memorias accionadas para poner en marcha los motores, además aquí también se resetean los errores en las tarjetas, de esta forma se garantiza que antes de que los motores se pongan en marcha no haya algún error guardado en las tarjetas controladoras.

Para llevar acabo esto se utilizan variable temporales denominadas *DIRECCION (#),* como están siempre en 0 lógico entonces su negación es usada para activar el reseteo de los bits utilizados en el arranque de cada motor.

Para poner en cero las palabras y las palabras dobles utilizadas, cargamos el valor 0 en ACU1 mediante el comando carga *L* y luego lo transferimos a las direcciones de las palabras utilizadas mediante el comando *T* (ver figura 68).

Figura 68: Reseteo motores.

```
//RESETEO MOTORES:

// EJE Y:

      UN    #DIRECCION1
      R     "GV-CM-EJE-Y".HMIAccesslocked
      R     "GV-CM-EJE-Y".Halt
      R     "GV-CM-EJE-Y".Stop
      R     "GV-CM-EJE-Y".EnableDrive
      R     "GV-CM-EJE-Y".StartTask
      R     "BASE DE DATOS GENERAL".FIN_DE_MOVIMIENTO

      UN    #DIRECCION2
      =     "GV-CM-EJE-Y".ResetFault

      L     0
      T     "GV-CM-EJE-Y".SetValuevelocity
      T     "GV-CM-EJE-Y".OPM
      T     "GV-CM-EJE-Y".StateOPM

// EJE X:

      UN    #DIRECCION3
      R     "GV-CM-EJE-X".HMIAccesslocked
      R     "GV-CM-EJE-X".Halt
      R     "GV-CM-EJE-X".Stop
      R     "GV-CM-EJE-X".EnableDrive
      R     "GV-CM-EJE-X".StartTask

      UN    #DIRECCION4
      =     "GV-CM-EJE-X".ResetFault

      L     0
      T     "GV-CM-EJE-X".OPM
      T     "GV-CM-EJE-X".StateOPM
      T     "GV-CM-EJE-X".SetValueVelocity

// EJE Z:

      UN    #DIRECCION5
      R     "Z-parameter".enable_drive
      R     "Z-parameter".start_homing_operation
      R     "Z-parameter".new_set_point
      R     "Z-parameter".homing_attained

      UN    #DIRECCION6
      =     "Z-parameter".reset_fault

      L     0
      T     "Z-parameter".profile_velocity
      T     "Z-parameter".profile_acceleration
      T     "Z-parameter".profile_deceleration

//RESETEO MOTORES.
```

Fuente: Autor.

La siguiente subrutina a analizar es el bloque de función *FB71* el cual fue denominado como *INICIO MOTORES.*

En este bloque de función se desarrolla el código orientado al accionamiento de los motores, el cual consiste en la carga de las variables de accionamiento y el desarrollo del recorrido de referencia.

El funcionamiento de esta subrutina es el siguiente. Una vez se haya llamado el bloque de función, se cargan los valores para poner en marcha los motores, es decir las velocidades de los ejes, la aceleración, la desaceleración y la velocidad de trote del eje z, los bits de accionamiento y el modo de funcionamiento de los ejes y, x el cual en este caso es modo 1, al mismo tiempo inicia la secuencia para el desarrollo del recorrido de referencia, primero la negación de la variable temporal *EXTRA4* genera un flanco positivo debido a la llamada de la función, este flanco es detectado por la variable temporal *FLANCO* la cual al detectar este flanco ancla la variable temporal *EXTRA5,* entonces se inicia un conteo de 1s con el temporizador *T3,* este tiempo de un segundo se da para que se defina el estado de las variables *EXTRA1 y EXTRA2.* La variable *EXTRA1* resulta en 1 lógico cuando el eje z está por encima del punto de seguridad y la variable *EXTRA2* resulta en 1 lógico si el recorrido de referencia en el eje z fue realizado de forma exitosa, entonces si *EXTRA1 y EXTRA2* están en 1 lógico quiere decir que el eje z está en la zona de seguridad y el recorrido de referencia se ejecutó con éxito.

Luego del conteo de 1s hecho por el temporizador *T3* se resetea la variable temporal *EXTRA5* y se ancla la variable *EXTRA7* y el bit *"Z-parameter".start_homing_operation,* el cual da inicio al recorrido de referencia en el eje z, la variable *EXTRA7* habilita el conteo de 500ms del temporizador *T4* el cual se da con finalidad de que si el eje z presento el error anteriormente mencionado y fue corregido entonces el bit *"Z-parameter".start_homing_operation* no sea anclado y des anclado al mismo tiempo. Desde el momento en que la variable *EXTRA7* es accionada, el conteo del temporizador *T4* queda dependiendo

de las variable *EXTRA1 y EXTRA2,* de tal forma que se garantiza que los ejes x, y no se muevan a menos que el eje z haya realizado el recorrido de referencia y este en la zona de seguridad. Una vez las tres variables estén en 1 lógico el temporizador *T4* comienza el conteo de 500ms y luego se resetea la variable *EXTRA7* y se anclan la variables *EXTRA8 y EXTRA9.* La variable *EXTRA8* da inicio al recorrido de referencia de los ejes X y Y accionando los bits *"GV-CM-EJE-X".StartHoming y "GV-CM-EJE-Y".StartHoming,* al mismo tiempo se resetea la señal de iniciar recorrido del eje z, es decir *"Z-parameter".start_homing_operation,* la variable EXTRA9 da inicio al conteo de 2s del temporizador *T5,* de esta manera las señales de iniciar recorrido quedan accionadas 2s.

La señal de recorrido de referencia del eje z se mantiene activa durante todo el recorrido ya que si esta se quita en algún punto de este, el eje z parara. A diferencia de este bit, las señales de recorrido de referencia de los ejes X y Y no deben estar en 1 lógico durante todo el recorrido ya que solo basta con un flanco positivo, de ahí que se da el temporizador *T5* de 2s para finalizar la secuencia (ver figura 69 y 70).

Figura 69: Inicio motores.

```
// VERIFICA EJE Z:

      L     "Z-parameter".position_actual_value
      L     L#-10187
      >=D
      =     #EXTRA1

      U     "Z-parameter".homing_attained
      =     #EXTRA2

// EJE Y:

      L     "BASE DE DATOS GENERAL".VELOCIDAD_EJE_Y_MM_S
      T     "GV-CM-EJE-Y".SetValuevelocity

      UN    #EXTRA3
      =     "GV-CM-EJE-Y".HMIAccesslocked
      =     "GV-CM-EJE-Y".Halt
      =     "GV-CM-EJE-Y".Stop
      =     "GV-CM-EJE-Y".EnableDrive
      =     "BASE DE DATOS GENERAL".HMI_ENTRABAJO

      L     1
      T     "GV-CM-EJE-Y".OPM
      T     "GV-CM-EJE-Y".StateOPM

      UN    #EXTRA4
      FP    #FLANCO
      S     #EXTRA5

      U     #EXTRA5
      L     S5T#1S
      SE    T       3
      U     T       3
      R     #EXTRA5
      S     #EXTRA7
      S     "Z-parameter".start_homing_operation
```

Fuente: Autor.

Figura 70: Inicio motores.

```
// EJE X:

      L     "BASE DE DATOS GENERAL".VELOCIDAD_EJE_X_MM_S
      T     "GV-CM-EJE-X".SetValueVelocity

      UN    #EXTRA6
      =     "GV-CM-EJE-X".HMIAccesslocked
      =     "GV-CM-EJE-X".Halt
      =     "GV-CM-EJE-X".Stop
      =     "GV-CM-EJE-X".EnableDrive

      L     1
      T     "GV-CM-EJE-X".OPM
      T     "GV-CM-EJE-X".StateOPM

      U     #EXTRA1
      U     #EXTRA2
      U     #EXTRA7
      L     S5T#500MS
      SE    T      4
      U     T      4
      R     #EXTRA7
      S     #EXTRA8
      S     #EXTRA9

      U     #EXTRA8
      =     "GV-CM-EJE-Y".StartHoming
      =     "GV-CM-EJE-X".StartHoming
      R     "Z-parameter".start_homing_operation

      U     #EXTRA9
      L     S5T#2S
      SE    T      5
      U     T      5
      R     #EXTRA8
      R     #EXTRA9

// EJE Z:

      L     "BASE DE DATOS GENERAL".VELOCIDAD_EJE_Z_MIC_S
      T     "Z-parameter".profile_velocity

      L     "BASE DE DATOS GENERAL".ACELERACION_EJE_Z_MIC_SC
      T     "Z-parameter".profile_acceleration

      L     "BASE DE DATOS GENERAL".DESACELERACION_EJEZMICSC
      T     "Z-parameter".profile_deceleration

      L     L#300000
      T     "Z-parameter".jogging_velocity

      UN    #EXTR10
      =     "Z-parameter".enable_drive
```

Fuente: Autor.

El siguiente bloque a analizar es el *FB72* el cual es llamado *SUBIDA_DE_HEMERGENCIA.*

El código en este bloque tiene la finalidad de subir automáticamente el eje z al punto de seguridad. Este es necesario en dos ocasiones, la primera es si se presenta el error de eje z el cual se ha mencionado anteriormente, entonces antes

de iniciar los recorridos de referencia el programa verifica si es necesario utilizar esta función, la segunda ocasión es si el eje z se dejó abajo tras haber cargado las coordenadas a los puntos de trabajo, es decir si el eje z se dejó por debajo del punto de seguridad cuando se movió en modo manual y además se entra a algún modo de operación, entonces al hacer llegar la orden de realizar movimiento, el eje subirá usando esta función.

La subida de emergencia funciona de la siguiente forma. Una vez dentro de la subrutina todas las variables necesarias para el movimiento del eje z son cargadas y además también se carga como nuevo set point un punto por encima del punto de seguridad, este punto tiene el valor de *L#-9187* y el punto de seguridad tiene el valor de *L#-10187*, al mismo tiempo *EXTRA1* acciona el bit de habilitación *"Z-parameter".enable_drive*. Como *EXTRA1* siempre está en 0 lógico entonces se usa su negación como accionamiento, además la variable temporal *EXTRA2* también está siempre desactivada, así que su negación se usa para llamar otra función la cual es denominada *"NUEVA META Z"* (ver figura 71).

Figura 71: Subida de emergencia

```
// EJE Z SUBIDA DE HEMERGENCIA:
// PALABRAS DOBLES:

      L     "BASE DE DATOS GENERAL".VELOCIDAD_EJE_Z_MIC_S
      T     "Z-parameter".profile_velocity

      L     "BASE DE DATOS GENERAL".ACELERACION_EJE_Z_MIC_SC
      T     "Z-parameter".profile_acceleration

      L     "BASE DE DATOS GENERAL".DESACELERACION_EJEZMICSC
      T     "Z-parameter".profile_deceleration

      L     L#300000
      T     "Z-parameter".jogging_velocity

      L     L#-9187
      T     "Z-parameter".target_position

// PALABRAS DOBLES.
// BITS:

      UN    #EXTRA1
      =     "Z-parameter".enable_drive

      UN    #EXTRA2
      CC    "NUEVA META Z"

// BITS.
// EJE Z SUBIDA DE HEMERGENCIA.
```

Fuente: Autor.

Esta función tiene la finalidad de accionar el bit de nuevo set point el cual inicia la corrida al punto indicado, además de esto mediante un temporizador verifica que el eje z llegue a la meta especificada. Para llevar a cabo esta verificación al entrar a la función *“NUEVA META Z”,* la negación de la variable temporal *VERIFICA* da inicio al conteo del temporizador *T15* el cual contara el tiempo especificado desde el panel táctil en la variable *TIEMPO_DE_VERIFICACION,* este tiempo es el plazo que se le da a los movimientos para que lleguen al punto definido, si el temporizador *T15* termina de contar y la función *“NUEVA META Z”* continua siendo llamada entonces se activan las variables *BITZ y VERIFICA,* al activarse la variable verifica el temporizador se desactiva , al desactivarse el temporizador se desactivan las variables *BITZ y VERIFICA ,* entonces esto causa que ambas variable se activen y desactiven casi de manera instantánea generando así un flanco negativo en las variables *BITZ y VERIFICA*, entonces el flanco negativo del variable *BITZ* genera un flanco positivo en el bit *"Z-parameter".new_set_point,* de esta forma si no se generó el movimiento a causa de no recibir el flanco positivo correctamente entonces el flanco se envía hasta que se haya logrado llegar a la meta, además de esta forma también se logra que el temporizador *T15* se reinicie cada vez que pase el tiempo de verificación y así se mantiene en continua verificación (ver figura 72).

Figura 72: Nueva meta z.

```
// INICIA CORRIDA A UNA NUEVA META:

      UN    #BITZ
      =     "Z-parameter".new_set_point

// INICIA CORRIDA A UNA NUEVA META.

// VERIFICA MOVIMIENTO:

      UN    #VERIFICA
      L     "BASE DE DATOS GENERAL".TIEMPO_DE_VERIFICACION
      SE    T     15
      U     T     15
      =     #BITZ
      =     #VERIFICA

// VERIFICA MOVIMIENTO.
```

Fuente: Autor.

La siguiente función a analizar es el *FB73,* denominado *RESETEO_TOTAL.* Esta función básicamente pone en 0 las memorias y variables del bloque *DB80* utilizadas en todo el programa, las que no se encuentran aquí es porque son variables de comunicación las cuales su estado depende solo de variables externas, de tal manera que no se ponen en cero para no generar conflicto con la comunicación (ver figura 73).

Figura 73: Reseteo total.

```
//RESETEO TOTAL:

      UN    #DIRECCION
      R     "A0.0"                                          A0.0
      R     "A0.2"                                          A0.2
      R     "A0.1"                                          A0.1
      R     "Z1"                                            Z1
      R     "Z2"                                            Z2
      R     "Z3"                                            Z3
      R     "Z4"                                            Z4
      R     "Z5"                                            Z5
      R     "Z6"                                            Z6
      R     "BASE DE DATOS GENERAL".SIMULACION_BIT_1        DB80.DBX242.0
      R     "BASE DE DATOS GENERAL".SIMULACION_BIT_2        DB80.DBX242.1
      R     "BASE DE DATOS GENERAL".SIMULACION_BIT_3        DB80.DBX242.2
      R     "BASE DE DATOS GENERAL".SIMULACION_BIT_4        DB80.DBX242.3
      R     "BASE DE DATOS GENERAL".SIMULACION_BIT_5        DB80.DBX242.4
      R     "BASE DE DATOS GENERAL".SIMULACION_BIT_6        DB80.DBX242.5
      R     "BASE DE DATOS GENERAL".BIT_PETICION_SIMULACION DB80.DBX280.1
      R     "BASE DE DATOS GENERAL".HMI_ERROR_DE_MOVIMIENTO DB80.DBX214.0
      R     "BASE DE DATOS GENERAL".HOMING_HECHO            DB80.DBX280.2

      L     0
      T     MD     0
      T     MD     4
      T     MD     8
      T     MD    12
      T     MW    16
      T     "BASE DE DATOS GENERAL".BANDA1                  DB80.DBW230
      T     "BASE DE DATOS GENERAL".BANDA2                  DB80.DBW232
      T     "BASE DE DATOS GENERAL".BANDA3                  DB80.DBW234
      T     "BASE DE DATOS GENERAL".BANDA4                  DB80.DBW236
      T     "BASE DE DATOS GENERAL".BANDA5                  DB80.DBW238
      T     "BASE DE DATOS GENERAL".BANDA6                  DB80.DBW240
      T     "BASE DE DATOS GENERAL".HMI_BANDA1              DB80.DBW216
      T     "BASE DE DATOS GENERAL".HMI_BANDA2              DB80.DBW218
      T     "BASE DE DATOS GENERAL".HMI_BANDA3              DB80.DBW220
      T     "BASE DE DATOS GENERAL".HMI_BANDA4              DB80.DBW222
      T     "BASE DE DATOS GENERAL".HMI_BANDA5              DB80.DBW224
      T     "BASE DE DATOS GENERAL".HMI_BANDA6              DB80.DBW226
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA1       DB80.DBW244
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA2       DB80.DBW246
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA3       DB80.DBW248
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA4       DB80.DBW250
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA5       DB80.DBW252
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA6       DB80.DBW254
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B1SIM       DB80.DBW256
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B2SIM       DB80.DBW258
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B3SIM       DB80.DBW260
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B4SIM       DB80.DBW262
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B5SIM       DB80.DBW264
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B6SIM       DB80.DBW266

//RESETEO TOTAL.
```

Fuente: Autor.

El siguiente bloque de función es el *FB81* el cual es definido como *SELECCIONAR_MODO*, esta función se compone de cuatro metas las cuales corresponden a los cuatro modos de operación automáticos (ver figura 74). El primer modo es el modo real con operario, el cual se define como real debido a que las señales que recibe son las señales enviadas por la CPU S7-200, es decir son las entradas a la CPU S7-200 que provienen de los sensores que indican si hay pieza o no en la banda.

Figura 74: Seleccionar modo.

```
// SELECCION POSICION:

// SALTOS:

      U     "M3.2"                                          M3.2
      SPB   RAL1

      U     "M3.3"                                          M3.3
      SPB   SIM1

      U     "M3.4"                                          M3.4
      SPB   RAL2

      U     "M3.5"                                          M3.5
      SPB   SIM2

      BEA

// SALTOS.

// CON OPERARIO:

RAL1: CALL  "CON_OPERARIO" , "MEMORIA_CON_OPERARIO"         FB78 / DB82
       BANDA1:="BASE DE DATOS GENERAL".BIT_1                DB80.DBX242.6
       BANDA2:="BASE DE DATOS GENERAL".BIT_2                DB80.DBX242.7
       BANDA3:="BASE DE DATOS GENERAL".BIT_3                DB80.DBX243.0
       BANDA4:="BASE DE DATOS GENERAL".BIT_4                DB80.DBX243.1
       BANDA5:="BASE DE DATOS GENERAL".BIT_5                DB80.DBX243.2
       BANDA6:="BASE DE DATOS GENERAL".BIT_6                DB80.DBX243.3

      BEA
```

Fuente: Autor.

Este primer modo se identifica con la meta *RAL1,* la cual es accionada mediante el salto condicional *SPB* que a su vez es accionado mediante la memoria M3.2.

Anteriormente se mencionó que mediante la pantalla se seleccionan los modos de operación, entonces como se puede ver esto se debe a que la memoria accionada desde el panel táctil activa la memoria correspondiente a la meta del modo, en este caso la memoria M3.6 que corresponde al modo real con operario es la que acciona la memoria M3.2 (ver figura 75). Los saltos se deben a que se requiere definir distintas variables a los bloques de los modos, entonces se necesita utilizar llamadas de bloque incondicionales porque estas son las que permiten definir parámetros locales a los parámetros formales, es decir, hay que utilizar el comando *CALL,* pero como este no depende del resultado lógico a diferencia de *CC,* entonces se controla mediante los saltos y las metas.

Figura 75: M3.6 activa M3.2.

```
U     #CONOPREAL
U     "M3.6"
=     "M3.2"
=     #SELECCION_MODO1
```

Fuente: Autor.

En la primera meta se llama al bloque *FB78* (*CON_OPERARIO*) junto con su bloque de datos de instancia que es el *DB82* y se le asignan los parámetros locales, que son los bits que reciben directamente las señales del intercambio de datos (ver figura 76), estos son *BIT_N.*

Figura 76: Comunicación 200-300.

```
// COMUNICACION 200-300:

      U     E     20.0
      =     "BASE DE DATOS GENERAL".BIT_1          DB80.DBX242.6
      =     "BASE DE DATOS GENERAL".HMI_SENSORB1   DB80.DBX228.0     -- SENSOR BANDA UNO

      U     E     20.1
      =     "BASE DE DATOS GENERAL".BIT_2          DB80.DBX242.7
      =     "BASE DE DATOS GENERAL".HMI_SENSORB2   DB80.DBX228.1     -- SENSOR BANDA DOS

      U     E     20.2
      =     "BASE DE DATOS GENERAL".BIT_3          DB80.DBX243.0
      =     "BASE DE DATOS GENERAL".HMI_SENSORB3   DB80.DBX228.2     -- SENSOR BANDA TRES

      U     E     20.3
      =     "BASE DE DATOS GENERAL".BIT_4          DB80.DBX243.1
      =     "BASE DE DATOS GENERAL".HMI_SENSORB4   DB80.DBX228.3     -- SENSOR BANDA CUATRO

      U     E     20.4
      =     "BASE DE DATOS GENERAL".BIT_5          DB80.DBX243.2
      =     "BASE DE DATOS GENERAL".HMI_SENSORB5   DB80.DBX228.4     -- SENSOR BANDA CINCO

      U     E     20.5
      =     "BASE DE DATOS GENERAL".BIT_6          DB80.DBX243.3
      =     "BASE DE DATOS GENERAL".HMI_SENSORB6   DB80.DBX228.5     -- SENSOR BANDA SEIS

// COMUNICACION 200-300.
```

Fuente: Autor.

El bloque de función *FB78* (*CON_OPERARIO)* funciona de la siguiente forma. Al ser llamado el bloque *FB78* desde el bloque *FB81* se acciona una secuencia la cual tiene la función de monitorear el estado de los bit de cada banda, así una vez detecta que hay pieza disponible en la banda, entonces ejecuta el movimiento de la posición actual a la posición de la banda y luego de la posición de la banda a la recepción correspondiente.

La secuencia inicia con la negación de la variable temporal *EXTRA0* la cual genera un flanco positivo por la llamada del bloque y este flanco es detectado por la variable temporal *EXTRA1,* luego que el flanco ha sido detectado se ancla la variable temporal *TEMPO1* y se resetea el bit *FIN_DE_MOVIMIENTO,* este bit es accionado cada vez que un movimiento es finalizado para hacer retornar de la función *MOVIMIENTO* a la secuencia, de tal forma que se resetea inicialmente como medida de seguridad, de esta forma no retorne sin realizar movimiento. La variable *TEMPO1* acciona el conteo de 100ms del temporizador *T16*, esto es con la finalidad de dar pasos temporizados en la secuencia y no pasos instantáneos de esta forma evitar que una señal quede estancada en el transcurso de la secuencia. Una vez finalizado el conteo de 100ms se resetea la variable *TEMPO1* y se anclan las variables *EXTRA2 y EXTRA3.* La variable *EXTRA2* habilita la comparación *HAY BIT* y la variable *EXTRA3* habilita la comparación *NO HAY BIT,* estas comparaciones son con respecto al bit de la banda 1 (ver figura 77).

Figura 77: Bloque de función con operario.

```
// 1
// INICIO CON FLANCO:

      UN    #EXTRA0
      FP    #EXTRA1
      S     #TEMP01
      R     "BASE DE DATOS GENERAL".FIN_DE_MOVIMIENT

// INICIO CON FLANCO.

// TIEMPO DE SECUENCIA #1:

      U     #TEMP01
      L     S5T#100MS
      SE    T     16
      U     T     16
      S     #EXTRA2
      S     #EXTRA3
      R     #TEMP01

// TIEMPO DE SECUENCIA #1.

// COMPARA HAY BIT1:

      U     #EXTRA2
      U     #BANDA1
      R     #EXTRA2
      R     #EXTRA3
      S     #EXTRA4

// COMPARA HAY BIT.

// COMPARA NO HAY BIT1:

      U     #EXTRA3
      UN    #BANDA1                 TEMP: BOOL
      R     #EXTRA2
      R     #EXTRA3
      S     #EXTRA5

// COMPARA NO HAY BIT.
```

Fuente: Autor.

Si la variable temporal *BANDA1* está en uno quiere decir que hay pieza en la banda 1, de tal forma que tiene que llevarse la pieza desde la banda hasta la recepción, entonces al estar en uno esta variable, se habilita la comparación hay bit y se deshabilita la comparación no hay bit, de tal forma que se resetean las variables *EXTRA2* y *EXTRA3* y a la vez se ancla la variable *EXTRA4.*

La variable *EXTRA4* activa el salto *OBJ1* el cual es donde se llama el bloque de función *FB77 (SELECCION POSICIONES)* (ver figura 78).

Figura 78: Saltos en FB78.

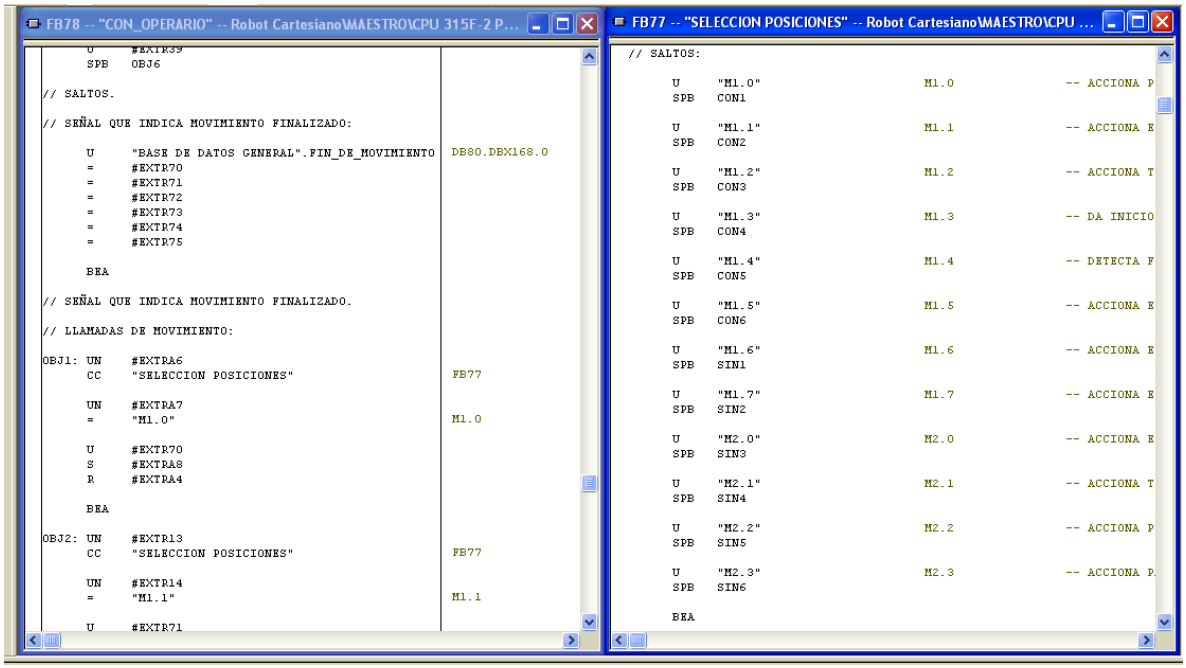
```
// SALTOS:

      U     #EXTRA4
      SPB   OBJ1

      U     #EXTR11
      SPB   OBJ2

      U     #EXTR18
      SPB   OBJ3

      U     #EXTR25
      SPB   OBJ4

      U     #EXTR32
      SPB   OBJ5

      U     #EXTR39
      SPB   OBJ6

// SALTOS.
```

Fuente: Autor.

Al saltar a la meta *OBJ1* la negación del bit *EXTRA6* llama la función *FB77*, al mismo tiempo la negación del bit *EXTRA6* activa el bit *M1.0,* de esta forma al llamar la función *FB77* y al activar el bit *M1.0* se ejecutaran los movimientos, esto se debe a que dentro de la función *FB77* se salta a la meta *CON1* mediante el bit *M1.0* en donde se llama a el bloque de función *FB74 (MOVIMIENTO)* con los datos correspondientes a las coordenadas de la banda *1* y de la recepción 1(ver figura 79 y 80).

Luego que se hayan ejecutados los movimientos y la pieza se encuentre en la recepción correspondiente, entonces se accionara el bit *FIN_DE_MOVIMIENTO,* el cual acciona las variables de la *EXTRA70 a la EXTRA75.*

Figura 79: llamada de FB77 dentro de meta OBJ1.



Fuente: Autor.

Figura 80: Meta CON1 dentro de FB77.

```
// CON OPERARIO:

CON1: CALL  "MOVIMIENTO" , "MEMORIA_MOVIMIENTO"
       CORDENADA1X     :="BASE DE DATOS GENERAL".BANDA1_X1
       CORDENADA1Y     :="BASE DE DATOS GENERAL".BANDA1_Y1
       CORDENADA1Z     :="BASE DE DATOS GENERAL".BANDA1_Z1
       CORDENADA2SUBIDA:="BASE DE DATOS GENERAL".CORDENADA_SUBIDA
       CORDENADA2X     :="BASE DE DATOS GENERAL".BANDA1_X2
       CORDENADA2Y     :="BASE DE DATOS GENERAL".BANDA1_Y2
       CORDENADA2Z     :="BASE DE DATOS GENERAL".BANDA1_Z2

      BEA
```

Fuente: Autor.

La variable *EXTRA70* da fin a la llamada de la meta *OBJ1* ya que resetea la variable *EXTRA4,* de esta forma una vez terminado el movimiento se deshabilitan las llamadas a las funciones *MOVIMIENTO* y *SELECCIÓN POCISIONES*, además se continua con la secuencia mediante el bit *EXTRA8* (ver figura 81).

Figura 81: Final de movimiento EXTRA70.

```
// SEÑAL QUE INDICA MOVIMIENTO FINALIZADO:

      U     "BASE DE DATOS GENERAL".FIN_DE_MOVIMIENTO
      =     #EXTR70
      =     #EXTR71
      =     #EXTR72
      =     #EXTR73
      =     #EXTR74
      =     #EXTR75

      BEA

// SEÑAL QUE INDICA MOVIMIENTO FINALIZADO.

// LLAMADAS DE MOVIMIENTO:

OBJ1: UN    #EXTRA6
      CC    "SELECCION POSICIONES"

      UN    #EXTRA7
      =     "M1.0"

      U     #EXTR70
      S     #EXTRA8
      R     #EXTRA4

      BEA
```

Fuente: Autor.

Ahora si en el momento de la verificación el bit *BANDA1* está en cero lógico entonces se habilita la comparación no hay bit, de tal manera que se resetean las variables *EXTRA2 y EXTRA3* y al mismo tiempo se ancla la variable *EXTRA5* (ver figura 82).

Figura 82: No hay bit.

```
// COMPARA NO HAY BIT1:

      U     #EXTRA3
      UN    #BANDA1
      R     #EXTRA2
      R     #EXTRA3
      S     #EXTRA5

// COMPARA NO HAY BIT.
```

Fuente: Autor.

Las variables *EXTRA5 y EXTRA8* son las que habilitan la continuación de la secuencia, debido a que se requiere que la secuencia continúe una vez terminado los movimientos o si no hay bit cuando se monitorea. De este punto en adelante la fusión *CON_OPERARIO* tiene la misma estructura.

El segundo modo es el modo simulación con operario, este modo funciona de la misma forma solo que los parámetros locales son los bits que se ingresan desde la pantalla (ver figura 83).

Figura 83: Modo simulación con operario.

```
SIM1: CALL  "CON_OPERARIO" , "MEMORIA_CON_OPERARIO"        FB78 / DB82
       BANDA1:="BASE DE DATOS GENERAL".SIMULACION_BIT_1    DB80.DBX242.0
       BANDA2:="BASE DE DATOS GENERAL".SIMULACION_BIT_2    DB80.DBX242.1
       BANDA3:="BASE DE DATOS GENERAL".SIMULACION_BIT_3    DB80.DBX242.2
       BANDA4:="BASE DE DATOS GENERAL".SIMULACION_BIT_4    DB80.DBX242.3
       BANDA5:="BASE DE DATOS GENERAL".SIMULACION_BIT_5    DB80.DBX242.4
       BANDA6:="BASE DE DATOS GENERAL".SIMULACION_BIT_6    DB80.DBX242.5

      BEA

// CON OPERARIO.
```

Fuente: Autor.

Antes de continuar con los modos sin operario analizaremos la función *MOVIMIENTO,* es decir el bloque *FB74.*

La función comienza con las comparaciones de las coordenadas del punto 1 que es la banda y el punto 2 que es la recepción o el punto común. Las comparaciones se llevan a cabo de la siguiente forma. Primero se da tolerancia positiva y negativa a los movimientos, es decir que a las coordenadas x, y del punto 1 y 2 se les define un intervalo debido a que el robot no llega al punto exacto que se indica, así que a las coordenadas x, y de los puntos 1 y 2 se les suma y se les resta el valor de L#500 y los resultados se cargan en memorias temporales. Estos resultados son los que se comparan con la posición actual de los ejes, es decir con las dobles palabras "*GV-CM-EJE-X".ActualPosition* y *"GV-CM-EJE-Y".ActualPosition.* Para

saber que el robot llego al punto 1 o 2 hay que preguntar si la posición está dentro del rango, así que se compara si las coordenadas x, y actuales son mayores que las coordenadas del punto menos la tolerancia y si son menores que las coordenadas del punto más la tolerancia, si es así entonces se activan dos bits para x y dos bits para y por punto. Estos cuatro bits activan un solo bit con la finalidad que este sea el que indica que el robot llego tanto a la coordenada x como a la y del punto específico. Para las coordenadas de los puntos 1 y 2 en z solo se compara el movimiento hacia abajo con el valor de la coordenada en z del punto más la tolerancia, es decir que se pregunta si la posición actual del eje z es menor que el valor de la coordenada en z del punto 1 o 2 más la tolerancia, si es así se activa un bit por punto. La coordenada de subida del eje z se compara de la misma forma que se hace en el OB1 para el error del eje z (ver figura 84 y 85).

Figura 84: Tolerancia y comparación, coordenada X punto1.

```
// MOVIMIENTO:

// COMPARACION PUNTO 1:

//P1X CARGA:

      L     #CORDENADA1X
      L     L#500
      -D
      T     #EXTRA0

      L     #CORDENADA1X
      L     L#500
      +D
      T     #EXTRA1

//P1X CARGA.

//P1X COMPARA:

      L     "GV-CM-EJE-X".ActualPosition    DB90.DBD32
      L     #EXTRA0
      >D
      =     #EXTRA10

      L     "GV-CM-EJE-X".ActualPosition    DB90.DBD32
      L     #EXTRA1
      <D
      =     #EXTRA11

//P1X COMPARA.
```

Fuente: Autor.

Figura 85: Tolerancia y comparación, coordenada Y, Z punto1.

```
//P1Y CARGA:

      L     #CORDENADA1Y
      L     L#500
      -D
      T     #EXTRA2

      L     #CORDENADA1Y
      L     L#500
      +D
      T     #EXTRA3

//P1Y CARGA.

//P1Y COMPARA:

      L     "GV-CM-EJE-Y".ActualPosition    DB91.DBD32
      L     #EXTRA2
      >D
      =     #EXTRA12

      L     "GV-CM-EJE-Y".ActualPosition    DB91.DBD32
      L     #EXTRA3
      <D
      =     #EXTRA13

//P1Y COMPARA.


//P1Z CARGA:

      L     #CORDENADA1Z
      L     L#500
      +D
      T     #EXTRA4

//P1Z CARGA.

//P1Z COMPARA:

      L     "Z-parameter".position_actual_value
      L     #EXTRA4
      <D
      =     #EXTRA14

//P1Z COMPARA.

      U     #EXTRA10
      U     #EXTRA11
      U     #EXTRA12
      U     #EXTRA13
      =     #EXTRA20
      =     "BASE DE DATOS GENERAL".HMI_MOVIMIENTOS1

// FIN DE COMPARACION PUNTO 1.
```

Fuente: Autor.

Luego de estas comparaciones viene la verificación. De manera similar a la que se realiza las verificación en el *OB1* se realiza en este bloque de función, la finalidad de la comparación en este bloque es evitar que el robot realice los movimientos con el eje z abajo ya que esto puede suceder si al cargar un punto se deja el eje z por debajo de la zona de seguridad (ver figura 86).

Figura 86: Seguridad en bloque MOVIMIENTO.

```
// VERIFICACION EJE Z ARRIBA:

      U     "Z-parameter".target_reached
      =     #EXTRA50
      =     #EXTRA51
      =     #EXTRA52
      =     #EXTRA53
      =     #EXTRA54

      L     "Z-parameter".position_actual_value
      L     L#-10187
      <=D
      =     #EXTRA22
      =     #EXTRA23
      =     #EXTRA70
      =     #EXTRA71

      UN    #EXTRA24
      FP    #EXTRA25
      S     #EXTRA26
      S     #EXTRA27

      U     #EXTRA26
      U     #EXTRA50
      UN    #EXTRA22
      L     S5T#500MS
      SE    T      7
      U     T      7
      R     #EXTRA26
      R     #EXTRA27
      S     #EXTRA28
      S     #EXTRA29

      U     #EXTRA27
      U     #EXTRA23
      CC    "SUBIDA_DE_HEMERGENCIA"

// FIN DE VERIFICACION EJE Z ARRIBA.
```

Fuente: Autor.

El bit *"Z-parameter".target_reached* es el que indica que el eje z se está moviendo, este bit esta en 1 lógico cuando el eje está quieto y en 0 lógico cuando está en movimiento, de tal manera que además de verificar si el eje está arriba aquí también se verifica que esté completamente quieto antes de iniciar el movimiento en los ejes x, y de tal forma que no se inicie el reseteo automático ya que si los ejes x, y están en movimiento a la vez que el eje z los motores se resetean para evitar daños catastróficos.

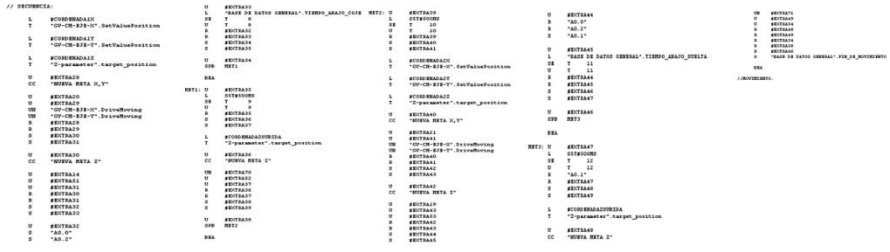
Una vez verificado que el eje z se encuentra en la zona de seguridad se inicia el conteo de 500 ms del temporizador *T7* el cual activa las variables temporales que dan inicio a la secuencia de movimiento (ver figura 87).

Durante esta secuencia se activan y desactivan las salidas A0.0, A0.1 y A0.2, se cargan las coordenadas con los parámetros formales *#CORDENADA #X, Y, Z* y se llaman los bloques de función *FB75 (NUEVA META X, Y) y FB76 (NUEVA META Z)*.

La salida A0.0 es la salida que acciona el vacío que genera la chupa, la A0.1 es la salida que acciona la expulsión de la chupa y la salida A0.2 acciona el *GRIPER*.

El bloque de función *FB75* Funciona de la misma forma que el bloque *FB76* la diferencia es que en este se verifica que el eje z no se mueva mientras los ejes x, y se mueven, si esto llegase a ocurrir entonces se resetearía todo y los motores quedarían inhabilitados (ver figura 88).

Figura 87: Secuencia.



Fuente: Autor.

Continuando con las metas del bloque *SELECCIONAR_MODO,* vemos que las metas *RAL2 y SIM2* son las metas que manejan el modo sin operario, en este hay seis parámetros formales más, los cuales indican el número de piezas en cada banda que la empacadora necesita.

Figura 88: NUEVA META X, Y.

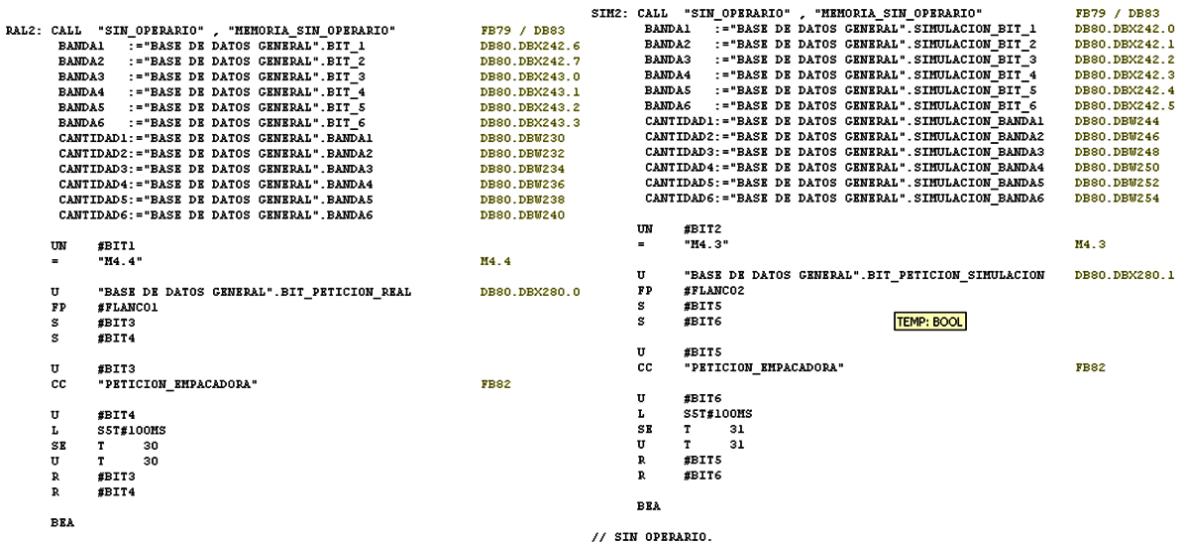
```
// INICIA CORRIDA A UNA NUEVA META:

      UN    #BITX
      =     "GV-CM-EJE-X".StartTask

      UN    #BITY
      =     "GV-CM-EJE-Y".StartTask

// INICIA CORRIDA A UNA NUEVA META.

// VERIFICA MOVIMIENTO:

      UN    #VERIFICA
      L     "BASE DE DATOS GENERAL".TIEMPO_DE_VERIFICACION
      SE    T     13
      U     T     13
      =     #BITX
      =     #BITY
      =     #VERIFICA

// VERIFICA MOVIMIENTO.

// RESETEA A MOVIMIENTO DOBLE:

      UN    "Z-parameter".target_reached
      L     S5T#100MS
      SE    T     14
      U     T     14
      =     #MOVIMIENTO_DOBLE1
      =     #MOVIMIENTO_DOBLE2
      S     "BASE DE DATOS GENERAL".HMI_ERROR_DE_MOVIMIENTO

      U     #MOVIMIENTO_DOBLE1
      CC    "RESETEO_MOTORES"

      U     #MOVIMIENTO_DOBLE2
      CC    "RESETEO_TOTAL"

// RESETEA A MOVIMIENTO DOBLE.
```

Fuente: Autor.

En estas metas se agrega una pequeña secuencia, la cual tiene la finalidad de llamar la función *PETICON_EMPACADORA,* en donde se cargan los datos

recibidos mediante la comunicación Ethernet a los parámetros formales agregados, de esta forma cada vez que la empacadora envié el bit de petición se cargaran los datos enviados a las variables *CANTIDAD#.* Las metas *RAL2 y SIM2* tienen la misma estructura solo que los bits de la meta *RAL2* son los bits reales y los bits de la meta *SIM2* son los bits de simulación, es decir los que se ingresan desde el panel táctil (ver figura 89 y 90).

Figura 89: Meta RAL2 y SIM2.

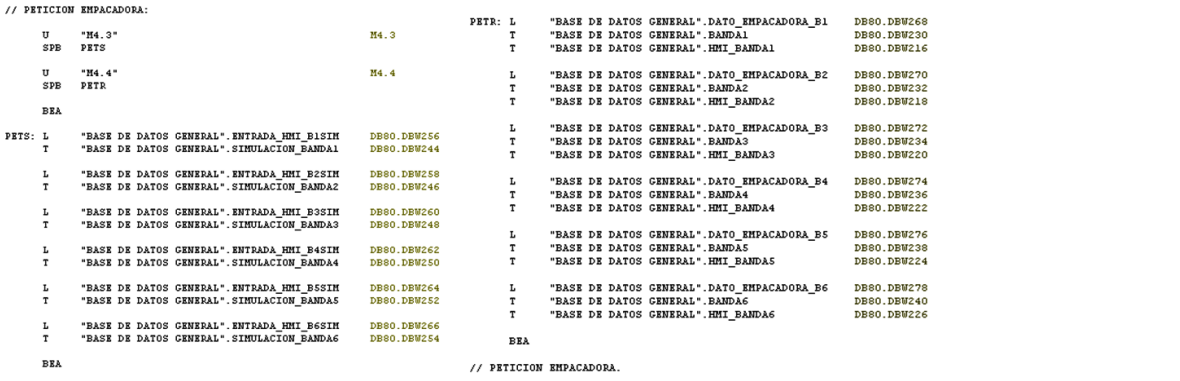


```
RAL2: CALL  "SIN_OPERARIO" , "MEMORIA_SIN_OPERARIO"          FB79 / DB83
       BANDA1   :="BASE DE DATOS GENERAL".BIT_1              DB80.DBX242.6
       BANDA2   :="BASE DE DATOS GENERAL".BIT_2              DB80.DBX242.7
       BANDA3   :="BASE DE DATOS GENERAL".BIT_3              DB80.DBX243.0
       BANDA4   :="BASE DE DATOS GENERAL".BIT_4              DB80.DBX243.1
       BANDA5   :="BASE DE DATOS GENERAL".BIT_5              DB80.DBX243.2
       BANDA6   :="BASE DE DATOS GENERAL".BIT_6              DB80.DBX243.3
       CANTIDAD1:="BASE DE DATOS GENERAL".BANDA1             DB80.DBW230
       CANTIDAD2:="BASE DE DATOS GENERAL".BANDA2             DB80.DBW232
       CANTIDAD3:="BASE DE DATOS GENERAL".BANDA3             DB80.DBW234
       CANTIDAD4:="BASE DE DATOS GENERAL".BANDA4             DB80.DBW236
       CANTIDAD5:="BASE DE DATOS GENERAL".BANDA5             DB80.DBW238
       CANTIDAD6:="BASE DE DATOS GENERAL".BANDA6             DB80.DBW240

      UN    #BIT1
      =     "M4.4"                                          M4.4

      U     "BASE DE DATOS GENERAL".BIT_PETICION_REAL       DB80.DBX280.0
      FP    #FLANCO1
      S     #BIT3
      S     #BIT4

      U     #BIT3
      CC    "PETICION_EMPACADORA"                           FB82

      U     #BIT4
      L     S5T#100MS
      SE    T     30
      U     T     30
      R     #BIT3
      R     #BIT4

      BEA
```

```
SIM2: CALL  "SIN_OPERARIO" , "MEMORIA_SIN_OPERARIO"          FB79 / DB83
       BANDA1   :="BASE DE DATOS GENERAL".SIMULACION_BIT_1   DB80.DBX242.0
       BANDA2   :="BASE DE DATOS GENERAL".SIMULACION_BIT_2   DB80.DBX242.1
       BANDA3   :="BASE DE DATOS GENERAL".SIMULACION_BIT_3   DB80.DBX242.2
       BANDA4   :="BASE DE DATOS GENERAL".SIMULACION_BIT_4   DB80.DBX242.3
       BANDA5   :="BASE DE DATOS GENERAL".SIMULACION_BIT_5   DB80.DBX242.4
       BANDA6   :="BASE DE DATOS GENERAL".SIMULACION_BIT_6   DB80.DBX242.5
       CANTIDAD1:="BASE DE DATOS GENERAL".SIMULACION_BANDA1  DB80.DBW244
       CANTIDAD2:="BASE DE DATOS GENERAL".SIMULACION_BANDA2  DB80.DBW246
       CANTIDAD3:="BASE DE DATOS GENERAL".SIMULACION_BANDA3  DB80.DBW248
       CANTIDAD4:="BASE DE DATOS GENERAL".SIMULACION_BANDA4  DB80.DBW250
       CANTIDAD5:="BASE DE DATOS GENERAL".SIMULACION_BANDA5  DB80.DBW252
       CANTIDAD6:="BASE DE DATOS GENERAL".SIMULACION_BANDA6  DB80.DBW254

      UN    #BIT2
      =     "M4.3"                                          M4.3

      U     "BASE DE DATOS GENERAL".BIT_PETICION_SIMULACION  DB80.DBX280.1
      FP    #FLANCO2
      S     #BIT5
      S     #BIT6                          TEMP: BOOL

      U     #BIT5
      CC    "PETICION_EMPACADORA"                           FB82

      U     #BIT6
      L     S5T#100MS
      SE    T     31
      U     T     31
      R     #BIT5
      R     #BIT6

      BEA

// SIN OPERARIO.
```

Fuente: Autor.

Figura 90: Petición empacadora.



```
// PETICION EMPACADORA:

      U     "M4.3"                                          M4.3
      SPB   PETS

      U     "M4.4"                                          M4.4
      SPB   PETR

      BEA

PETS: L     "BASE DE DATOS GENERAL".ENTRADA_HMI_B1SIM      DB80.DBW256
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA1      DB80.DBW244

      L     "BASE DE DATOS GENERAL".ENTRADA_HMI_B2SIM      DB80.DBW258
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA2      DB80.DBW246

      L     "BASE DE DATOS GENERAL".ENTRADA_HMI_B3SIM      DB80.DBW260
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA3      DB80.DBW248

      L     "BASE DE DATOS GENERAL".ENTRADA_HMI_B4SIM      DB80.DBW262
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA4      DB80.DBW250

      L     "BASE DE DATOS GENERAL".ENTRADA_HMI_B5SIM      DB80.DBW264
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA5      DB80.DBW252

      L     "BASE DE DATOS GENERAL".ENTRADA_HMI_B6SIM      DB80.DBW266
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA6      DB80.DBW254

      BEA
```

```
PETR: L     "BASE DE DATOS GENERAL".DATO_EMPACADORA_B1     DB80.DBW268
      T     "BASE DE DATOS GENERAL".BANDA1                 DB80.DBW230
      T     "BASE DE DATOS GENERAL".HMI_BANDA1             DB80.DBW216

      L     "BASE DE DATOS GENERAL".DATO_EMPACADORA_B2     DB80.DBW270
      T     "BASE DE DATOS GENERAL".BANDA2                 DB80.DBW232
      T     "BASE DE DATOS GENERAL".HMI_BANDA2             DB80.DBW218

      L     "BASE DE DATOS GENERAL".DATO_EMPACADORA_B3     DB80.DBW272
      T     "BASE DE DATOS GENERAL".BANDA3                 DB80.DBW234
      T     "BASE DE DATOS GENERAL".HMI_BANDA3             DB80.DBW220

      L     "BASE DE DATOS GENERAL".DATO_EMPACADORA_B4     DB80.DBW274
      T     "BASE DE DATOS GENERAL".BANDA4                 DB80.DBW236
      T     "BASE DE DATOS GENERAL".HMI_BANDA4             DB80.DBW222

      L     "BASE DE DATOS GENERAL".DATO_EMPACADORA_B5     DB80.DBW276
      T     "BASE DE DATOS GENERAL".BANDA5                 DB80.DBW238
      T     "BASE DE DATOS GENERAL".HMI_BANDA5             DB80.DBW224

      L     "BASE DE DATOS GENERAL".DATO_EMPACADORA_B6     DB80.DBW278
      T     "BASE DE DATOS GENERAL".BANDA6                 DB80.DBW240
      T     "BASE DE DATOS GENERAL".HMI_BANDA6             DB80.DBW226

      BEA

// PETICION EMPACADORA.
```

Fuente: Autor.

El siguiente bloque de función es el *FB79 o SIN_OPERARIO,* este tiene la misma estructura del bloque *CON_OPERARIO* así que solo se mostraran las diferencias.

Esta función empieza comparando todos los valores actuales de petición con el valor 0, de tal forma que si la empacadora pide una cierta cantidad de piezas de una banda, el valor actual de petición cargado en la variable CANTIDAD# sería diferente de 0 y el bit correspondiente a la comparación pasa a 1 lógico (ver figura 91).

Figura 91: Primera comparación.

```
// COMPARACIONES:

      L     #CANTIDAD1
      L     0
      <>I
      =     #COMPARACION1

      L     #CANTIDAD2
      L     0
      <>I
      =     #COMPARACION2

      L     #CANTIDAD3
      L     0
      <>I
      =     #COMPARACION3

      L     #CANTIDAD4
      L     0
      <>I
      =     #COMPARACION4

      L     #CANTIDAD5
      L     0
      <>I
      =     #COMPARACION5

      L     #CANTIDAD6
      L     0
      <>I
      =     #COMPARACION6

// COMPARACIONES.
```

Fuente: Autor.

La siguiente comparación consiste en preguntar si el contador de cada banda tiene valor actual diferente de 0, si es así se activa el bit *Z#_CON_0*. Esto se realiza con la finalidad de no tener en cuenta el valor 0 en la tercera comparación (ver figura 92).

En la última comparación se compara el valor actual de los contadores con el valor de la petición, de tal forma que cuando el valor actual del contador es igual que el número de piezas pedido se activa el bit *Z#_CON_B#* (ver figura 93).

El contador de cada banda incrementa cada vez que una pieza de la banda ha llegado al punto común, de esta forma el contador es el indicador de que se ha cumplido la petición.

Figura 92: Segunda comparación.

```
// COMPARACIONES CONTADORES CERO:

      L     "Z1"                        Z1
      L     0
      <>I
      =     #Z1_CON_0

      L     "Z2"                        Z2
      L     0
      <>I
      =     #Z2_CON_0

      L     "Z3"                        Z3
      L     0
      <>I
      =     #Z3_CON_0

      L     "Z4"                        Z4
      L     0
      <>I
      =     #Z4_CON_0

      L     "Z5"                        Z5
      L     0
      <>I
      =     #Z5_CON_0

      L     "Z6"                        Z6
      L     0
      <>I
      =     #Z6_CON_0

// COMPARACIONES CONTADORES CERO.
```

Fuente: Autor.

Luego de esto se pregunta mediante los bits *Z#_CON_0* y *Z#_CON_B#* si el valor del contador es diferente de 0 y si es igual al valor de la petición, si esto es así entonces quiere decir que ya se han llevado las piezas pedidas al punto común, de tal forma que se tiene que borrar el pedido y esto se hace mediante la función *BORRAR_DATOS_BANDAS* (ver figura 94).

Figura 93: Última comparación.

```
// COMPARACIONES CONTADORES BANDAS:

      L     "Z1"                        Z1
      L     #CANTIDAD1
      ==I
      =     #Z1_CON_B1

      L     "Z2"                        Z2
      L     #CANTIDAD2
      ==I
      =     #Z2_CON_B2

      L     "Z3"                        Z3
      L     #CANTIDAD3
      ==I
      =     #Z3_CON_B3

      L     "Z4"                        Z4
      L     #CANTIDAD4
      ==I
      =     #Z4_CON_B4

      L     "Z5"                        Z5
      L     #CANTIDAD5
      ==I
      =     #Z5_CON_B5

      L     "Z6"                        Z6
      L     #CANTIDAD6
      ==I
      =     #Z6_CON_B6

// COMPARACIONES CONTADORES BANDAS.
```

Fuente: Autor.

La función *BORRAR_DATOS_BANDAS* consiste simplemente en 6 saltos y 6 metas. Cada una de las metas corresponde a una banda así que en cada meta se borra el contador correspondiente a la banda y además se carga 0 a las palabras que guardan los datos del número de piezas pedidas, de esta forma se continua

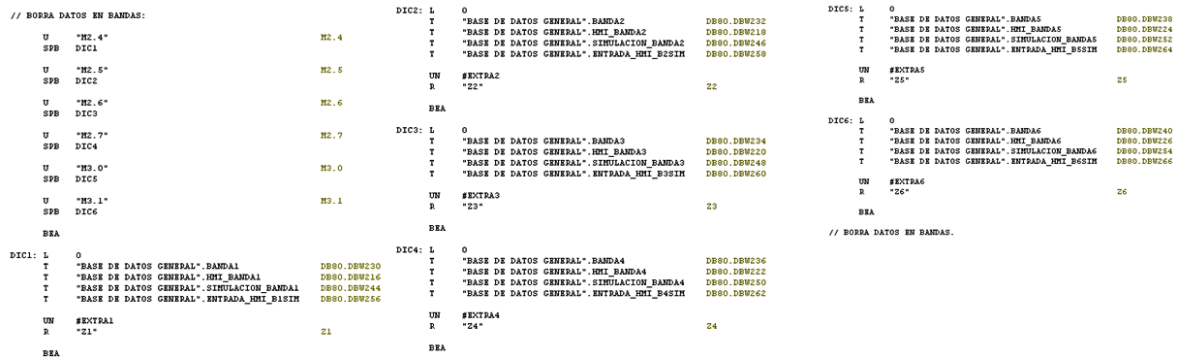
con la secuencia de verificación una vez desarrollada la petición, esto se debe a que el bit de la primera comparación (*COMPARACION#*) pasaría a ser 0 debido a que el valor actual de la petición (*CANTIDAD#*) pasaría a ser 0 (ver figura 95).

Figura 94: llama función BORRAR_DATOS_BANDAS.

```
// COMPARACIONES CONTADORES BANDAS.

// ACCIONAMIENTOS DE COMPARACIONES CONTADORE

      U     #Z1_CON_0
      U     #Z1_CON_B1
      =     "M2.4"                      M2.4
      =     #ADICIONAL3

      U     #Z2_CON_0
      U     #Z2_CON_B2
      =     "M2.5"                      M2.5
      =     #ADICIONAL6

      U     #Z3_CON_0
      U     #Z3_CON_B3
      =     "M2.6"                      M2.6
      =     #ADICIONAL9

      U     #Z4_CON_0
      U     #Z4_CON_B4
      =     "M2.7"                      M2.7
      =     #ADICIONAL12

      U     #Z5_CON_0
      U     #Z5_CON_B5
      =     "M3.0"                      M3.0
      =     #ADICIONAL15

      U     #Z6_CON_0
      U     #Z6_CON_B6
      =     "M3.1"                      M3.1
      =     #ADICIONAL18

// ACCIONAMIENTOS DE COMPARACIONES CONTADORE


// LLAMA FUNCION BORRAR:

      O     #ADICIONAL3
      O     #ADICIONAL6
      O     #ADICIONAL9
      O     #ADICIONAL12
      O     #ADICIONAL15
      O     #ADICIONAL18
      CC    "BORRA_DATOS_BANDAS"                    FB80

// LLAMA FUNCION BORRAR.
```

Fuente: Autor.

Figura 95: Función BORRAR_DATOS_BANDAS.

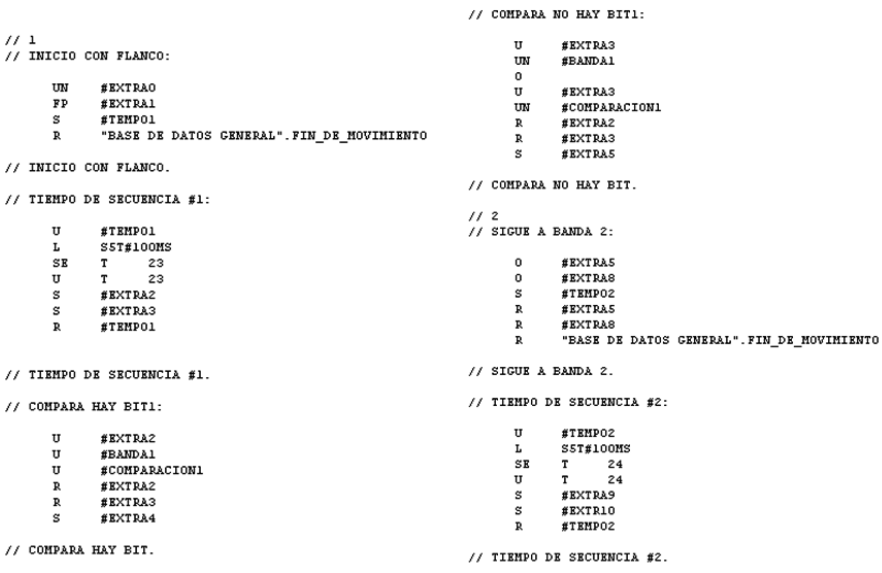


```
// BORRA DATOS EN BANDAS:

      U     "M2.4"                                  M2.4
      SPB   DIC1

      U     "M2.5"                                  M2.5
      SPB   DIC2

      U     "M2.6"                                  M2.6
      SPB   DIC3

      U     "M2.7"                                  M2.7
      SPB   DIC4

      U     "M3.0"                                  M3.0
      SPB   DIC5

      U     "M3.1"                                  M3.1
      SPB   DIC6

      BEA

DIC1: L     0
      T     "BASE DE DATOS GENERAL".BANDA1          DB80.DBW230
      T     "BASE DE DATOS GENERAL".HMI_BANDA1      DB80.DBW216
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA1 DB80.DBW244
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B1SIM DB80.DBW256

      UN    #EXTRA1
      R     "Z1"                                    Z1

      BEA

DIC2: L     0
      T     "BASE DE DATOS GENERAL".BANDA2          DB80.DBW232
      T     "BASE DE DATOS GENERAL".HMI_BANDA2      DB80.DBW218
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA2 DB80.DBW246
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B2SIM DB80.DBW258

      UN    #EXTRA2
      R     "Z2"                                    Z2

      BEA

DIC3: L     0
      T     "BASE DE DATOS GENERAL".BANDA3          DB80.DBW234
      T     "BASE DE DATOS GENERAL".HMI_BANDA3      DB80.DBW220
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA3 DB80.DBW248
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B3SIM DB80.DBW260

      UN    #EXTRA3
      R     "Z3"                                    Z3

      BEA

DIC4: L     0
      T     "BASE DE DATOS GENERAL".BANDA4          DB80.DBW236
      T     "BASE DE DATOS GENERAL".HMI_BANDA4      DB80.DBW222
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA4 DB80.DBW250
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B4SIM DB80.DBW262

      UN    #EXTRA4
      R     "Z4"                                    Z4

      BEA

DIC5: L     0
      T     "BASE DE DATOS GENERAL".BANDA5          DB80.DBW238
      T     "BASE DE DATOS GENERAL".HMI_BANDA5      DB80.DBW224
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA5 DB80.DBW252
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B5SIM DB80.DBW264

      UN    #EXTRA5
      R     "Z5"                                    Z5

      BEA

DIC6: L     0
      T     "BASE DE DATOS GENERAL".BANDA6          DB80.DBW240
      T     "BASE DE DATOS GENERAL".HMI_BANDA6      DB80.DBW226
      T     "BASE DE DATOS GENERAL".SIMULACION_BANDA6 DB80.DBW254
      T     "BASE DE DATOS GENERAL".ENTRADA_HMI_B6SIM DB80.DBW266

      UN    #EXTRA6
      R     "Z6"                                    Z6

      BEA

// BORRA DATOS EN BANDAS.
```

Fuente: Autor.

Luego de esto comienza la secuencia, pero esta vez no pregunta solamente si hay pieza en la banda si no que también pregunta si hay pedido, además en las metas se incrementan los contadores y activa una variable temporal la cual hace que se repita la verificación volviendo a empezar el tramo de código correspondiente, de esta forma si se piden N piezas de la banda N entonces se harán N movimientos (ver figura 96 y 97).

Figura 96: Inicio secuencia SIN_OPERARIO.



```
// 1
// INICIO CON FLANCO:

      UN    #EXTRA0
      FP    #EXTRA1
      S     #TEMP01
      R     "BASE DE DATOS GENERAL".FIN_DE_MOVIMIENTO

// INICIO CON FLANCO.

// TIEMPO DE SECUENCIA #1:

      U     #TEMP01
      L     S5T#100MS
      SE    T     23
      U     T     23
      S     #EXTRA2
      S     #EXTRA3
      R     #TEMP01


// TIEMPO DE SECUENCIA #1.

// COMPARA HAY BIT1:

      U     #EXTRA2
      U     #BANDA1
      U     #COMPARACION1
      R     #EXTRA2
      R     #EXTRA3
      S     #EXTRA4

// COMPARA HAY BIT.

// COMPARA NO HAY BIT1:

      U     #EXTRA3
      UN    #BANDA1
      O
      U     #EXTRA3
      UN    #COMPARACION1
      R     #EXTRA2
      R     #EXTRA3
      S     #EXTRA5

// COMPARA NO HAY BIT.

// 2
// SIGUE A BANDA 2:

      O     #EXTRA5
      O     #EXTRA8
      S     #TEMP02
      R     #EXTRA5
      R     #EXTRA8
      R     "BASE DE DATOS GENERAL".FIN_DE_MOVIMIENTO

// SIGUE A BANDA 2.

// TIEMPO DE SECUENCIA #2:

      U     #TEMP02
      L     S5T#100MS
      SE    T     24
      U     T     24
      S     #EXTRA9
      S     #EXTR10
      R     #TEMP02

// TIEMPO DE SECUENCIA #2.
```

Fuente: Autor.

Figura 97: Metas 1 y 2 bloque SIN_OPERARIO.

```
// LLAMADAS DE MOVIMIENTO:

OBJ1: UN    #EXTRA6
      CC    "SELECCION POSICIONES"      FB77

      UN    #EXTRA7
      =     "M1.6"                      M1.6

      U     #EXTR70
      =     #EXTRA0
      =     #ADICIONAL1
      =     #ADICIONAL2

      U     #ADICIONAL1
      R     #EXTRA4

      U     #ADICIONAL2
      ZV    "Z1"                        Z1

      BEA

OBJ2: UN    #EXTR13
      CC    "SELECCION POSICIONES"      FB77

      UN    #EXTR14
      =     "M1.7"                      M1.7

      U     #EXTR71
      =     #ADICIONAL4
      =     #ADICIONAL5

      U     #ADICIONAL4
      S     #EXTRA8
      R     #EXTR11

      U     #ADICIONAL5
      ZV    "Z2"                        Z2

      BEA
```

Fuente: Autor.

Por último que da el bloque *FB83 (CARGAR_ POCISIONES),* este básicamente se compone de 13 saltos y 13 metas, ya que son 13 puntos los que se pueden modificar, así que cada salto es accionado por una memoria, la cual es activada por la memoria que es forzada desde el panel táctil. En cada meta se cargan los valores actuales de las posiciones x, y, z a las memorias que guardan las coordenadas del punto correspondiente (ver figura 98).

Figura 98: Banda 1 y 2 bloque FB3.

```
U     M     32.0
SPB   M1B1

U     M     32.1
SPB   M2B1

U     M     32.2
SPB   M1B2

U     M     32.3
SPB   M2B2

U     M     32.4
SPB   M1B3

U     M     32.5
SPB   M2B3

U     M     32.6
SPB   M1B4

U     M     32.7
SPB   M2B4

U     M     33.1
SPB   M1B5

U     M     33.3
SPB   M2B5

U     M     33.5
SPB   M1B6

U     M     33.7
SPB   M2B6

U     M     34.1
SPB   PCOM

BEA
```

```
M1B1: L     "GV-CM-EJE-X".ActualPosition          DB90.DBD32
      T     "BASE DE DATOS GENERAL".BANDA1_X1      DB80.DBD8

      L     "GV-CM-EJE-Y".ActualPosition          DB91.DBD32
      T     "BASE DE DATOS GENERAL".BANDA1_Y1      DB80.DBD12

      L     "Z-parameter".position_actual_value   DB92.DBD28
      T     "BASE DE DATOS GENERAL".BANDA1_Z1      DB80.DBD16

      BEA

M2B1: L     "GV-CM-EJE-X".ActualPosition          DB90.DBD32
      T     "BASE DE DATOS GENERAL".BANDA1_X2      DB80.DBD20

      L     "GV-CM-EJE-Y".ActualPosition          DB91.DBD32
      T     "BASE DE DATOS GENERAL".BANDA1_Y2      DB80.DBD24

      L     "Z-parameter".position_actual_value   DB92.DBD28
      T     "BASE DE DATOS GENERAL".BANDA1_Z2      DB80.DBD28

      BEA

M1B2: L     "GV-CM-EJE-X".ActualPosition          DB90.DBD32
      T     "BASE DE DATOS GENERAL".BANDA2_X1      DB80.DBD32

      L     "GV-CM-EJE-Y".ActualPosition          DB91.DBD32
      T     "BASE DE DATOS GENERAL".BANDA2_Y1      DB80.DBD36

      L     "Z-parameter".position_actual_value   DB92.DBD28
      T     "BASE DE DATOS GENERAL".BANDA2_Z1      DB80.DBD40

      BEA

M2B2: L     "GV-CM-EJE-X".ActualPosition          DB90.DBD32
      T     "BASE DE DATOS GENERAL".BANDA2_X2      DB80.DBD44

      L     "GV-CM-EJE-Y".ActualPosition          DB91.DBD32
      T     "BASE DE DATOS GENERAL".BANDA2_Y2      DB80.DBD48

      L     "Z-parameter".position_actual_value   DB92.DBD28
      T     "BASE DE DATOS GENERAL".BANDA2_Z2      DB80.DBD52

      BEA
```

Fuente: Autor.

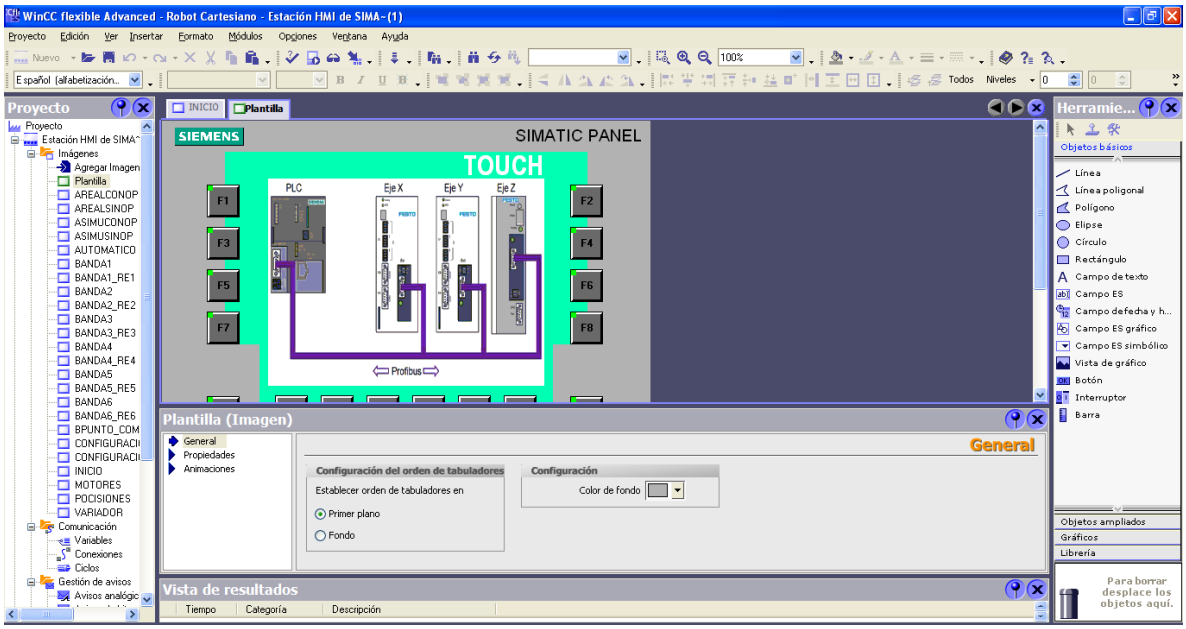
Con esto se termina la programación del robot cartesiano, ahora el siguiente paso es realizar la configuración del panel táctil[10].

[10] Los documentos consultados para realizar el código de trabajo son las referencias [15], [17], [18].

6. CÓNFIGURACION PANEL TACTIL

El primer paso es agregar la plantilla, es decir la imagen que aparecerá de fondo. Para agregarla nos dirigimos a mano izquierda en donde dice *Plantilla* y pegamos la imagen que queramos, en este caso el PLC comunicando con las tres tarjetas controladoras (ver figura 99).

Figura 99: Plantilla.

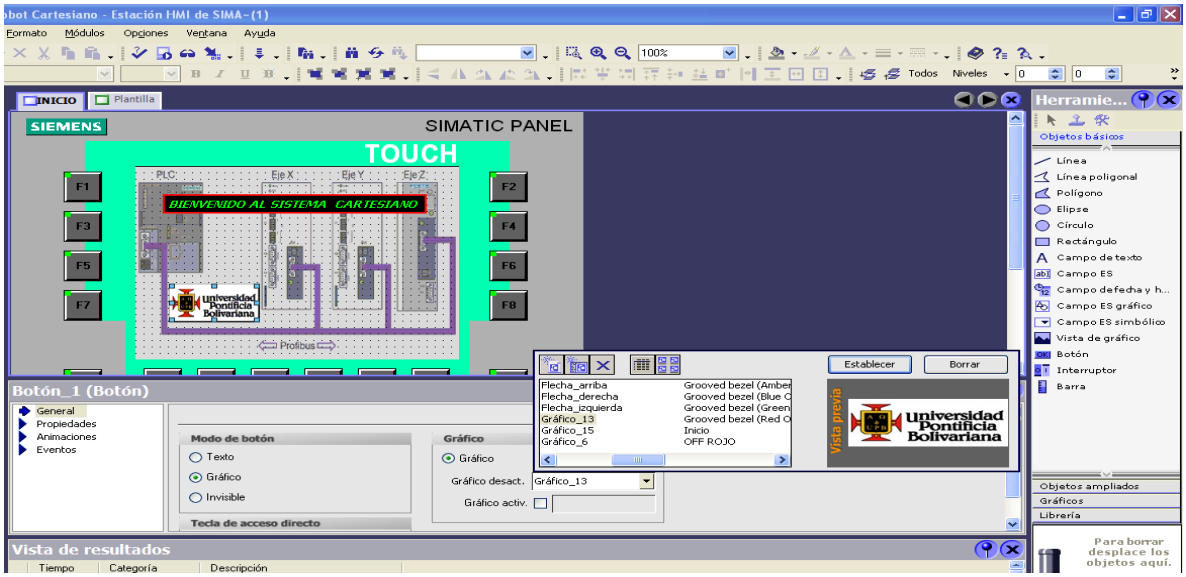


Fuente: Autor.

Luego de esto se crea la primera imagen dando click izquierdo en agregar imagen, a esta imagen se le da el nombre de *INICIO* y se le agrega un botón y un mensaje desde el menú de herramientas a mano derecha. Para agregar el boto se elige del menú de herramientas la opción *Botón* y para agregar el mensaje se elige la opción *Campo de texto.* Al botón se le agrega como imagen el símbolo de la universidad pontificia bolivariana y también se le agregan eventos como cambiar de imagen y activar un bit (ver figura 100 y 101).

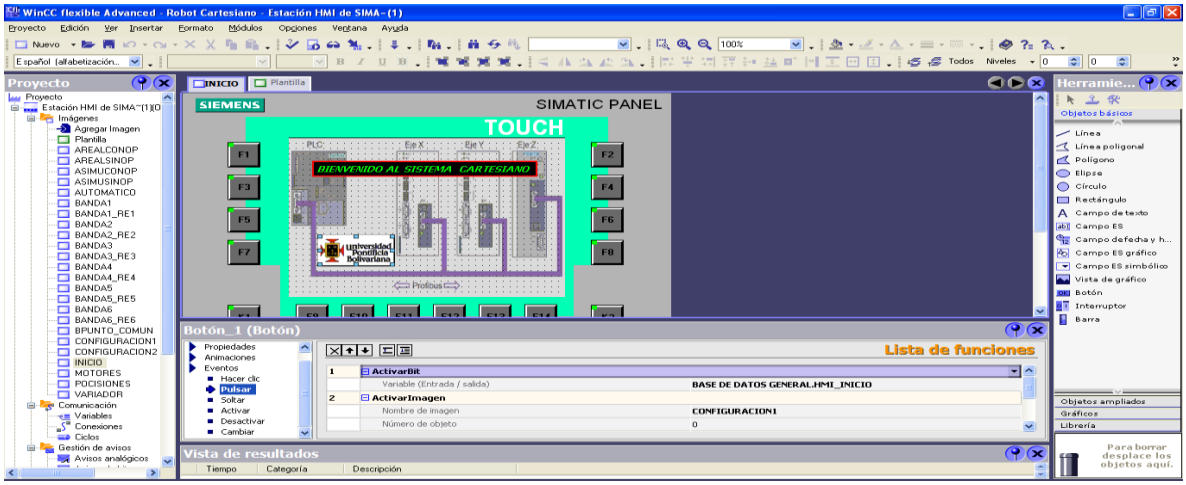
La siguiente imagen es *CONFIGURACION1*, aquí se agregan las velocidades de los motores x, y, los tiempos abajo y el tiempo de verificación, para esto se usa dos deslizadores obtenidos del menú objetos ampliados, tres campos de escritura, cuatro campos de texto y dos botones.

Figura 100: Imagen UPB.



Fuente: Autor.

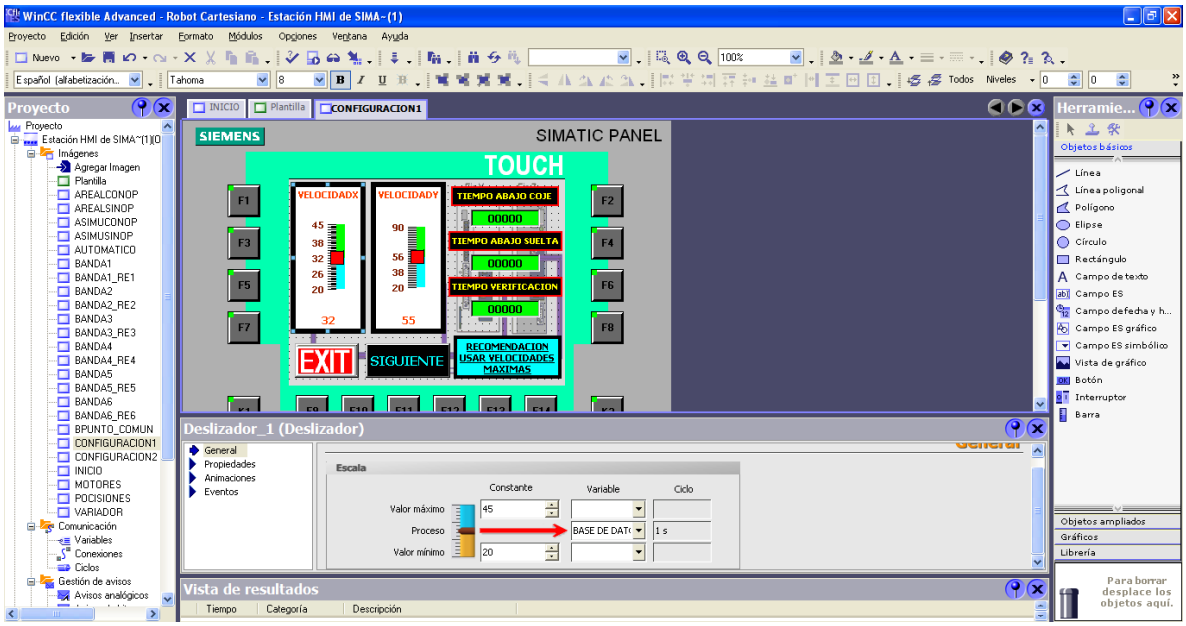
Figura 101: Eventos Botón_1.



Fuente: Autor.

En los deslizadores se asignan las variables *BASE DE DATOS GENERAL.VELOCIDAD_EJE_X_CM_S y BASE DE DATOS GENERAL.VELOCIDAD_EJE_Y_CM_S,* en los campos de escritura se asignan las variables *BASE DE DATOS GENERAL.TIEMPO_ABAJO_COJE, BASE DE DATOS GENERAL.TIEMPO_ABAJO_SUELTA y BASE DE DATOS GENERAL.TIEMPO_DE_VERIFICACION* y para los botones se les asigna como evento el cambio de imagen (ver figura 102).

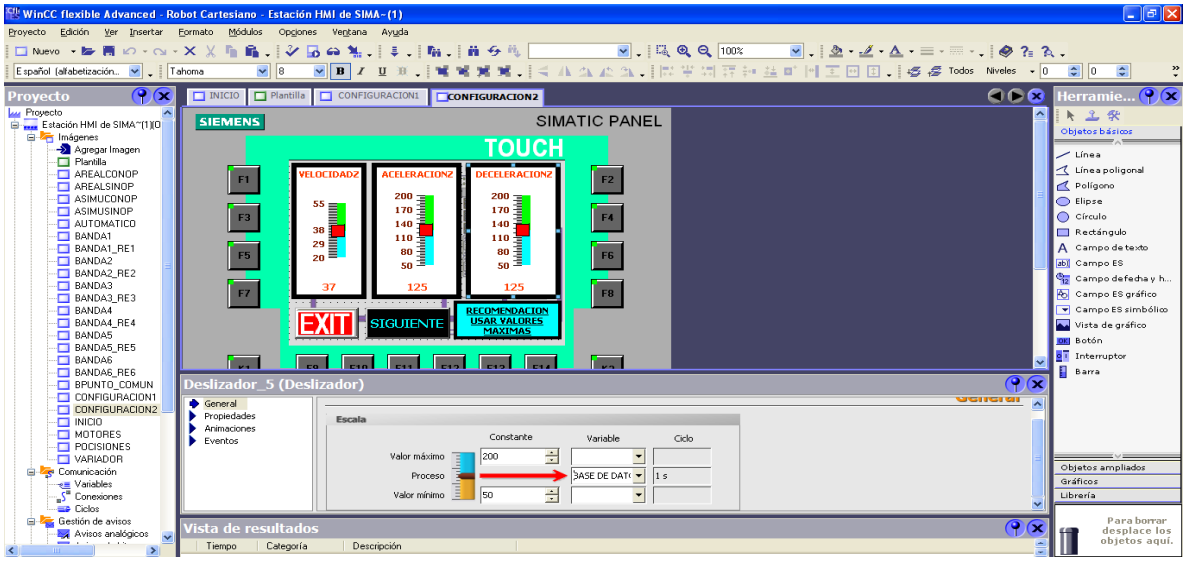
Figura 102: Ingreso inicial de datos de configuración.



Fuente: Autor.

Con el botón siguiente se pasa a la imagen *CONFIGURACION2,* en esta imagen se agregan los datos del eje z mediante tres deslizadores, a estos se les asigna las siguientes variables: *BASE DE DATOS GENERAL.VELOCIDAD_EJE_Z_CM_S, BASE DE DATOS GENERAL.ACELERACION_EJE_Z_CM_SC y BASE DE DATOS GENERAL.DESACELERACION_EJEZ_CMSC* (ver figura 103).

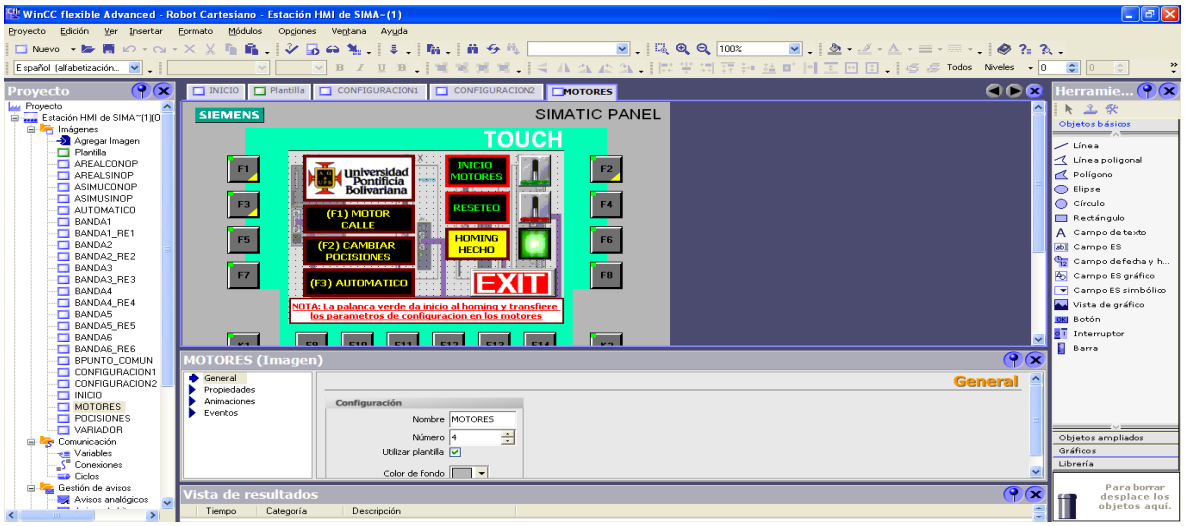
Figura 103: Ingreso parámetros eje z.



Fuente: Autor.

Con el botón siguiente se pasa a la imagen *MOTORES*, en esta imagen se da inicio al recorrido de referencia, se resetean los motores y las memorias principales y también se accede a las tres opciones (*MOTOR CALLE*, *CAMBIAR POSICIONES y AUTOMATICO*) (ver figura 104).

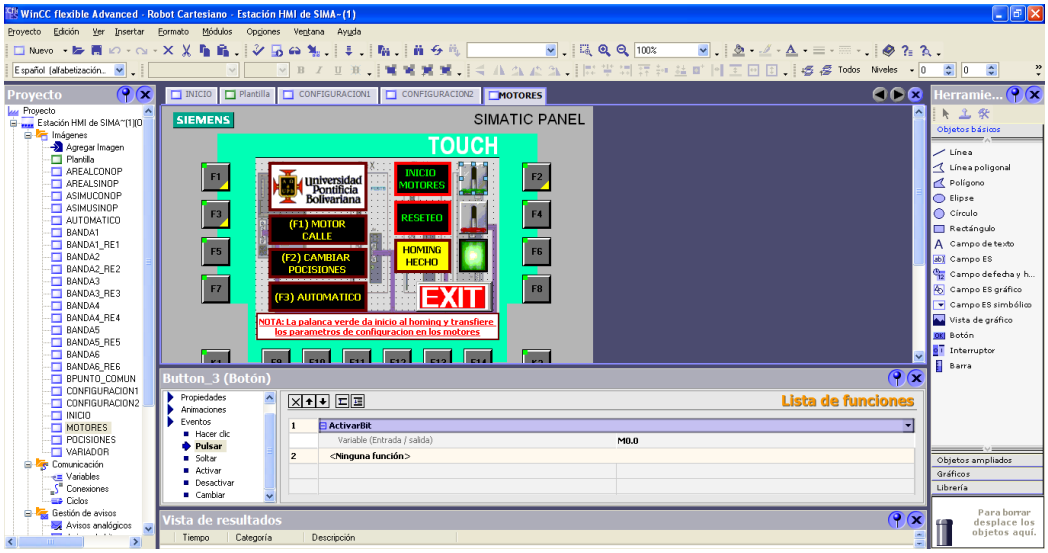
Figura 104: MOTORES.



Fuente: Autor.

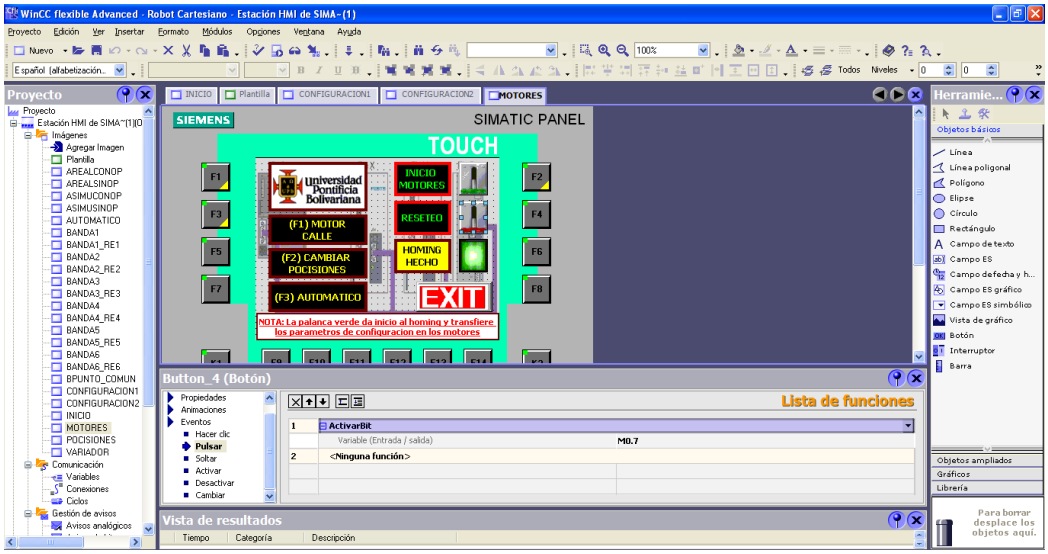
Para activar el recorrido de referencia, el botón tres tiene como evento activar la memoria M0.0 y para activar el reseteo, el botón cuatro tiene como evento activar la memoria M0.7 (ver figura 105 y 106).

Figura 105: M0.0 accionada por BOTÓN_3.



Fuente: Autor.

Figura 106: M0.7 accionada por BOTÓN_4.

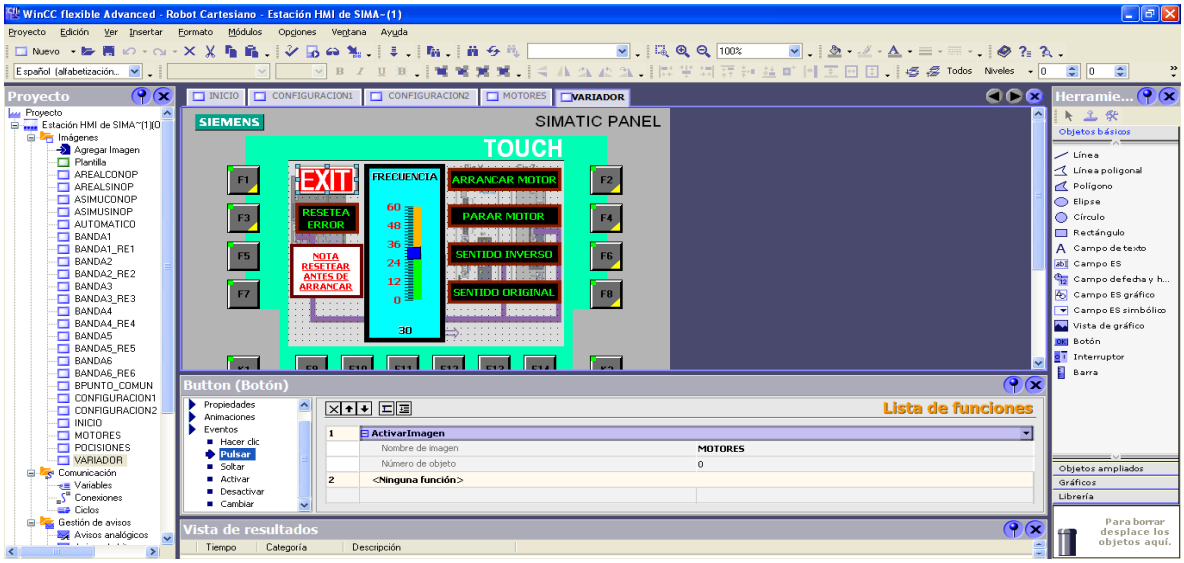


Fuente: Autor.

Los botones *F1, F2 y F3* del panel táctil activan las opciones que aparecen en la imagen motores. La primera opción es accionada por el botón *F1*, en esta imagen se manipula el motor que hace girar la banda de la calle de selección.

Para iniciar la manipulación primero se resetea error, debido a que generalmente el variador de frecuencia arranca con un error el cual impide el arranque del motor, entonces para resetear el error se oprime y suelta el boto *F3* el cual activa y desactiva el bit *DRIVE_FAULT_ACT,* luego de esto se asigna un valor de frecuencia mediante el deslizador, el cual carga su valor en la memoria *MD14* que dentro del programa es convertida en un dato real para ser cargado en el bloque de función que controla el variador, una vez la perilla se ha deslizado al valor que se desea, entonces se da inicio al motor mediante el botón *F2* el cual activa el bit *DRIVE_ON_OFF,* para detener el motor se oprime el botón *F4,* este desactiva el bit *DRIVE_ON_OFF.* Por ultimo si se quiere cambiar el sentido de giro se oprime el botón *F6* el cual acciona el bit *DRIVE_REVERSING* y para volverlo al sentido original entonces se pulsa el botón *F8* el cual resetea el bit *DRIVE_REVERSING* (ver figura 107).

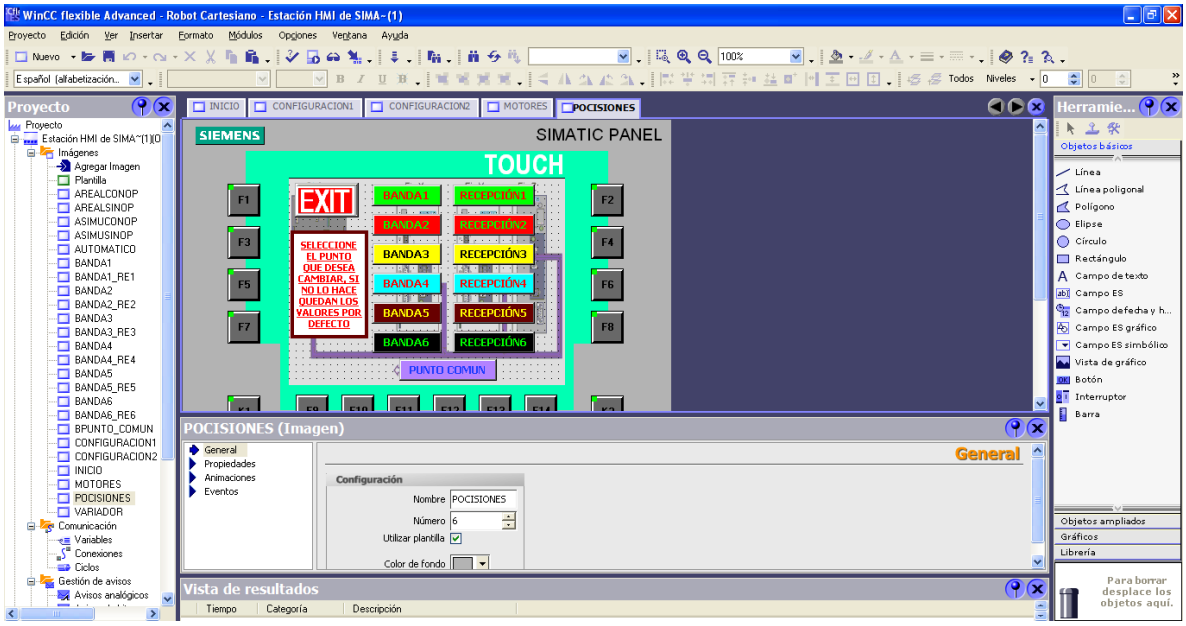
Figura 107: Variador de frecuencia MM440.



Fuente: Autor.

La siguiente opción es activada por F2 (*CAMBIA POSICIONES),* en esta opción se elige un punto, luego se mueve el robot hasta las nuevas coordenadas de ese punto y por último se cargan (ver figura 108).

Figura 108: Cambiar coordenadas punto.



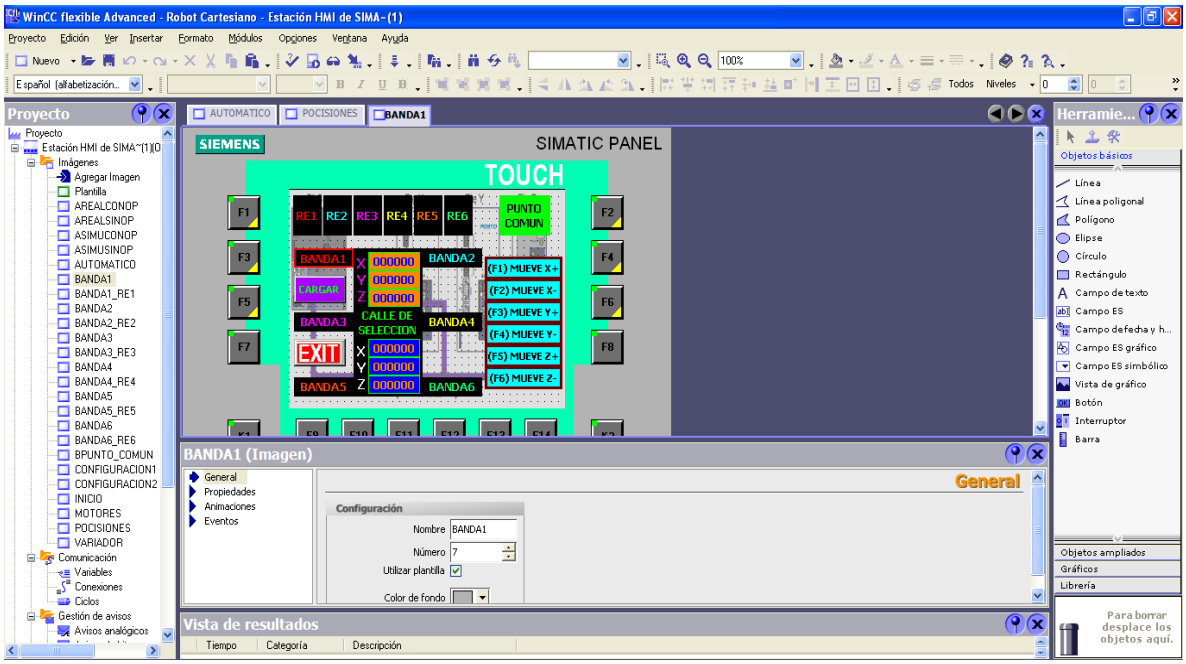
Fuente: Autor.

Al elegir un punto se activara una imagen en donde aparece la ubicación de cada punto en la calle de selección, el recuadro subrayado por el borde indica el punto actual.

Mediante los botones *F1, F2, F3, F4, F5, F6* se mueven los ejes, el botón *F1* activa y desactiva el bit *GV-CM-EJE-X.JogPos,* el botón *F2* activa y desactiva el bit *GV-CM-EJE-X.JogNeg,* el botón *F3* activa y desactiva el bit *GV-CM-EJE-Y.JogPos,* el botón *F4* activa y desactiva el bit *GV-CM-EJE-Y.JogNeg,* el botón *F3* activa y desactiva el bit *Z-parameter.jogging_plus,* el botón *F4* activa y desactiva el bit *Z-parameter.jogging_minus.* Mediante el botón *CARGAR* se transmiten los datos de las posiciones actuales a las memorias que guardan las coordenadas del punto respectivo (ver figura 109).

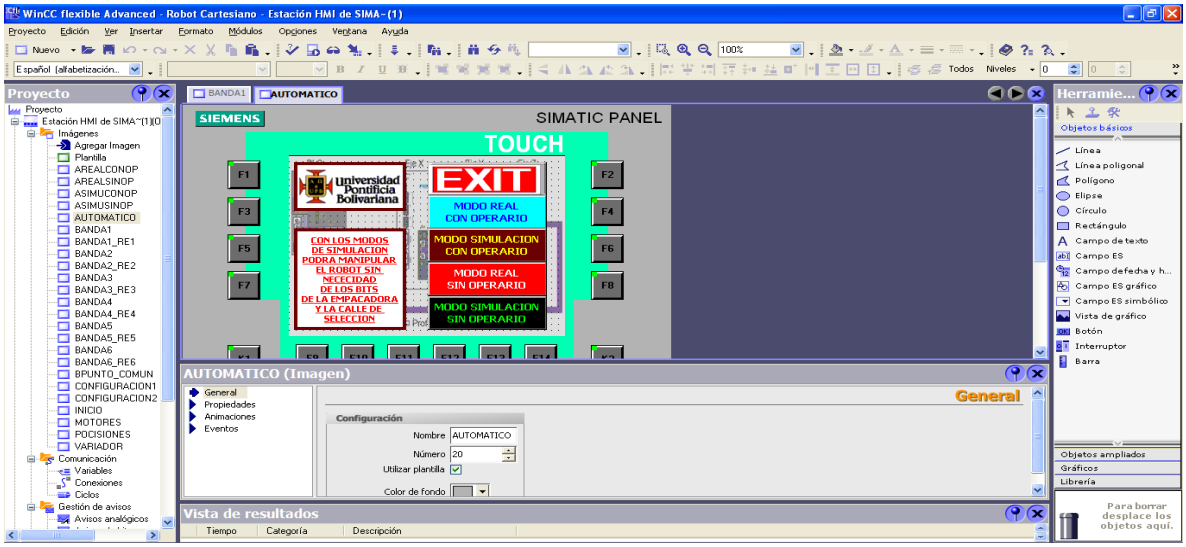
Por ultimo esta la opción tres que es el modo automático, en este modo podemos elegir alguno de los cuatro modos disponibles (ver figura 110). En el primer modo (modo con operario real) solamente se visualizan señales, como los bits de cada banda y los bits que indican que el movimiento 1 y el movimiento 2 fueron realizados (ver figura 111). En el segundo modo (modo con operario simulado) se activan bits que remplazan las señales de los sensores de cada banda y además se visualizan los bits de los movimientos 1 y 2 (ver figura 112).

Figura 109: Carga de posiciones BANDA1.



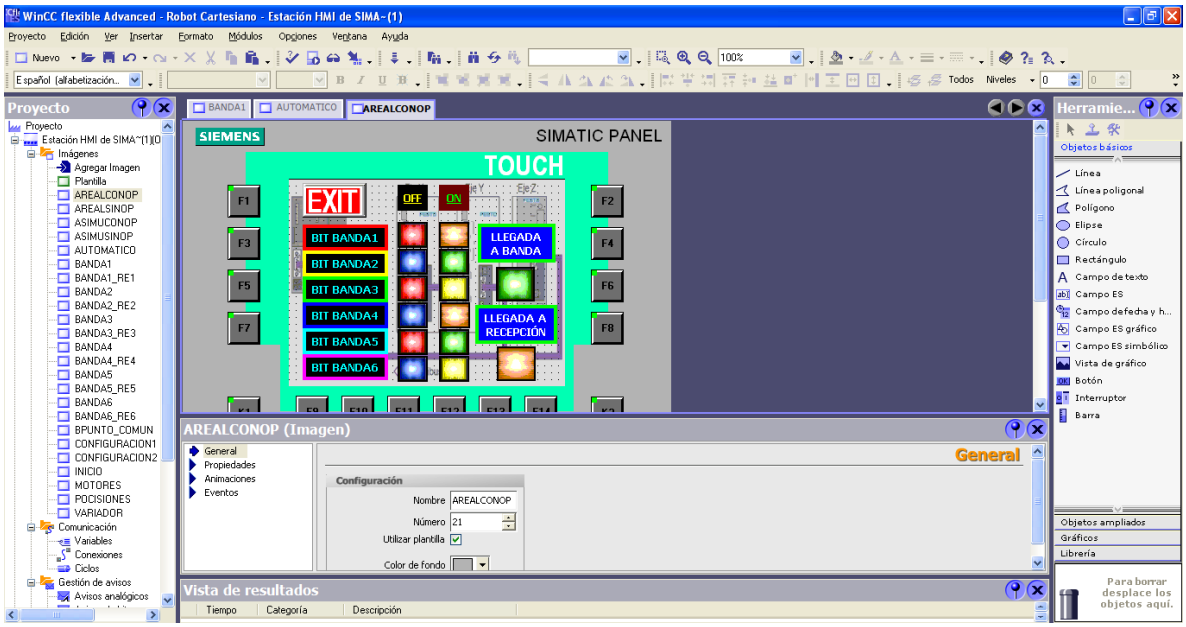
Fuente: Autor.

Figura 110: Modo automático.



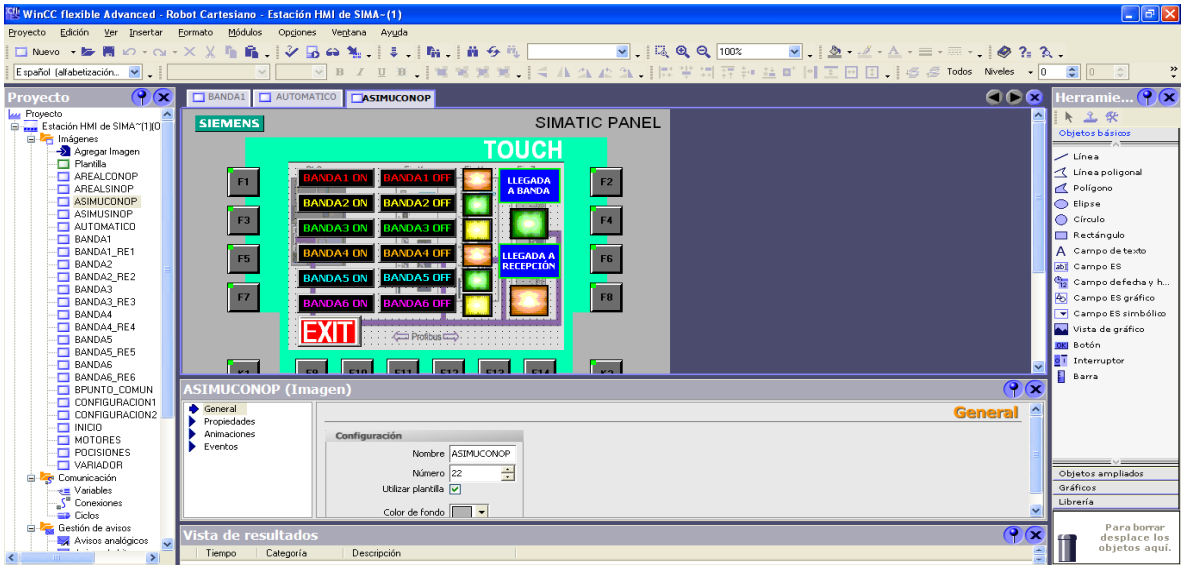
Fuente: Autor.

Figura 111: Modo con operario real.



Fuente: Autor.

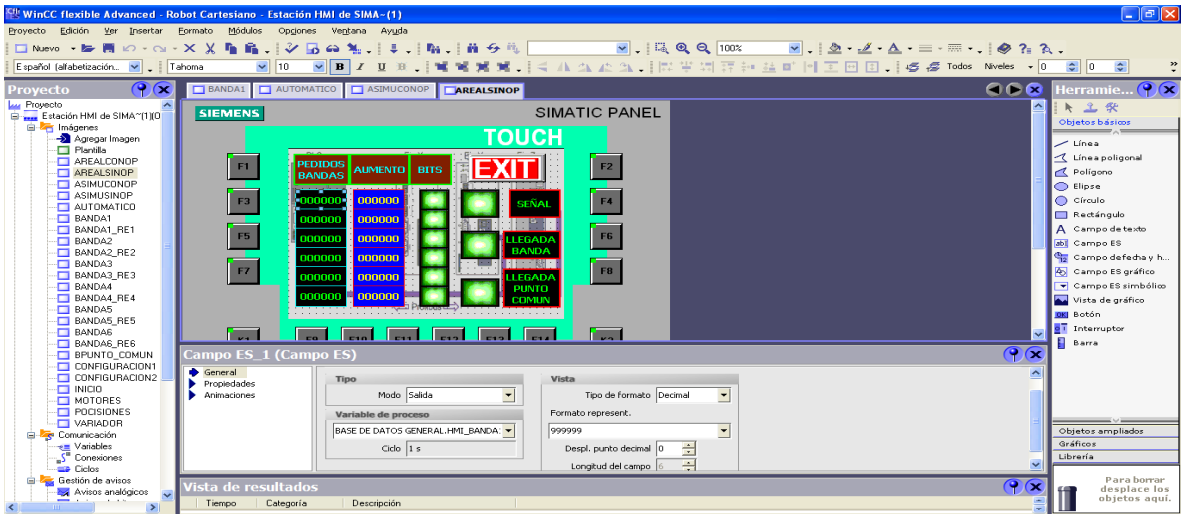
Figura 112: Modo con operario simulado.



Fuente: Autor.

En el tercer modo (modo sin operario real) se visualizan el número de piezas pedidas en cada banda, el número de piezas llevadas al punto común, las señales de los sensores de las bandas, la señal de petición de la empacadora, el bit de movimiento 1 y el bit de movimiento 2 (ver figura 113).

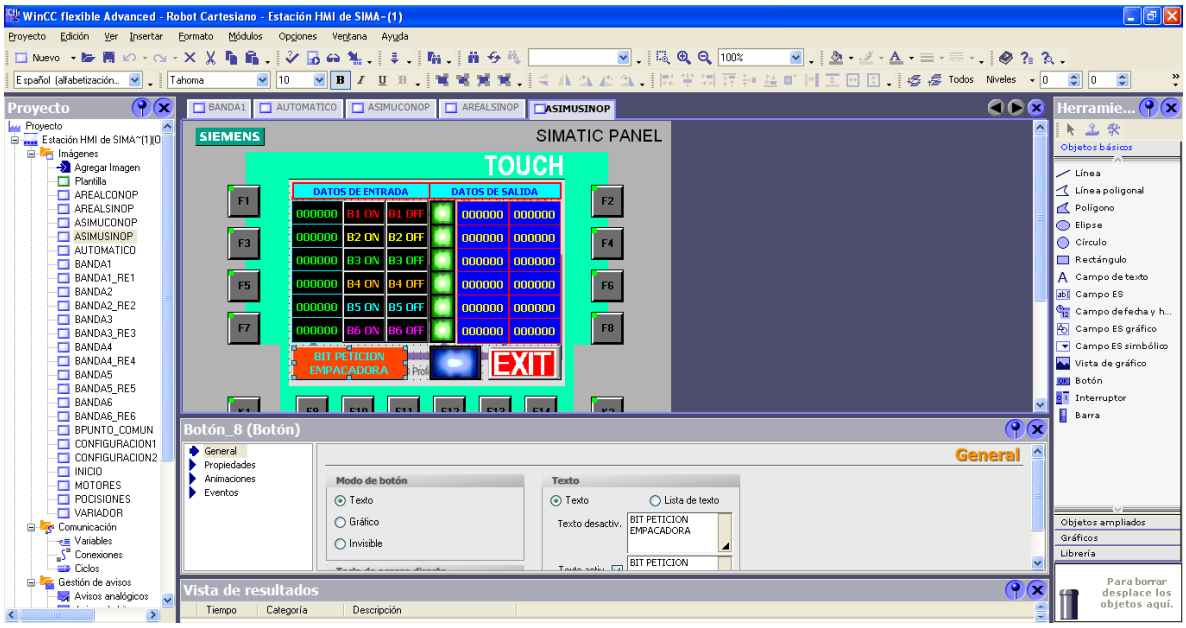
Figura 113: Modo sin operario real.



Fuente: Autor.

En el cuarto y último modo de operación (modo sin operario simulado) se ingresan el número de piezas que se requieren de cada banda, se accionan los bits que remplazan los sensores de las bandas y se activa el bit que remplaza el bit de petición real, también se visualizan los datos de petición, los contadores, el estado de los bits de simulación de cada banda y el estado del bit de petición para simulación (ver figura 114).

Figura 114: Modo sin operario simulado.



Fuente: Autor.

Con esto se terminan todos los aspectos necesarios para el funcionamiento del robot cartesiano tipo pórtico[11].

[11] Los documentos consultados para realizar la configuración de la HMI son las referencias [15] y [19].

RESULTADOS

Durante este proyecto se realizaron dos modelos de programas, como el primer código tuvo muchos inconvenientes, se decidió reprogramar para corregir todos los defectos posibles que se generaban.

El primer inconveniente fue que las secuencias se estancaban de un momento a otro, de tal forma que se realizó un seguimiento detallado para determinar la causa de este problema, el segundo inconveniente se presentó en paradas inesperadas de los movimientos y no siendo causa de un estancamiento en la secuencia se realizó una revisión en la función principal de movimiento, luego de esto se pudo observar que los movimientos del eje z solían ejecutarse con anticipación, de tal forma que se ponía en peligro la integridad de la máquina, también se pudo observar que si uno de los modos de funcionamiento era terminado durante la ejecución de un movimiento, el robot se descontrolaba, además de esto se observó que si algún movimiento ya sea el recorrido de referencia o un modo de operación se ejecutaba con el eje z abajo, se chocaba en algunas ocasiones, por último se pudo ver que al conmutar las paradas de emergencia físicas, el eje z no realizaba de nuevo el recorrido de referencia de tal forma que este se chocaba.

Desde el comienzo de este proyecto se ejecutaron todas las pruebas sobre una estructura de ICOPOR, de tal forma que los choques no generaron daños en la máquina.

El segundo código logro corregir estos inconvenientes mediante el uso de nuevas secuencias que remplazaron las defectuosas, de tal forma que la maquina quedo protegida de estos efectos peligrosos y con movimientos limpios y estables.

CONCLUSIONES Y RECOMENDACIONES

Para corregir los errores anteriores se llegó a diferentes conclusiones.

En primer lugar las secuencias deben tener tiempos intermedios para evitar el estancamiento, de tal forma que deben tener temporizadores para verificar correctamente las señales necesarias y además para evitar justamente este problema, el cual ocurre con mayor frecuencia en secuencias que manejan saltos y metas.

En segundo lugar se agregó tiempo de verificación para las señales que establecen un nuevo movimiento, de tal forma que si no se realiza el movimiento se genere la señal hasta que el robot cartesiano ejecute la orden. Esto ocurría debido a que el dato de la posición se definía al mismo tiempo que la señal, de tal forma que no existía un desfase de tiempo.

En tercer lugar se concluyó que los movimientos anticipados se debían a que el bit de movimiento del eje Y cambiaba de estado llegando a la parte final de la coordenada meta, así que se realizó un intervalo pequeño de comparación y no una comparación exacta por que los movimientos tienen un pequeño desfase. De esta forma el posicionamiento del robot no quedo solamente atado a los bits sino también al rango de comparación de posición en cada eje, asegurando así movimientos precisos.

En cuarto lugar para evitar el descontrol al terminar un modo de operación durante un movimiento, se concluyó que la mejor forma es usar variables temporales en cada subrutina, ya que estas se resetean automáticamente al salir del bloque que las contiene. Este error se presentaba debido a que el estado de las memorias se borraba al final de cada secuencia, de tal forma que si se terminaba el modo de operación durante el movimiento, no se reseteaban todas las memorias que contenía este, entonces cuando se activaba de nuevo la secuencia, esta entraba en conflicto.

En quinto lugar para evitar los choques en los movimientos a causa de que el eje z está extendido se concluyó que se debía crear una función de seguridad, para esto se definió un punto de seguridad, de tal forma que cada vez que se realiza la orden de posicionamiento se pregunta si el eje z está en la zona de seguridad, entonces si no esta se ejecuta la función de seguridad la cual lo hace subir hasta la zona y si esta entonces el robot ejecuta la orden De esta misma forma se eliminó el error producido por las paradas de emergencia.

Finalmente como recomendación para modificaciones futuras, tener siempre en cuenta los aspectos mencionados anteriormente y recalcando el uso de un código de seguridad para evitar que los ejes X y Y se muevan mientras el eje Z este extendido o se muevan al tiempo que el eje Z, de esta forma evitar la destrucción de la máquina.

BIBLIOGRAFÍA

[1] «Industrial, Robots Scara, robots Cartesianos y robots de seis ejes para los sectores de la alimentación, farmacéutico, logístico, empaquetado, pale tizado, automoción y electrónico». [En línea]. Disponible: http://www.tmrobotics.es/industries.html. [Consultado: 11-feb-2013].

[2] «Manufactura integrada por computador». [En línea]. Disponible: http: //www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r48942.PDF. [Consultado: 11-feb-2013].

[3] «ROBOT». [En línea]. Disponible: http://www.industriaynegocios.cl/Academicos/AlexanderBorger/Docts%20Docencia/Seminario%20de%20Aut/trabajos/Trabajos%202005/Robotica/5.-%20ROBOT.htm. [Consultado: 11-feb-2013].

[4] «Tipos de robots industriales». [En línea]. Disponible: http://es.scribd.com/doc/49615065/Tipos-de-robots-industriales. [Consultado: 11-feb-2013].

[5] «Robots industrials». [En línea]. Disponible: http://platea.pntic.mec.es/vgonzale/cyr_0708/archivos/_15/Tema_5.4.htm. [Consultado: 11-feb-2013].

[6] «Universidad Andrés Bello ». [En línea]. Disponible: http://noticias.unab.cl/facultades/ingenieria/laboratorio-de-automatizacion-y-robotica-estrena-moderno-equipamiento/. [Consultado: 12-feb-2013].

[7] «Universidad autónoma del caribe ». [En línea]. Disponible: http://www.uac.edu.co/facultad-ingenieria/recursos-e-infraestructura-facultad-ingenieria/927-laboratorio-de-robotica.html. [Consultado: 12-feb-2013].

[8] « FANUC ». [En línea]. Disponible: http://www.fanucrobotics.es/es/products/a_industrial-robots/lr%20mate%20series/lr%20mate%20200ic. [Consultado: 13-feb-2013].

[9] F.F.Rincon, J.E.Vegas, "Diseño, construcción y puesta en marcha de un prototipo robotizado cartesiano para prácticas de procesos industriales de almacenamiento y ensamble", tesis, Depto. Ing. Mecánica., UPB., Bucaramanga, Santander, 2009.

[10] *Festo Handling and Positioning Profile*. FESTO, República Federal de Alemania, 2007.

[11] *PROFIBUS para controladores de motor CMMS*. FESTO, Esslingen, 2007.

[12] *Function block S7 for stepper motor controller CMMS-ST*. FESTO, Esslingen, 2007.

[13] *STEP 7 function block to control an MM4 via Profibus-DP*. SIEMENS, 2005.

[14] F.F.Rincon, J.E.Vegas, «PROTOCOLO DE OPERACIÓN DEL SISTEMA ROBOTICO CARTESIANO DEL SALON I-304 DE LA UNIVERSIDAD PONTIFICIA BOLIVARIANA – UPB SECCIONAL BUCARAMANGA», 2010.

[15] Enrique Mandado Pérez, Jorge Marcos Acevedo, Celso Fernández Silva, y José I. Armesto Quiroga, *AUTÓMATAS PROGRAMABLES Y SISTEMAS DE AUTOMATIZACIÓN*, 2a EDICION. Alfaomega, 2009.

[16] *Smart Electromotor Controller*. FESTO, Federal Republic of Germany, 2003.

[17] «Tutorial de programación en Simatic S7». [En línea]. Disponible: http://lra.unileon.es/sites/lra.unileon.es/files/Documents/plc/Simatic/Manual_programacion_simatic_s7_300.pdf. [Consultado: 14-feb-2013].

[18] *Programar con STEP 7*, 03 ed. SIEMENS, 2006.

[19] «SIMATIC HMI WinCC flexible 200 8 Getting Started – Básico Getting Started». [En línea]. Disponible en: http://cache.automation.siemens.com/dnl/DQ/DQwNjk5AAAA_18660846_HB/WinCCflexible-GettingStarted-Basico_s.pdf. [Consultado: 16-feb-2013].