UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

PROYECTO DE AUTOMATIZACIÓN COORDINADA DE DOS PROTOTIPOS DE ROBOT MANIPULADOR CON DOS AUTÓMATAS SIEMENS S7 1200 MEDIANTE COMUNICACIÓN PROFINET, DESARROLLO DE APLICACIONES SCADA EN MATLAB Y PANTALLA HMI

AUTOR: David Enrique Juan San Valero

TUTOR: José Vicente Salcedo Romero de Ávila

Curso Académico: 2019-20

AGRADECIMIENTOS

Quiero aprovechar la ocasión para agradecer a todas las personas que me han acompañado durante toda esta etapa universitaria:

A mis compis del café, que me han hecho compañía y me han animado durante tantas jornadas de estudio en la biblioteca.

A mis amigos del cole, con los que he seguido manteniendo una estrecha amistad a pesar de que nuestros caminos se bifurcaran.

A mi tutor, que me ha ofrecido una gran ayuda durante la elaboración de este trabajo.

A mi familia, que me ha apoyado durante toda la vida, animándome y confiando en mi capacidad, incluso más que yo mismo.

A mi difunto abuelo, que me ha inspirado a seguir sus pasos, y que sé que estaría muy orgulloso de verme aquí.

Y por último, a mi novia, que me ha dado ánimos y fuerzas cada día de todos estos meses.

RESUMEN

En este proyecto se desarrollará la automatización coordinada de los robots manipuladores que realizan el transporte de piezas en un entorno productivo. La automatización se realizará con dos autómatas S7 1200 (uno para cada manipulador) que se comunicarán mediante Profinet. Para representar el funcionamiento de los motores que propulsan las articulaciones se utilizará un prototipo de motor de corriente continua que se controlará desde los citados S7 mediante señales analógicas.

Finalmente, se desarrollará una aplicación SCADA en Matlab para la supervisión del funcionamiento de la instalación automatizada que se comunicará mediante Modbus TCP/IP con los autómatas, a utilizar desde la sala de control de la planta industrial, y una aplicación de supervisión en una pantalla HMI disponible para los técnicos que trabajan directamente sobre el proceso automatizado.

Palabras clave: Automatización; SCADA; HMI; Profinet; Modbus TCP/IP; Prototipo de laboratorio; OPC

RESUM

En aquest projecte es desenvoluparà la automatització coordinada dels robots manipuladors que realitzen el transport de peces en un entorn productiu. La automatització es realitzarà amb dos autòmats S7 1200 (un per cada manipulador) que es comunicaran amb connexió Profinet. Per representar el funcionament dels motors que propulsen les articulacions, s'utilitzarà un prototip de motor de corrent continua que es controlarà des de els citats S7 per mitjà de senyals analògiques.

Finalment, es desenvoluparà una aplicació SCADA en Matlab per la supervisió del funcionament de la instal·lació automatitzada que es comunicarà amb Modbus TCP/IP amb els autòmats, per utilitzar des de la sala de control de la planta industrial, i una aplicació de supervisió en una pantalla HMI disponible per als tècnics que treballen directament sobre el procés automatitzat.

Paraules clau: Automatització; SCADA; HMI; Profinet; Modbus TCP/IP; Prototip de laboratori; OPC

ABSTRACT

This project will consist in the coordinated automation of two gripper robots which carry out parts transportation in a productive environment. The robots will be controlled with two Siemens S7 1200 PLCs (One for each prototype) with Profinet Communication. To simulate the real motors that would drive the joints of the robots, a continuous current motor prototype will be used, controlled by the PLCs through analogic signals.

Additionally, a SCADA application will be developed for the supervision and control of the automated process from the Control Room, using Modbus TCP/IP communication with the PLCs. This application will be complemented with a supervision HMI for the technicians, located next to the robots.

Key words: Automation, SCADA; HMI; Profinet; Modbus TCP/IP; lab prototype; OPC

Índice de contenidos

1. Introducción

1.1 Objeto

El objeto del presente Trabajo de Fin de Máster es el diseño, desarrollo y ejecución de los programas necesarios para monitorizar y controlar una línea productiva. Este proceso se ha llevado a cabo sobre una pequeña maqueta en los laboratorios de la universidad, pero tiene como objetivo el ser adaptable a, y útil para, un proceso real.

El trabajo comenzó por estudiar los componentes proporcionados por el Departamento de Ingeniería de Sistemas y Automática, diseñar un proceso apto para integrar dichos elementos y planear el funcionamiento de los distintos programas a desarrollar. Tras esto, se abordó la programación individual de cada uno de estos sistemas (PLC, SCADA y HMI) y más tarde su integración conjunta en el proyecto.

Por último, con afán de ampliar el alcance del proyecto y acercarlo a la industria actual, se ha recopilado una serie de aspectos a modificar e ideas de diseño para adaptar el resultado del proyecto a una línea productiva real.

1.2 Objetivos y justificación

- Experimentar las fases de diseño y desarrollo que forman parte de un proyecto de automatización.

- Asentar y ampliar el conocimiento adquirido durante grado y máster de Ingeniería industrial, tanto en la UPV como en universidades extranjeras.

- Aplicar la experiencia y el entendimiento sobre los sistemas de automatización y robótica ganados durante la realización de prácticas como programador de PLC en Ford.

- Crear un sistema de control completo y competente, aplicable a la industria actual.

1.3 Limitaciones encontradas

Es importante destacar que una gran parte de este proyecto se ha llevado a cabo durante el estado de emergencia en el país, lo cual ha impedido el acceso a los laboratorios y a los componentes físicos utilizados para el proyecto. Por este motivo, el funcionamiento de todos los programas ha tenido que ensayarse mediante simulaciones, realizadas a través de las herramientas de software disponibles. No obstante, se considera que el resultado es satisfactorio, y sólo requeriría pequeños ajustes y correcciones para crear un sistema completamente operativo.

1.4 Estructura del documento

El presente documento se ha organizado en distintos apartados, con la intención de ordenar y mostrar la información considerada necesaria para comprender todos los aspectos del proyecto.

En primer lugar se ofrece una visión general y completa del sistema diseñado y los componentes que forman parte de éste. Seguidamente, se ha dedicado un apartado a cada uno de los sistemas de control (PLC, HMI y SCADA). Tras esto se sugieren una serie de cambios para adaptar el proyecto a una línea real.

Para completar el TFM, se ha incluido un presupuesto del diseño inicial de la línea real, así como una serie de conclusiones obtenidas tras la realización del proyecto.

Por último se adjuntan dos anexos de programación, para añadir información adicional sobre el diseño y la ejecución de los sistemas desarrollados.

2. Descripción del sistema

2.1 Visión general

El objetivo de este proyecto es simular una línea de producción real. Para ello, se han utilizado los siguientes componentes:

- Dos prototipos de robot manipulador.
- Dos motores eléctricos para simular el funcionamiento de los motores de los robots.
- Dos PLC Siemens 1200 que se encargan de controlar los componentes anteriores.
- Un HMI ("Human Machine Interface", o Interfaz Humano-Máquina). Se trata de una pantalla táctil.
- Un ordenador.

Los robots, con sus respectivos motores, forman parte de una línea productiva en la que se llevan a cabo unos determinados procesos. Estos se encargan de transportar las piezas de una estación a otra, de manera secuencial. Todos los componentes de la línea están controlados por los PLC. Estos, además, se comunican con los maestros SCADA.

A pie de línea se encuentra el HMI. Desde aquí, los operarios tienen control básico e información sobre el sistema.

Por último, en la sala de control de la planta industrial, se encuentra el ordenador con el sistema de control (SCADA). Desde aquí, los ingenieros de la planta tienen control, información y diagnóstico avanzado sobre todas las líneas productivas de la fábrica.

En la siguiente figura se muestra un esquema de la planta industrial.

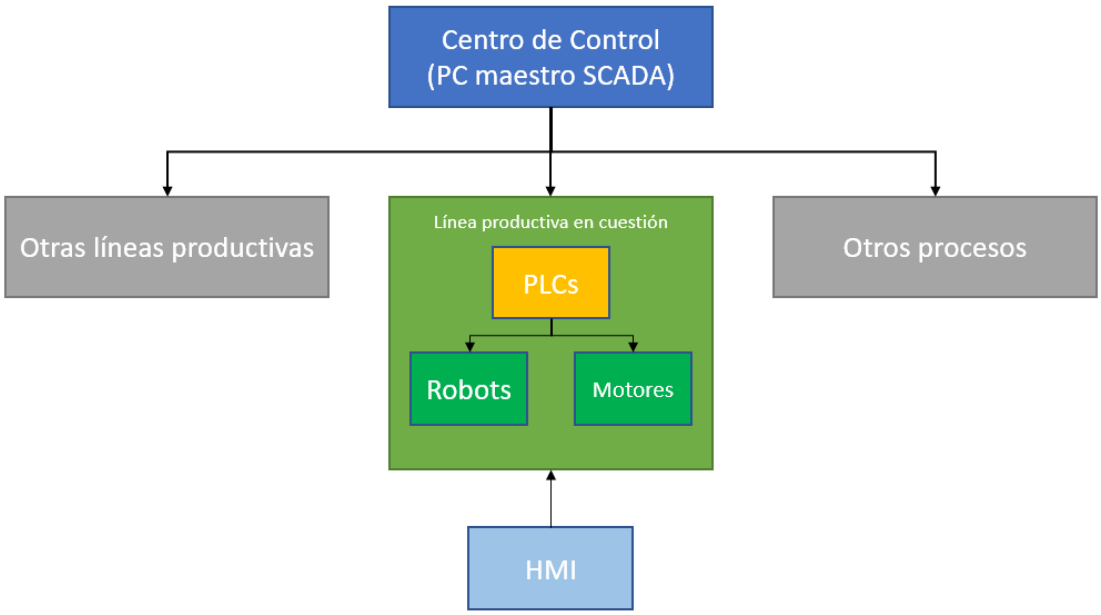


Ilustración 1 Esquema planta industrial. Fuente: Elaboración propia

2.2 Descripción del proceso

Para este proyecto se ha diseñado un proceso con una disposición y funcionalidad similares a lo que podría ser una línea productiva real. Consiste en dos robots manipuladores y diversas estaciones, en cada una de las cuales se llevará a cabo un proceso.

Aunque la industria actual está sujeta a una estandarización muy marcada, en muchas ocasiones se pueden encontrar piezas que tienen distintas variaciones. Por ejemplo, la carrocería de un coche varía según si es modelo híbrido, eléctrico o motor de combustión. Por este motivo es común encontrar líneas productivas con capacidad de incluir pequeñas modificaciones a una misma pieza para obtener distintas variantes o modelos. Estos cambios no son suficientemente grandes como para crear una nueva línea productiva, a no ser que se trate de una producción extremadamente especializada. Para añadir realismo al sistema se ha decidido incluir dos posibles modelos a producir. Según el modelo, las estaciones en las que los manipuladores dejan el producto cambian.

De esta manera, el Robot 1 (R1) coge la pieza desde la entrada de la línea, la deposita en la estación correspondiente según el modelo, espera a que acabe el proceso en dicha estación, la recoge y la lleva a la estación intermedia entre R1 y R2. Aquí se lleva a cabo otro proceso y finalmente R2 toma la pieza y la deposita en una última estación de salida de la línea, también dependiendo del modelo. Es importante destacar que el primer manipulador tiene un tiempo de ciclo mayor al segundo, por tanto no habrá un cuello de botella en la estación intermedia que bloquee a R1, a no ser que haya una avería en la segunda parte de la línea.

Además, cada robot tiene la opción de descartar las piezas defectuosas en una estación especial.

El siguiente diagrama de bloques resume el proceso:

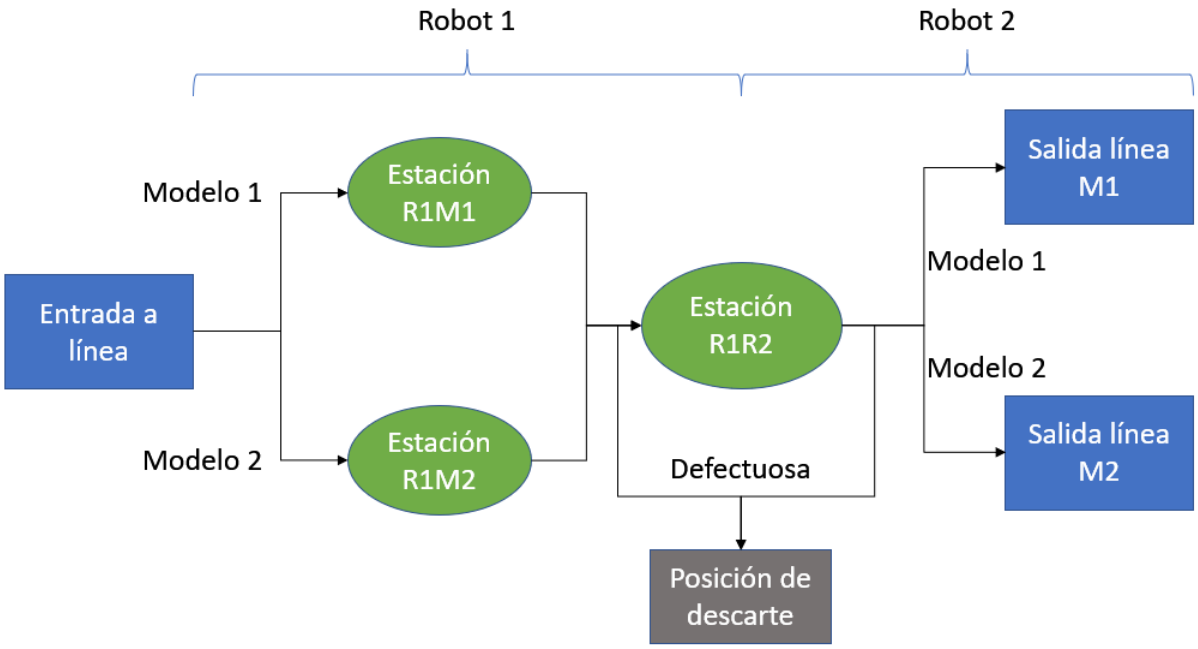


Ilustración 2 Esquema línea. Fuente: Elaboración propia

2.3 Robots manipuladores

Los prototipos usados para este proyecto son dos Robot manipulador de vacío de FischerTechnik. Estos disponen de movilidad en las tres dimensiones, permitiendo un amplio rango de operaciones. La siguiente imagen muestra una vista general del modelo utilizado:

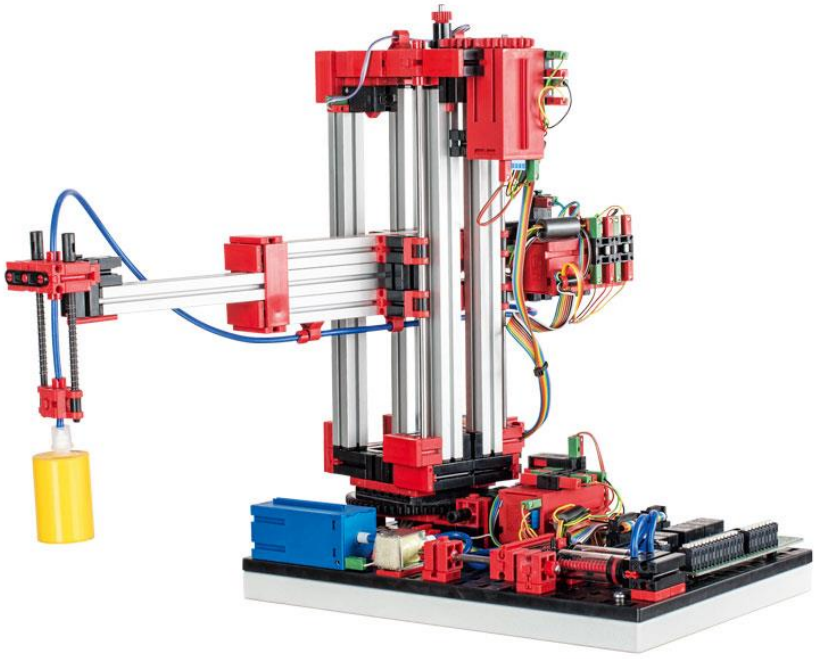


Ilustración 3 Robot manipulador de vacío. Fuente: Página web Fischertechnik

La maqueta cuenta con tres motores de 24V, uno para cada eje de movimiento (vertical, horizontal y giro sobre eje vertical). Estos motores pueden girar en directo o inverso, pero no se puede regular su velocidad.

Cada motor cuenta con un encoder, que permite calcular la posición actual, así como un sensor de final de carrera en uno de sus extremos. El robot cuenta, por tanto, con I1 para el movimiento vertical, I2 para el horizontal e I3 para el giro. Además, las lecturas de cada encoder han sido normalizadas de 0 a 100, para que sean más descriptivas.

En la siguiente figura se muestra una representación gráfica de cada uno de estos movimientos:

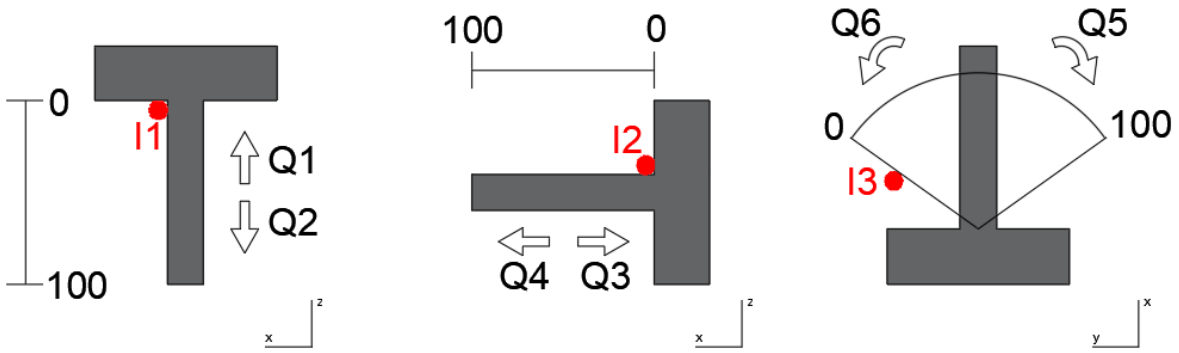


Ilustración 4 Movimientos robot manipulador. Fuente: Elaboración propia

Cada uno de los movimientos es controlado por el PLC a través de las señales Q1 a Q6 (salidas del PLC). Los pulsos del encoder se envían al PLC mediante las entradas B1, B3 y B5.

Adicionalmente, el robot cuenta con un compresor de aire y una ventosa, para simular el manipulador. Estos componentes están controlados por las señales Q7 y Q8, respectivamente.

En las siguientes tablas se muestra un resumen de las entradas y salidas, así como las direcciones de memoria. [1]

NOTA: el término entrada o salida hace referencia al PLC. Por tanto, Q1 a Q9 serán SALIDAS y el resto serán ENTRADAS.

Entrada	Descripción	Dirección de memoria
I1	Final de carrera referencia vertical	%I0.4
I2	Final de carrera referencia horizontal	%I0.5
I3	Final de carrera referencia giro	%I0.6
B1	Pulsos encoder movimiento vertical	%I0.0
B3	Pulsos encoder movimiento horizontal	%I0.1
B5	Pulsos encoder movimiento giro	%I0.2

Tabla 1 Entradas robot. Fuente: Elaboración propia

Salida	Descripción	Dirección de memoria
Q1	Motor movimiento vertical arriba	%Q0.0
Q2	Motor movimiento vertical abajo	%Q0.1
Q3	Motor movimiento retroceso horizontal	%Q0.2
Q4	Motor movimiento avance horizontal	%Q0.3
Q5	Motor movimiento giro horario	%Q0.4
Q6	Motor movimiento giro antihorario	%Q0.5
Q7	Habilitar compresor	%Q0.6
Q8	Succión ventosa	%Q0.7
Q9	Habilitar señales de entrada del proceso	%Q1.0

Tabla 2 Salidas robot. Fuente: Elaboración propia

2.4 Motores eléctricos

Para poder representar el control de los motores que propulsan las articulaciones de los brazos robóticos se ha utilizado un motor externo de corriente continua, que se controla desde el PLC mediante señales analógicas. El control se realiza con un PID y el parámetro a controlar es la velocidad de giro, que se ajusta modificando el voltaje de entrada al motor.

Se ha utilizado el motor Artitecnic v2.0, que tiene las siguientes características.

- Conectores:
  - Masa
  - Entrada de tensión [-10, 10] V
  - Salida de tensión con la velocidad del motor [-10, 10] V
  - Salida de tensión con la posición del motor [-10, 10] V
  - Salida de tensión con la posición del flex-link.

  En este caso sólo se ha utilizado la entrada de tensión del motor y la salida con la velocidad.
- Interruptores:
  - R Carga, que sirve para modificar la ganancia de la respuesta. Tiene dos posiciones, ganancia alta y ganancia baja.
  - C Inercia, utilizado para seleccionar la dinámica de la respuesta. Tiene dos posiciones, dinámica rápida y dinámica lenta.

En este caso se ha utilizado las posiciones de ganancia alta y dinámica lenta. Con esta configuración, el motor tiene una zona muerta entre [-0.6, 0.6] V y un comportamiento lineal para un rango de entradas de [-6, 6] V. Por este motivo, la velocidad correspondiente a 6V es la máxima velocidad utilizada durante el control.

Para las simulaciones y el ajuste de los parámetros del PID se ha utilizado el modelo de motor [2]:

$$Gp(s) = \frac{0.8374}{1 + 0.9987s}$$

Ecuación 1 función de transferencia motor eléctrico. Fuente: Documentación DISA

Este modelo se puede obtener con facilidad realizando un ensayo de escalón en el laboratorio. Como se puede observar se trata de un sistema de primer orden con un error fijo en estado estacionario.

El comportamiento en bucle abierto es el siguiente (derecha):

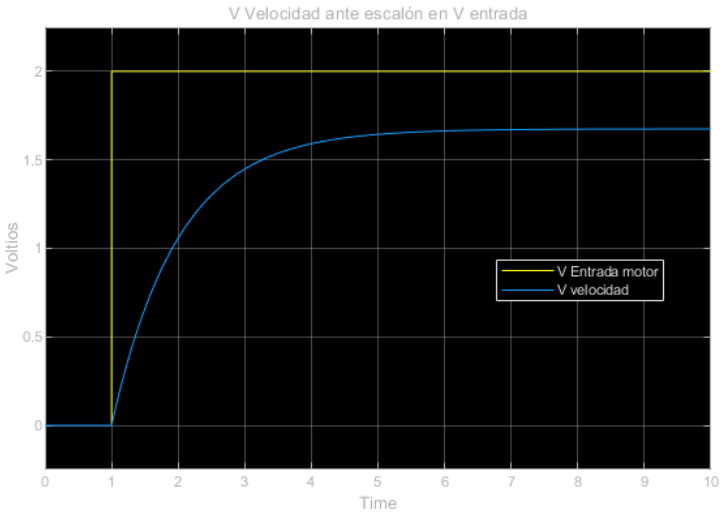


Ilustración 5 Respuesta ante escalón sin controlador. Fuente: Elaboración propia

2.5 PLC Siemens S7 1200

Un controlador lógico programable, o PLC por sus siglas en inglés (*Programmable Logic Controller*), es un ordenador industrial especializado en el control de procesos automáticos. Cuenta con interfaces para la recepción y envío de señales de entrada y salida y suele ser resistente a gran parte de las condiciones adversas que se pueden encontrar en una planta industrial.

El PLC ejecuta el programa que ha sido escrito previamente en su memoria, transferido desde un ordenador personal. Con este código y la información recopilada de las entradas, el autómata lleva a cabo un proceso de control cíclico, gobernando el sistema mediante las señales de salida.

El modelo utilizado para este proyecto es SIEMENS 1214C AC/DC/Rly. Es un PLC compacto y de configuración flexible, que permite añadir módulos para funcionalidades adicionales.

Las principales características de este modelo, extraídas de la documentación del PLC [3], son:

- Dimensiones: 110 x 100 x 75 mm.
- Memoria de usuario: 100KB de trabajo y 4MB de carga.
- Entradas y salidas integradas:
    - Digital: 14 entradas y 10 salidas.
    - Analógico: 2 entradas.
- Tamaño de memoria:
    - 1024 bytes para las entradas (I).
    - 1024 bytes para las salidas (Q).
    - 8192 bytes para el resto de las variables o marcas (M).

Adicionalmente se ha utilizado una tarjeta de señales AQ 1x12BIT, que permite una salida analógica, usada para controlar el voltaje de entrada al motor eléctrico.

Permite, entre otras, comunicaciones Profinet y Modbus TCP/IP mediante conexión ethernet. Estas son los protocolos de comunicación que se utilizarán para intercambios de información entre ambos PLC y el HMI y entre PLC y SCADA, respectivamente.

La programación de los autómatas se ha llevado a cabo usando el software de Siemens *TIA Portal V13*, usando una licencia cedida por la UPV. Este programa también permite simular los PLC de manera virtual con su extensión *S7-PLCSIM V13*.

Adicionalmente, para poder simular las comunicaciones entre el SCADA de *Matlab* y el autómata, se ha usado *NetToPLCsim*. Este software gratuito, permite crear una interfaz virtual de comunicaciones para conectar la simulación con la red.

2.6 Centro de control. SCADA Matlab.

Un SCADA (*Supervisory Control And Data Acquisition*) es un sistema que permite supervisar y controlar un proceso a distancia en tiempo real. Recopila información sobre el estado del proceso, integrando datos recogidos de diversas fuentes, y permite actuar directamente y de manera inmediata sobre él. En este caso la supervisión se llevará a cabo desde el centro de control, a través de una interfaz gráfica.

Por tanto, esta aplicación se encargará de recoger los datos de ambos PLC, los mostrará al usuario y coordinará el proceso en consecuencia. Además, permitirá controlar el funcionamiento del sistema y la producción, gestionar las emergencias y diagnosticar los fallos que ocurran en la línea.

La programación del SCADA se ha realizado en *AppDesigner*, una extensión de *Matlab*. Este programa permite crear una interfaz mediante una programación orientada a elementos gráficos, utilizable desde un PC. Tiene la ventaja, frente a otras aplicaciones, de contar con la flexibilidad y la potencia de cálculo de *Matlab*, permitiendo el uso de todas sus librerías especializadas. La licencia de este programa también ha sido proporcionada por la UPV.

La conexión con los PLC se realizará a través de un cable ethernet y las comunicaciones utilizarán el protocolo MODBUS TCP/IP.

2.7 HMI

La función del HMI es recopilar información básica sobre el estado del proceso para mostrarla de manera ordenada y sencilla. También es un elemento de apoyo para las operaciones de mantenimiento. Se encuentra situado junto a la línea productiva, siendo fácilmente accesible a los operarios. Permite, además, cierto control sobre el sistema, para poder realizar el arranque y la parada de la línea, así como para solucionar pequeños problemas que puedan surgir.

Se encuentra conectado a los PLC mediante Ethernet, utilizando el protocolo Profinet.

El modelo elegido para este proyecto es el KTP700 Basic PN de SIEMENS. Este modelo se ha utilizado únicamente desde la simulación que permite llevar a cabo *TIA Portal*.

2.8 Comunicaciones

2.8.1 Profinet [4]

Se trata de un protocolo de comunicación diseñado especialmente para el intercambio de datos entre dispositivos (Sensores, bloques de entrada y salida, etc.) y controladores (PLC) a través de cable ethernet industrial. Parte de los protocolos PROFIBUS y Ethernet doméstico. Además, ha sido concebido para ser usado en entornos industriales y proporcionar la velocidad necesaria por estos sistemas.

Hace uso de tres canales de comunicación distintos, que pueden ser usados de manera simultánea:

- El Standard de TCP/IP (NRT): Comunicaciones normales entre dispositivos y transmisiones de grandes cantidades de información.
- Tiempo real (Profinet RT): Se salta las capas TCP/IP del protocolo para permitir una gran velocidad de transmisión, en el rango de 1 a 10 ms. Especialmente útil para transmisiones de entradas y salidas.
- Tiempo real isócrono (Profinet IRT): Mecanismos adicionales que permiten sincronización de alta precisión para aplicaciones como, por ejemplo, control de movimientos.

En la siguiente imagen se puede observar el modelo OSI utilizado por cada uno de los canales:

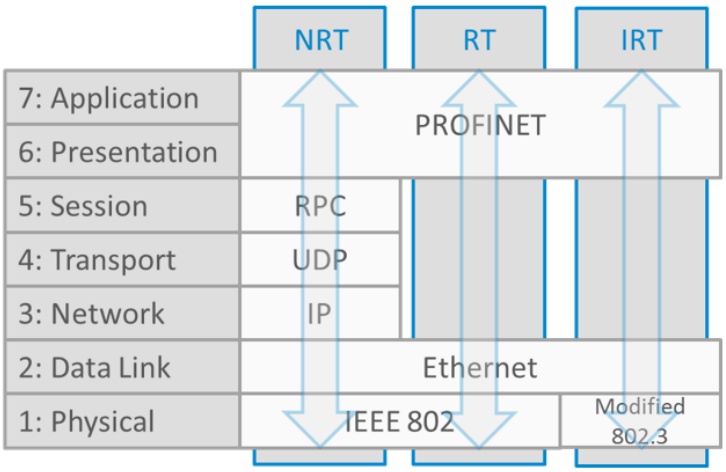


Ilustración 6 Modelo OSI canales. Fuente: Página web Profinet

Se ha elegido este protocolo para la comunicación PLC-PLC y PLC-HMI, para simular un entorno industrial moderno y eficiente.

2.8.2 Modbus [5]

Se trata de un protocolo diseñado para la comunicación entre servidores y clientes conectados a una misma red Ethernet TCP/IP. Está basado en intercambio de información a través de cuatro tipos de mensaje:

- Petición MODBUS: el CLIENTE envía una petición al servidor, indicando qué información requiere.
- Indicación MODBUS: el mensaje de petición recibido por el SERVIDOR.
- Respuesta MODBUS: mensaje de respuesta enviado por el SERVIDOR. Contiene la información pedida por el cliente.
- Confirmación MODBUS: el mensaje de respuesta del servidor recibido por el CLIENTE.

Fue diseñado también para su uso en entornos industriales para comunicación de PLC y proporciona bastante flexibilidad, ya que impone pocas restricciones.

En cada mensaje, además de la cabecera usada para el direccionamiento, se pueden distinguir dos tipos de bloque: el código de función (leer bit, leer bits, leer palabra, escribir bit, etc.) y la información (ya sea la dirección a la que se quiera acceder o la información almacenada en dicha dirección).

Este protocolo distingue cuatro tipos de datos:

- *Discrete input*: Bit de lectura. Ejemplo: entradas del PLC (I)
- *Coil*: Bit de lectura/escritura. Ejemplo: salidas del PLC (Q)
- *Input register*: palabra de 16 bits de lectura. Ejemplo: valor de una entrada analógica.
- *Holding register*: palabra de 16 bits de lectura/escritura. Ejemplo: contador con el número de piezas producidas.

En los PLC Siemens se utiliza un bloque específico (MB_Server) para crear el servidor Modbus. Este bloque permite también especificar en qué dirección empiezan los *Holding Register*. La siguiente imagen muestra algunos ejemplos de los códigos de función, los códigos de dirección Modbus y el área de memoria de la CPU a la que hacen referencia dichas direcciones [3].

Funciones Modbus						S7-1200	
Códigos	Función	Área de datos	Rango de direcciones			Área de datos	Dirección de la CPU
01	Leer bits	Salida	1	a	8192	Memoria imagen de proceso de las salidas	Q0.0 a Q1023.7
02	Leer bits	Entrada	10001	a	18192	Memoria imagen de proceso de las entradas	I0.0 a I1023.7
04	Leer palabras	Entrada	30001	a	30512	Memoria imagen de proceso de las entradas	IW0 a IW1022
05	Escribir bit	Salida	1	a	8192	Memoria imagen de proceso de las salidas	Q0.0 a Q1023.7
15	Escribir bits	Salida	1	a	8192	Memoria imagen de proceso de las salidas	Q0.0 a Q1023.7

Ilustración 7 Mapeo de direcciones Modbus. Fuente: Manual PLC S7 1200, Siemens

Se ha elegido este protocolo porque es uno de los más usados en la industria actual.

3. Programación de los PLC

3.1 Estructura del programa

Todas las funciones de los PLC han sido programadas utilizando lenguaje Ladder (Diagrama de contactos). Este lenguaje de programación es especialmente útil para llevar a cabo operaciones secuenciales y programas de este tipo, ya que es muy visual y de fácil interpretación. Al no necesitar ningún tipo de operación especial o complicada (prácticamente sólo se trabaja con números enteros, booleanos y condicionales), no hace falta un lenguaje de más alto nivel. Esto hace que encontrar y solucionar los posibles fallos de código sea mucho más sencillo.

El principal objetivo durante la fase de diseño del programa ha sido mantener una programación modular, de manera que fuese fácilmente editable en caso de que se modificase el proceso, y para poder replicar el programa en robots similares.

Para lograr este objetivo, se ha realizado una programación orientada a funciones. Cada proceso se ha separado en bloques de función (FC) simples que realizan una única instrucción (movimiento, accionamiento de la ventosa, etc.). También se han organizado los datos en bloques según su uso y procedencia en Bloques de Datos (DB). Así las variables destinadas a, por ejemplo, la comunicación entre PLC, tienen su propio DB.

En cuanto al funcionamiento automático del sistema, el programa está dividido en pasos. Cada paso ejecuta una o varias instrucciones y espera ciertas condiciones para avanzar al siguiente. Así se puede modificar el comportamiento del robot sin tener que hacer grandes cambios estructurales en la programación. De esta manera, el programa base para los dos prototipos es el mismo, cambiando únicamente las instrucciones específicas de cada paso del proceso y algunas variables auxiliares y de comunicación.

En la siguiente ilustración se muestra un esquema del funcionamiento del PLC.

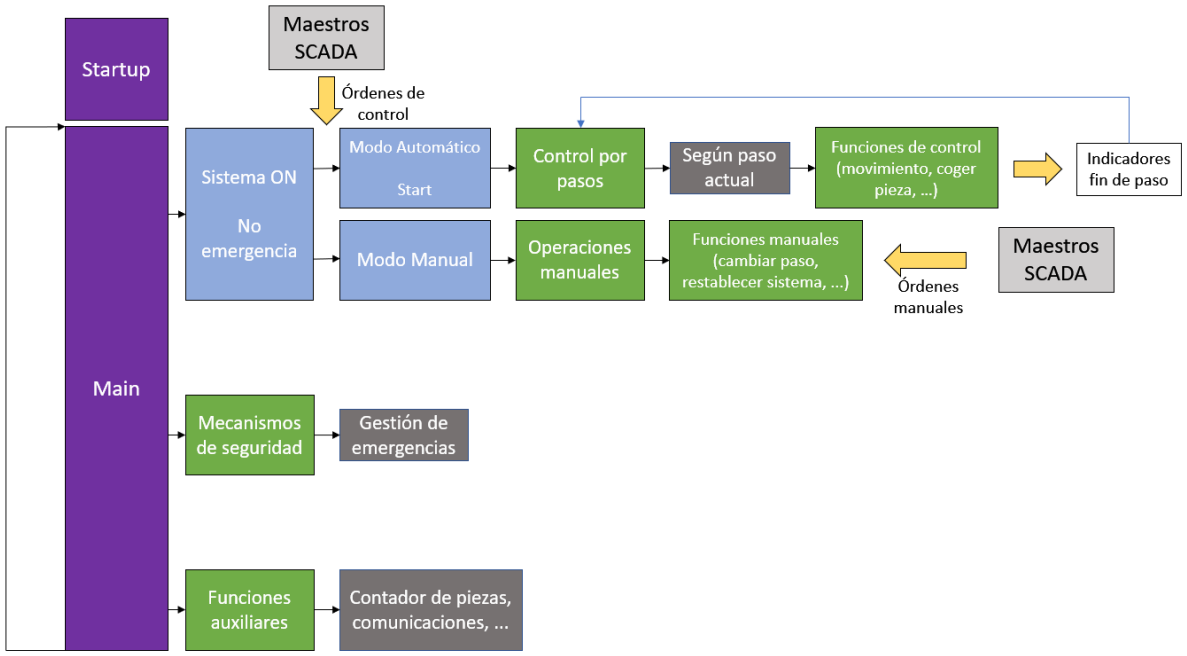


Ilustración 8 Estructura del programa PLC. Fuente: Elaboración propia

Como se puede observar en la imagen anterior, el programa tiene un único bucle de control que se repite de manera continua. Según el modo de funcionamiento y el estado del sistema distintas funciones son llamadas, para realizar las operaciones pertinentes. Además, los mecanismos de seguridad y las funciones auxiliares se llaman en todo momento. En el siguiente apartado se explicará con más detalle el funcionamiento de cada una de las partes.

Por último, cabe destacar que este estilo de programación por funciones se asemeja más a lo que se podría encontrar en una línea real. Los robots usados en industria cuentan con un procesador programable propio que controla movimientos, frenos, motores, etc. En este caso el PLC sólo especifica qué orden ha de ejecutar el robot (por ejemplo, moverse a la posición 3) y es el propio robot el que se encarga de controlar sus procesos. Teniendo esto en cuenta, el programa utilizado aquí podría ser fácilmente trasladable a un proyecto real, reemplazando las funciones de control del robot con bloques de comunicación que enviasen la orden pertinente.

3.2 Modos de funcionamiento

On/Off

En primer lugar, el sistema puede estar encendido o apagado. Según la aplicación real, el funcionamiento de estos estados podría variar. Por ejemplo, podría hacer referencia a la alimentación de electricidad, a la capacidad para actuar sobre el sistema, etc.

En este caso, debido a la imposibilidad de cortar la alimentación de manera directa, el propósito es el de bloquear todos los movimientos y evitar que se puedan modificar el resto de los parámetros del sistema.

De esta manera, sólo si el sistema está encendido se podrán ejecutar los procesos y cambiar el modo de funcionamiento (manual o automático).

Manual/Automático

Como en la mayoría de los procesos automatizados, existen dos modos de funcionamiento.

El modo automático ejecuta de manera autónoma una serie de órdenes, según la información que recopila del sistema y del proceso en cuestión (sensores, paso, etc.). Este es el modo de funcionamiento principal que estará activo durante la mayor parte del proceso productivo. Como se puede observar en la ilustración anterior (Ilustración 8), mientras este modo esté activo se llevará a cabo un control por pasos. Según el paso del proceso en el que se encuentre el sistema, el PLC dará las órdenes pertinentes al robot. Durante la ejecución de estas el autómata monitoriza las condiciones que se han de cumplir para avanzar de paso.

El modo manual sirve para solucionar posibles problemas que puedan surgir en el funcionamiento automático, así como para controlar el sistema durante las fases de diseño o reajuste de este. Son, en resumen, las operaciones llevadas a cabo por una persona actuando directamente sobre el sistema, a través del HMI o del centro de control. En este modo se puede cambiar el paso actual, restablecer el sistema a condiciones iniciales, descartar piezas defectuosas y controlar manualmente la posición del brazo manipulador. Además, desde el centro de control se puede modificar el modelo a producir, así como el objetivo de producción.

Emergencia

En cualquier proceso industrial existe la posibilidad de que se produzcan accidentes o imprevistos, desde emergencias que afecten a toda la planta industrial (un incendio), hasta pequeños problemas localizados en una única parte de la línea. Además, en estos procesos suele estar involucrada maquinaria potencialmente peligrosa. Por esta razón deben existir mecanismos de seguridad.

Hay una gran variedad de medidas de seguridad que pueden activar este estado de alarma, como por ejemplo barreras láser, escáneres, puertas, etc. De entre todos ellos, el más sencillo y utilizado es la seta de emergencia. Un número suficiente de estos pulsadores ha de estar distribuido por toda la línea industrial, permitiendo el fácil acceso y accionamiento desde cualquier punto de los alrededores. Para

simular esto, se usará una seta de emergencia en el HMI (que representa a todas las setas de emergencia de la línea) y un botón de emergencia en el centro de control.

En general, todos los mecanismos de seguridad tienen un mismo objetivo: detener de manera inmediata el funcionamiento de la línea, ya sea accionando frenos, desactivando los movimientos o quitando la alimentación.

Debido a las limitaciones de equipamiento, y ya que se trata de una maqueta de pequeña escala, se han concentrado todos los posibles mecanismos de seguridad del sistema en las dos setas de emergencia previamente citadas.

Por tanto, al activar cualquiera de estos accionadores, el PLC entrará en modo Emergencia. Esto desactiva los modos de funcionamiento habituales (manual y automático) y cancela todos los movimientos. Cabe destacar que no se desactivan el compresor y la ventosa, ya que esto podría causar que la pieza cayese donde no debe, generando potencialmente una nueva amenaza.

Para desactivar el modo Emergencia, ambas setas deben estar desactivadas. Además, como mecanismo extra de seguridad, se debe restablecer el sistema. En la siguiente ilustración se explica el funcionamiento de las emergencias mediante un diagrama de bloques.

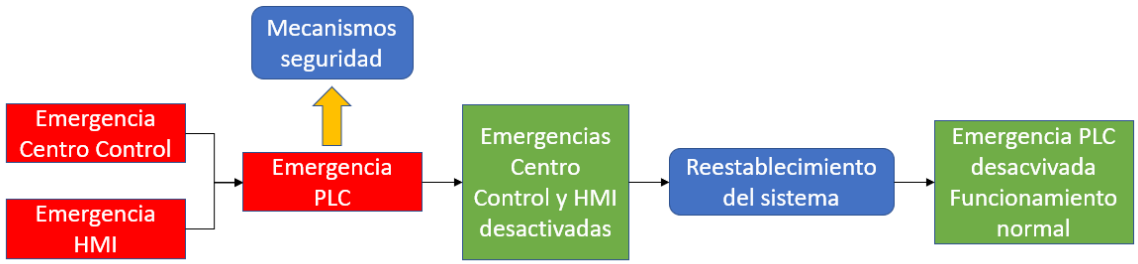


Ilustración 9 Esquema emergencias. Fuente: Elaboración propia

3.3 Funciones Utilizadas

Como ya se ha explicado anteriormente, la programación del PLC se ha orientado al uso de funciones. En TIA Portal existen cuatro tipos de bloques:

- Bloque de organización (OB): son los bloques que se encargan del bucle de control, así como de eventos especiales. En este programa usamos cuatro tipos:
    - Inicio (*Startup*): Bloque que se ejecuta una única vez cuando el PLC pasa a modo RUN.
    - Ciclo del programa (*Program cycle*): Bloque que se ejecuta de manera cíclica, en bucle continuo. Desde aquí se llaman al resto de funciones. Se trata del bloque principal del programa
    - Interrupción de Hardware (*Hardware Interrupt – HI*): Cuando se recibe señal de una determinada entrada del PLC se llevan a cabo las instrucciones especificadas.
    - Interrupción cíclica (*Cyclic Interrupt*): Bloque que se ejecuta con un periodo predefinido. Sirve para llevar a cabo operaciones repetitivas independientes del resto del programa. En este caso se utiliza para el control PID del motor.
- Bloque de función (FB): funciones que guardan de manera permanente sus variables en un bloque de datos, de manera que se sigan conservando después de la ejecución completa de la función.
- Funciones (FC): como el anterior, pero sin memoria dedicada. Las variables de bloque usadas en estas funciones son temporales y pierden su valor una vez se termina de ejecutar.
- Bloques de datos (DB): bloques de memoria que guardan datos.

Como se puede observar en las tablas siguientes, las FC que se han usado para el programa están clasificadas en cinco tipos:

- AUTO: funciones que recogen el funcionamiento en modo automático de la línea.
- AUX: funciones auxiliares al sistema. Aportan información extra al sistema o llevan a cabo operaciones secundarias, como las comunicaciones.
- INS: funciones de instrucción. Ejecutan una orden simple, como por ejemplo mover el robot a unas coordenadas específicas. Estos bloques son llamados por las funciones AUTO y MAN.
- MAN: funciones manuales. Lllevan a cabo las operaciones manuales ordenadas desde el maestro SCADA y el HMI.
- SEG: funciones de seguridad. Sirven para desactivar movimientos y gestionar emergencias.

También se han usado tres funciones del tipo HI que gestionan los pulsos de entrada de los encoders (Entradas B1, B3, B5).

A continuación se muestra un resumen de las funciones utilizadas. Para más detalle acudir al Anexo I: Programación PLC.

Función	Tipo	Descripción	Funciones llamadas
AUTO_Control_Pasos	FC	Funcion principal de control. Según paso actual ejecuta las ordenes correspondientes y una vez completadas cambia de paso. Requiere modo AUTO y START	INS_Posicion_Home, INS_Movimiento_Robot, INS_Coger_Pieza, INS_Dejar_Pieza, INS_Activar_Compresor
AUX_Comunicaciones	FC	Gestiona comunicaciones entre PLCs, MASTER SCADA y HMI	
AUX_Contador_Piezas	FC	Cuenta las piezas completadas o defectuosas de cada modelo.	
AUX_Numero_Pasos	FC	Asigna un valor a StepNo según el paso actual	
AUX_Restablecer_Encoders	FC	Al llegar al final de carrera, los contadores de los encoders se reinician	
AUX_Restablecer_Sistema	FC	Devuelve el sistema al paso 1 y restablece variables	
AUX_SetPoint_Motor	FC	Modifica la velocidad a la que gira el motor DC que simula los motores del manipulador. Tres velocidades: Parada, media, rápida.	
AUX_Tiempos_Ciclo	FC	Cronometra el tiempo en completar un ciclo entero del programa	
INS_Activar_Compresor	FC	Activa el compresor de aire para poder utilizar la ventosa. Introduce un retardo para dar tiempo a que se encienda.	
INS_Coger pieza	FC	Acciona la ventosa para coger la pieza.	
INS_Dejar pieza	FC	Deja de accionar ventosa para dejar la pieza. Apaga compresor.	
INS_Movimiento_Robot	FC	Mueve el robot a la posición objetivo y modifica la velocidad del motor según parámetro de entrada	

Tabla 3 Resumen funciones PLC (1) . Fuente: Elaboración propia

INS_Posicion_Home	FC	Mueve el robot a la posición de home	
MAN_Cambiar_Paso	FC	Si el sistema está en modo manual, el MASTER puede modificar el paso actual, dando orden de aumentar o disminuir.	
MAN_Ir_A_Posicion	FC	Si el sistema está en modo manual, el MASTER puede dar orden de ir a una posicion específica.	INS_Posicion_Home, INS_Movimiento_Robot
MAN_Modo_Manual	FC	Llama a las funciones manuales si el modo Manual está activo. También permite hacer RESET (Poner en sistema en paso 1 y reiniciar variables)	MAN_Aumentar_Reducir_Paso, MAN_Ir_A_Posicion, AUX_Restablecer_Sistema
SEG_Desactivar_Movimientos	FC	Desactiva las salidas de movimiento al robot. No desactiva ventosa y compresor. Desactiva el START.	
SEG_Mecanismos_Seguridad	FC	Gestion de emergencias y mecanismos de seguirdad.	SEG_Desactivar_Movimientos
Main[OB1]	Program_Cycle	Loop principal de control	Restablecer_Sistema, Control_Pasos, Modo_Manual, Contador_Piezas, Tiempos_Ciclo, Mecanismos_Seguridad, Numero_Pasos, Reset_Encoders, Comunicaciones, SetPoint_Motor
OB_HI_B1_vertical[OB40]	HI	Contador encoder vertical. Da la posición del robot en el eje vertical.	
OB_HI_B3_horizontal[OB41]	HI	Contador encoder horizontal. Da la posición del robot en el eje horizontal.	
OB_HI_B5_giro[OB42]	HI	Contador encoder giro. Da la posición del robot en el giro.	
Startup[OB100]	Startup	Inicializar variables. Sistema en paso 1.	
Control_Motor	CI	Lleva a cabo el control de la velocidad del motor y, en caso de estar activada, la simulación del motor.	

Tabla 4 Resumen funciones PLC (2) . Fuente: Elaboración propia

3.4 Pasos

El funcionamiento en modo automático de los robots está gobernado por un sistema de pasos. Cada paso representa la siguiente acción que debe llevar a cabo el sistema. Están formados por la orden a ejecutar y por las condiciones que se han de cumplir antes de pasar al siguiente. En general, la condición para avanzar al siguiente paso es que se completen las órdenes del actual (por ejemplo, que el robot coja la pieza o que llegue a la posición objetivo), pero también pueden incluir información externa (por ejemplo, que haya una nueva pieza en la estación inicial). En la siguiente ilustración se muestra una explicación gráfica:

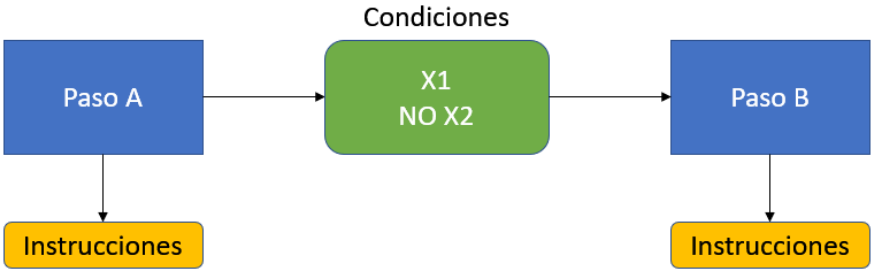


Ilustración 10 Esquema cambio de paso. Fuente: Elaboración propia

Cada uno de los manipuladores lleva a cabo un proceso distinto, por tanto, los pasos que siguen no son exactamente los mismos.

Con el fin de poder modificar y ampliar fácilmente el programa de cada robot, se han creado más pasos de los estrictamente necesarios. Estos pasos adicionales están marcados como RESERVA y no ejecutan ninguna orden. Cada programa cuenta con 30 pasos. Del 1 al 24 se utilizan para el funcionamiento normal del sistema y del 25 al 30 para las operaciones de descarte de piezas.

En la siguiente tabla se muestran los pasos de ambos robots:

Paso	PLC1	PLC2
1	Llevar robot a Home	Llevar robot a Home
2	Esperar orden de nueva pieza	Esperar orden de nueva pieza
3	Ir a la posicion de coger pieza	Ir a posición de coger pieza
4	Bajada a pieza y activar compresor	Bajada a pieza y activar compresor
5	Coger pieza	Coger Pieza
6	Levantar pieza	Levantar pieza
7	Ir a posicion de dejar la pieza (Segun modelo)	Ir a posición de dejar pieza (Segun modelo)
8	Bajada vertical a dejar pieza	RESERVA
9	Dejar pieza en estacion intermedia	RESERVA
10	Subida vertical a posicion de aproximacion	RESERVA
11	Ir a posicion de espera (Segun modelo)	RESERVA
12	Esperar proceso completado	RESERVA
13	Ir a posicion de aproximacion a pieza (Segun modelo)	RESERVA
14	Bajada a pieza y activar compresor	RESERVA
15	Coger Pieza	RESERVA
16	Levantar Pieza	RESERVA
17	Ir a posicion aproximacion dejada final (Al robot 2)	RESERVA
18	RESERVA	RESERVA
19	RESERVA	RESERVA
20	RESERVA	RESERVA
21	RESERVA	RESERVA
22	Bajada vertical a dejar pieza	Bajada vertical a dejar pieza
23	Dejar pieza en posicion final	Dejar pieza en posicion final
24	FIN CICLO: Subida vertical	FIN CICLO: Subida vertical
25	PIEZA DEF. Ir a Posicion descartar	PIEZA DEF. Ir a Posicion descartar
26	PIEZA DEF. Bajada vertical a dejar pieza	PIEZA DEF. Bajada vertical a dejar pieza
27	PIEZA DEF. Dejar pieza	PIEZA DEF. Dejar pieza
28	PIEZA DEF. Subida vertical	PIEZA DEF. Subida vertical
29	RESERVA	RESERVA
30	RESERVA	RESERVA

Tabla 5 Pasos robots. Fuente: Elaboración propia

Se puede observar que el robot 2 tiene menos pasos ocupados que el robot 1, ya que su proceso es más sencillo. Además, la mayoría de los pasos de ambos son iguales. Esto es posible gracias a la programación modular, que permite reutilizar gran parte del código. Cuando se llega al final del ciclo (ya sea en funcionamiento normal o en descarte de pieza) el PLC vuelve al paso 1.

En cuanto al proceso de descarte, una vez el PLC recibe la información de que la pieza es defectuosa, continúa el funcionamiento normal hasta llegar a algún paso en el que se haya cargado la pieza (pasos 7 y 17 en PLC1, 7 en PLC2) y entonces da el salto al paso 25.

Para más información sobre las condiciones de cambio de paso, acudir al Anexo I: Programación PLC.

3.5 Comunicaciones entre PLC

La única interacción entre ambos robots es la estación intermedia R1R2 (como se muestra en la Ilustración 2). En una línea industrial, lo más probable es que se usaran sensores que detectasen la pieza e informasen al robot 2 de que puede iniciar su proceso. Sin embargo, como en la maqueta no se dispone de dicho equipamiento, es el PLC1 el que informa al PLC2 de que se ha depositado una pieza. A su vez, el segundo informa al primero cuando se ha retirado la pieza, para evitar que este intente depositar dos piezas en el mismo lugar de manera simultánea. De esta manera, ambos PLC mantienen una comunicación que les informa si hay pieza o no en la estación R1R2.

Para asegurar que la comunicación entre ambos autómatas está activa en todo momento, cada uno tiene un bit de vida que manda al otro. Este bit de vida parpadea (cambia de 0 a 1) con una frecuencia de 0.5Hz.

Además, para posibles ampliaciones en el futuro, se han dejado en reserva 6 bits más que se envían entre ambos PLC, de manera que, en total, se envía un byte.

Estos intercambios de información entre los PLC se llevan a cabo mediante comunicaciones PUT/GET, un protocolo específico de los autómatas S7 de SIEMENS que utiliza la conexión Profinet.

3.6 Diagnóstico

Uno de los aspectos más importantes que se debe cumplir en cualquier proceso industrial es la capacidad de diagnosticar los fallos de manera precisa y rápida, para poder solucionarlos con seguridad y rapidez. Para ello es crucial que el sistema informe al operario en qué punto del proceso está y cuál es el error que ha ocurrido. Encontrar el error en un sistema con pocos elementos, como este, es fácil. Sin embargo, en una línea productiva real, con cientos de sensores, accionadores, elementos móviles, etc., puede ser muy complicado si no hay un sistema de diagnóstico adecuado.

Para abordar este aspecto, se ha introducido en los PLC un sistema de Matriz de Fallos. Es importante, por tanto, definir qué se considera como fallo. En este proyecto se pueden diferenciar dos tipos: fallos de paso y errores.

Fallos de paso

Ya se ha explicado en el apartado de Pasos que el sistema sólo avanzará de uno a otro si se cumplen las condiciones necesarias. De este concepto surge otro complementario: los fallos. Estos podrían definirse como las condiciones a “eliminar” para pasar al siguiente paso o, lo que es lo mismo, el contrario a dichas condiciones. Por ejemplo, como se puede observar en la ilustración 11, el paso uno es llevar el robot a la posición inicial de HOME. La condición para avanzar es llegar a dicha posición (*HomePosOk*), por tanto, el fallo correspondiente será que NO esté activa dicha condición.

Cabe destacar que el hecho de que la activación de un fallo no tiene por qué significar que el sistema no esté funcionando correctamente. Sin embargo, si el proceso se queda parado aporta información que puede ayudar a identificar el problema.

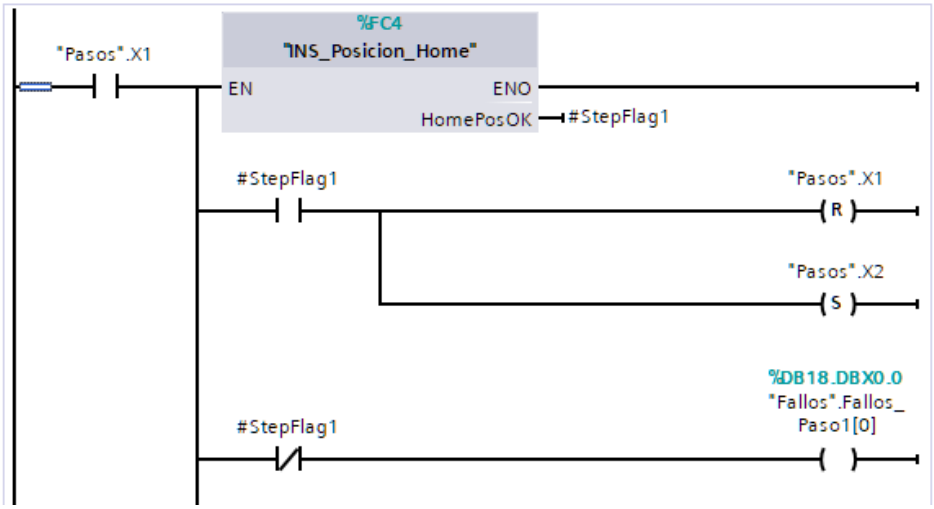


Ilustración 11 Ejemplo condiciones y fallos. Fuente: Elaboración propia

Ampliando el esquema de la ilustración 10 para añadir los fallos:

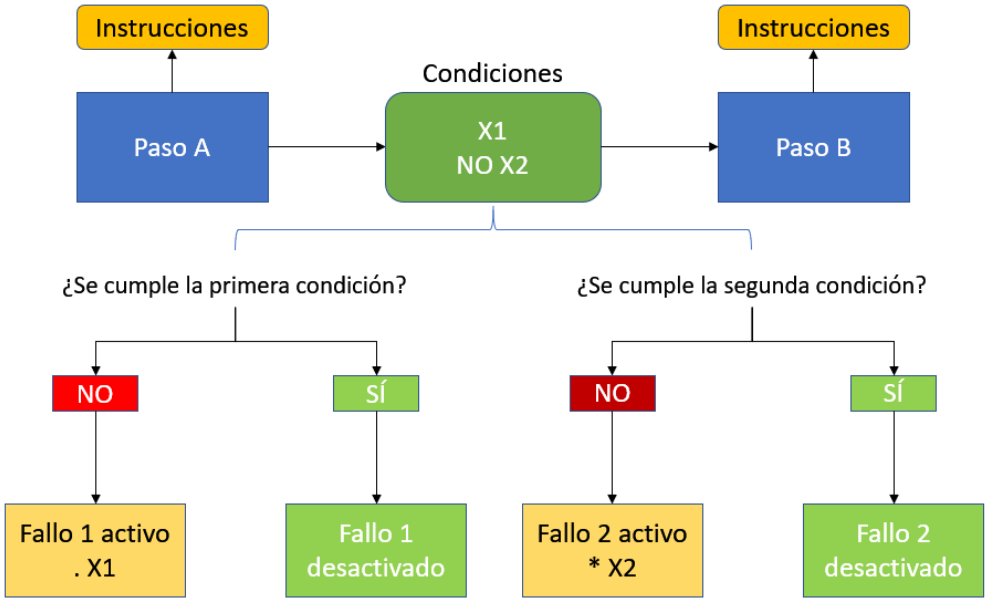


Ilustración 12 Esquema cambio de paso con fallos. Fuente: Elaboración propia

Se puede observar que aparece una nueva notación en los fallos. El "." que observamos en ". X1" hace referencia a que el sistema QUIERE que se active la variable X1 para avanzar de paso. El "*" que se observa en "* X2" hace referencia a que el sistema NO QUIERE X2 activa.

El sistema sólo avanzará de paso si se han cumplido todas las condiciones o, lo que es lo mismo, se han eliminado todos los fallos.

En resumen: los fallos indican qué se necesita para avanzar. De esta manera, si el proceso se queda parado es más fácil saber qué parte del sistema no ha funcionado como se esperaba y qué condición no se ha cumplido.

Cada paso tiene asignado un "byte de fallos", lo cual equivale a 8 posibles errores (En un byte hay 8 elementos que pueden valer 1 o 0).

Errores

Además de los fallos de paso, también hay otros sucesos que pueden afectar al funcionamiento normal del sistema y que es importante identificar. Para este proyecto se han clasificado en dos tipos, para cada uno de los cuales se ha asignado un byte.

El byte de comunicaciones diagnostica errores en este ámbito. Si, por ejemplo, un PLC deja de recibir el bit de vida del otro (detecta que el valor es fijo durante más de 1 segundo), se activará el primer bit de fallo de este autómata.

El byte general aporta información sobre otros posibles errores. Por ejemplo, si está la emergencia activada en el PLC, en el maestro SCADA o en el HMI, o si el PLC requiere un "Reset" para volver al modo de funcionamiento normal.

Estos 32 bytes forman una matriz de fallos de 256 componentes. Con esto se pueden comunicar una gran cantidad de errores enviando muy poca información a través de la red. Esta matriz es enviada al centro de control, cuya aplicación SCADA tiene información sobre el significado de cada bit. De esta manera, el PC muestra al operario cuáles son los fallos y errores actuales. La siguiente ilustración muestra un esquema de este proceso.

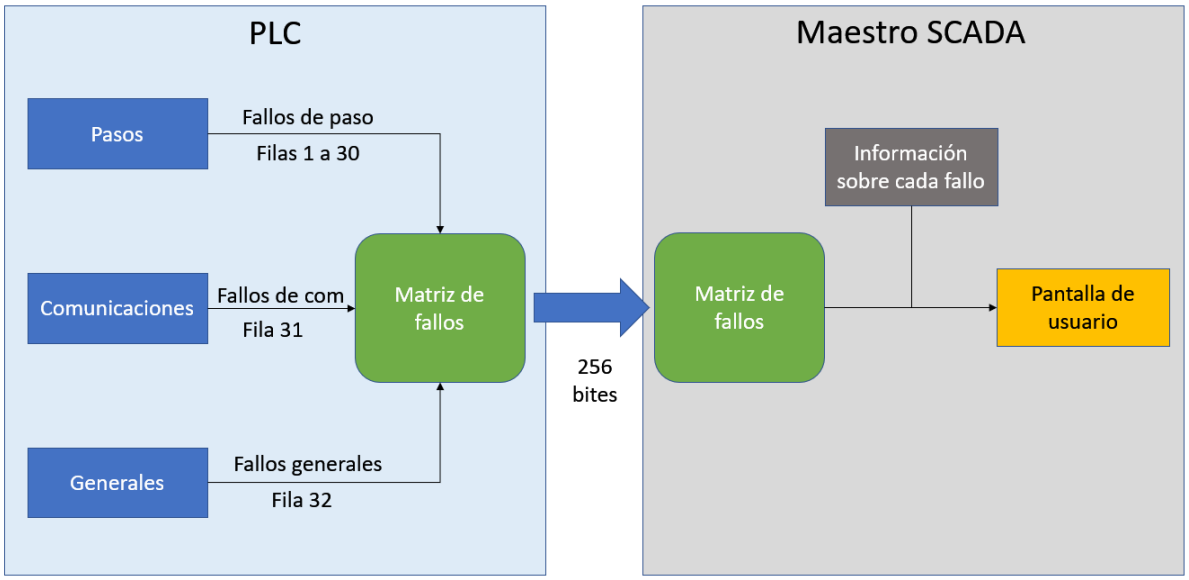


Ilustración 13 Esquema matriz de fallos. Fuente: Elaboración propia

Para más información sobre la matriz de fallos acudir al Anexo I: Programación PLC correspondiente.

3.7 Diseño e implementación del PID para control de los motores eléctricos

El control de los motores eléctricos se realiza con un PID y a través de señales analógicas intercambiadas entre estos y los PLC. Como se puede observar en la ilustración 5, el motor en bucle abierto tarda unos 5 segundos en alcanzar la velocidad de régimen permanente para un escalón de 2 Voltios. Además, una vez alcanzado este estado, presenta un error continuo de aproximadamente 0.3V. Con esta información podemos deducir que al menos necesitaremos un control PI, para anular el error en régimen permanente y reducir el tiempo de subida.

Ya que se va a necesitar un control continuo sobre el motor, el PID se ha implementado en un bloque de interrupción cíclica, que se ejecutará a intervalos constantes (de 100ms en este caso), independientemente del resto del programa. Para el controlador se utilizará el bloque PID_Compact. Este bloque coge como entrada el punto de funcionamiento deseado (Set Point) y el valor actual de la variable a controlar, y devuelve el valor de la salida modificada para controlar el proceso. En este caso:

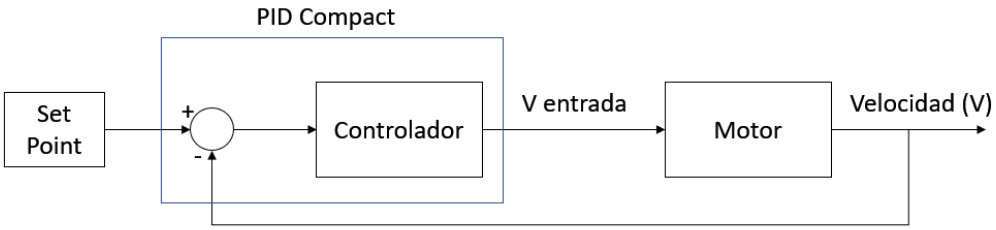


Ilustración 14 Control motor eléctrico. Fuente: Elaboración propia

Este bloque también se encarga de transformar los valores de las entradas/salidas analógicas a los valores reales utilizados en el proceso. Para ello se introduce la siguiente configuración:

- Las entradas/salidas analógicas del PLC tienen un rango de [0, 27648].
- Los valores de voltaje están en el rango de [0, 10] V.
- Los límites del valor de salida del PID (V entrada al motor) se va a fijar en [0, 8] V, para proteger al motor de sobre tensiones.
- Los límites para que salte el aviso de tensión alta/baja se van a fijar en [-0.5, 8] V. El aviso de tensión baja no se pone en 0, ya que este será el valor con el motor parado.

Una vez configuradas las entradas y salidas, el siguiente paso es establecer los parámetros de control del PID. El controlador tiene la forma [3]:

$$y = K_P\left[(b\cdot w - x) + \frac{1}{T_I\cdot s}(w-x) + \frac{T_D\cdot s}{a\cdot T_D\cdot s+1}(c\cdot w - x)\right]$$

Ecuación 2 PID Compact. Fuente: Documentación PLC s7 1200

Siendo “y” la salida del algoritmo (V entrada motor), “Kp” la ganancia proporcional, “s” el operador de Laplace, “b” la ponderación de la acción proporcional, “w” el Punto de funcionamiento elegido, “x” el

valor del proceso (Velocidad), “Ti” el tiempo de integración, “a” el coeficiente del retardo derivativo, “Td” el tiempo derivativo y “c” la ponderación de la acción derivativa.

TIA Portal proporciona una herramienta para la calibración inicial de los parámetros anteriores. Una vez conectado el PLC al proceso, se puede llevar a cabo optimización inicial y optimización fina. Tras esto se han modificado de manera empírica algunos parámetros para perfeccionar el ajuste. El controlador utiliza los valores siguientes:

Parámetro	Kp	Ti	Td	a	b	c
Valor	30	0,4	0,1	0,1	0,8	0

Tabla 6 Parámetros PID. Fuente: Elaboración propia

Con ello se consigue un control más rápido, con un tiempo de subida de aproximadamente 0.7 segundos, poco sobre pasamiento y sin error en régimen estacionario. La respuesta ante un escalón de 2 V es la siguiente:

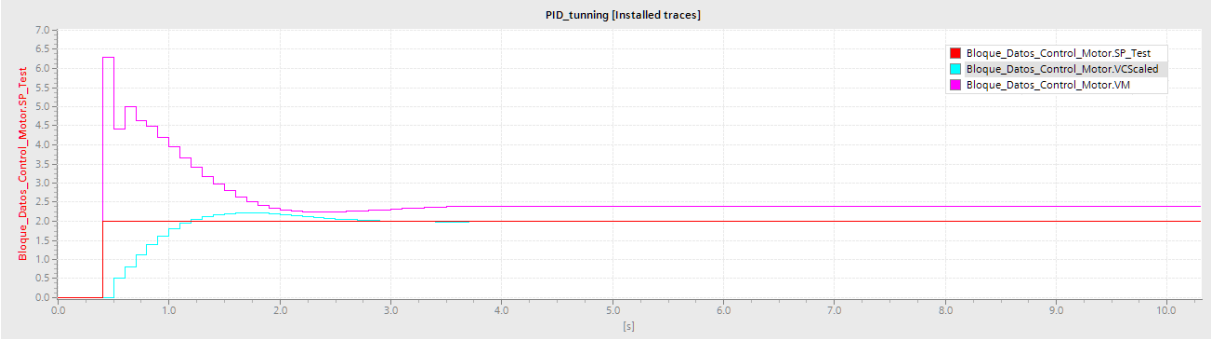


Ilustración 15 Respuesta ante escalón con controlador PID. Fuente: Elaboración propia

En la figura observamos el punto de funcionamiento elegido (SP_Test, en rojo), el valor de la velocidad en Voltios (VCScaled, en azul) y la tensión del motor (VM, en morado). El eje de este último parámetro va de 0 a 10 V. Como se puede observar, la respuesta no puede ser más rápida, debido a la limitación de voltaje del motor.

Las dos siguientes ilustraciones muestran la diferencia entre el sistema sin controlador y con control PID frente a cambios en el Set Point.

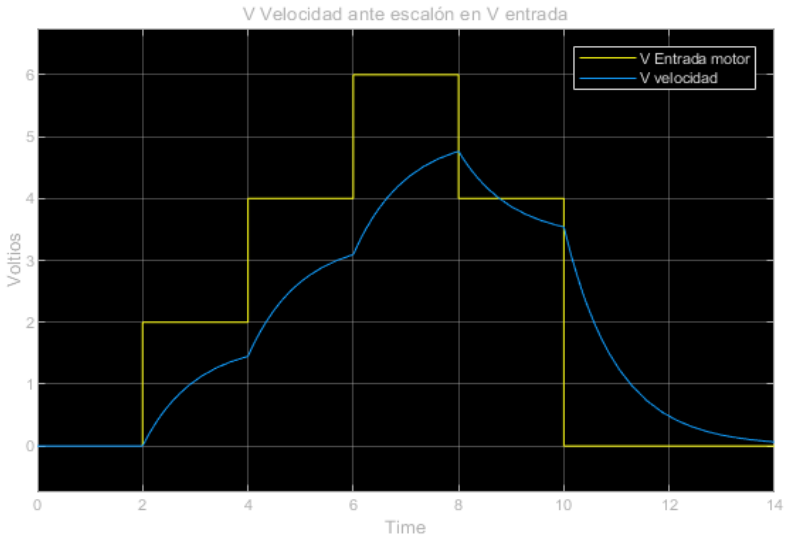


Ilustración 16 Respuesta ante escalones sin controlador. Fuente: Elaboración propia

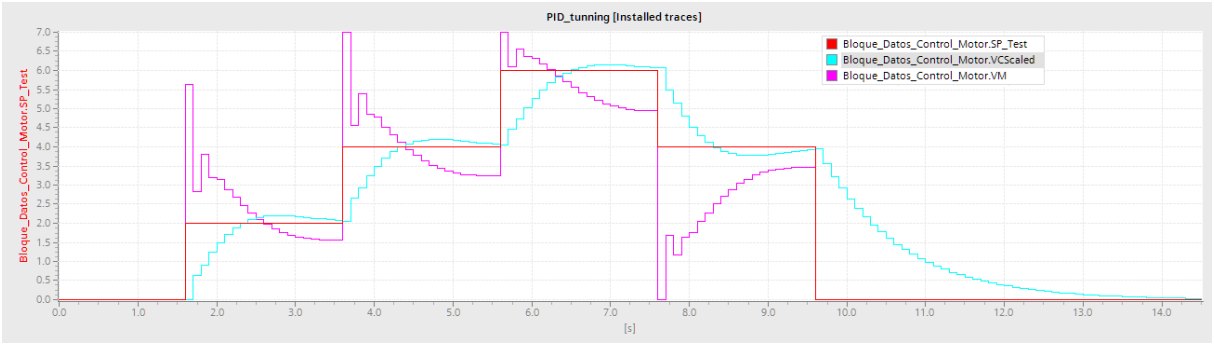


Ilustración 17 Respuesta ante escalones con control PID. Fuente: Elaboración propia

Cabe destacar que la respuesta ante escalones descendentes sería mucho más rápida en un sistema real, ya que contaría con un freno.

Durante el funcionamiento del proceso se han elegido tres puntos de funcionamiento posibles:

- Velocidad rápida: Para desplazamientos del robot entre estaciones. Set point 6 V.
- Velocidad lenta: Para operaciones de aproximación a cogida o dejada de pieza. Set point 3V.
- Parada: Motor parado. Set point 0 V.

Por último, como no ha sido posible acceder a los laboratorios, para llevar a cabo este proceso se ha utilizado un sistema simulado con la función de transferencia anteriormente explicada (Ecuación 1). Para ello se ha utilizado el bloque LSim. Para facilitar el cambio entre simulación y funcionamiento normal se ha habilitado un bit “Simular”, cuyo valor se puede cambiar desde la función Startup.

4. Programación del SCADA en Matlab

4.1 Función

El objetivo principal del SCADA es el de coordinar y controlar la línea productiva desde el centro de control. Sus principales funciones son:

- Monitorizar el sistema: aporta información sobre los sensores, el paso actual y el estado de la línea.
- Diagnóstico avanzado: muestra los errores y fallos que se pueden producir en la línea para que el ingeniero pueda localizar y solucionar el problema más fácilmente.
- Controles básicos: encender y apagar el sistema, iniciar la línea y cambiar el modo de funcionamiento (automático o manual).
- Controles manuales: permiten llevar a cabo operaciones especiales en el proceso. Cambiar el paso en el que se encuentran los robots, restablecer el sistema a condiciones iniciales, descartar piezas defectuosas y mover los manipuladores de forma manual.
- Control de producción: permite cambiar el modelo a producir y el objetivo de producción. También aporta información sobre la cantidad de piezas producidas de cada modelo y el tiempo de ciclo de cada robot.

Este sistema de control está pensado para ser usado únicamente por personal cualificado y sólo se puede acceder a él desde el centro de control, por tanto, permite operaciones más delicadas que el HMI a pie de línea. Se presupone que dicho centro cuenta con las medidas de seguridad necesarias para restringir el acceso al sistema SCADA. En la siguiente ilustración se puede observar la pantalla principal del sistema con algunas de sus utilidades.

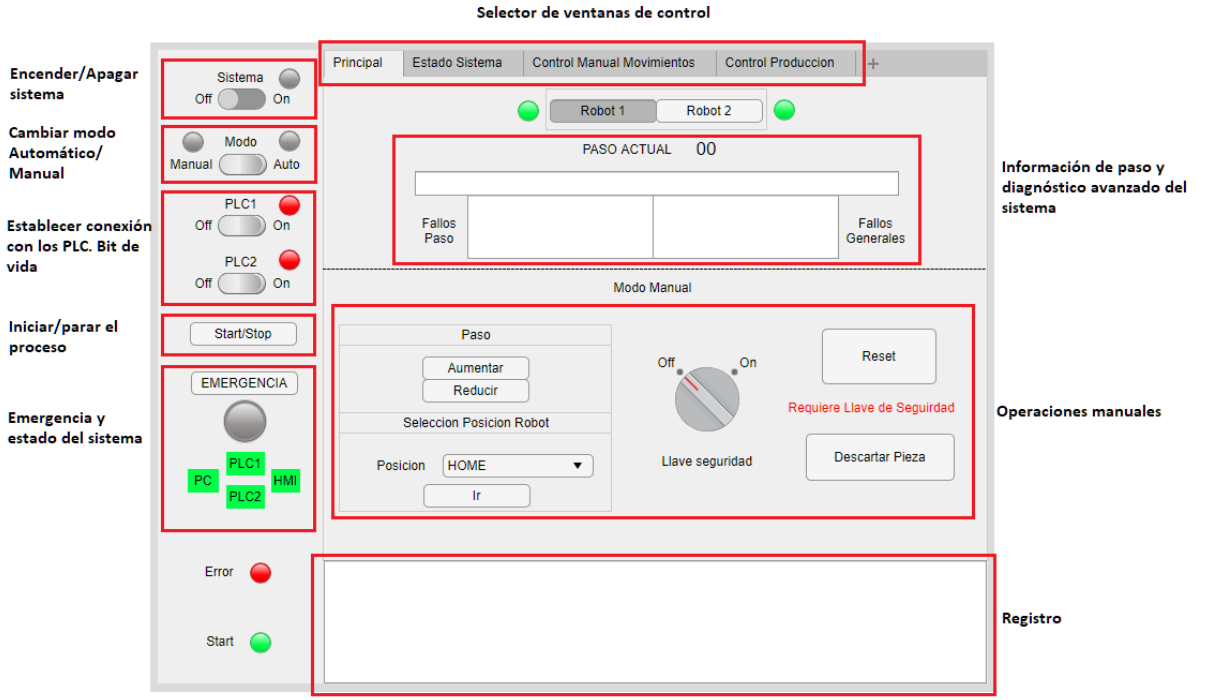


Ilustración 18 Pantalla principal SCADA. Fuente: Elaboración propia

4.2 Estructura del programa

El *AppDesigner*, como la mayoría de los programas para el diseño de interfaces gráficas, hace uso de una programación orientada parcialmente a elementos gráficos. Cada uno de estos elementos puede tener asignada una o varias funciones que se activan al interaccionar con él. Además de esto, tiene detrás toda la potencia de cálculo y flexibilidad de *Matlab*. Esto permite crear funciones más complejas y completas que en otros editores ya que, a diferencia de estos, permite escribir todo el programa en código, lo que da una mayor libertad a la hora de crear la aplicación.

Al igual que en la programación del PLC, se ha tratado de orientar el programa a funciones modulares con la intención de que sea fácilmente modificable y ampliable, en caso de añadir nuevos elementos al proceso.

Teniendo estos aspectos en cuenta, podemos distinguir tres tipos de funciones en el programa:

- Funciones de retro llamada (o *Callbacks*): ligadas a los elementos gráficos de la interfaz. Son ejecutadas cuando el usuario interactúa con dichos elementos. Por ejemplo: Acción al apretar un botón, al cambiar de estado un interruptor, al modificar un valor numérico, etc.

- Funciones auxiliares: son llamadas por las funciones anteriores o por procesos cíclicos dominados por un temporizador. Estas funciones llevan a cabo acciones que se repiten varias veces a lo largo del programa, como por ejemplo enviar o recibir información de los PLC o actualizar elementos gráficos (bombillas, texto, etc.).

- Funciones de aplicación: se ejecutan al principio o al final del programa. La gran mayoría son creadas automáticamente por el editor y sirven para crear los elementos gráficos. También se encuentran en esta categoría la función Startup (Llamada una única vez al ejecutar la aplicación) y la función de cierre (ejecutada cuando el usuario cierra la aplicación). Estas funciones se usan para inicializar variables y otros elementos del programa.

En la siguiente ilustración se muestra un esquema del funcionamiento del programa:

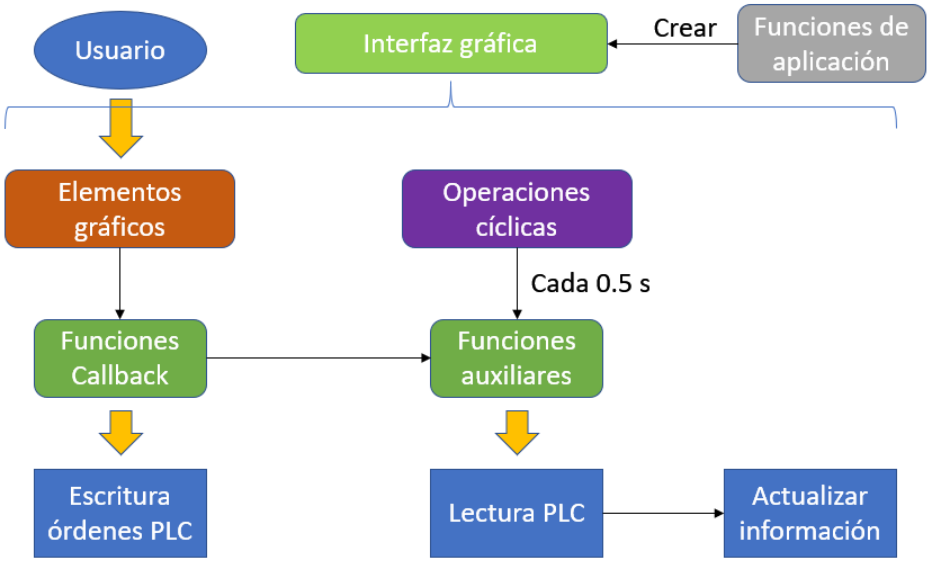


Ilustración 19 Funcionamiento SCADA. Fuente: Elaboración propia

Para contribuir a la modularidad del programa, se ha construido el código alrededor de la Clase PLC. Una clase es una herramienta de programación que permite crear una variable objeto con una estructura predefinida. De esta manera se pueden agrupar un número determinado de características bajo un mismo objeto.

Por ejemplo, una de las propiedades de la clase PLC es el vector Q, que recoge los valores de las salidas leídas del PLC. Para acceder a estos datos, el código sería:

$$app.PLC(n).Q$$

Donde *n* es el número del PLC. Con esto es mucho más sencillo poder reutilizar fragmentos de código para varios PLC, optimizando así el programa.

Para más información acerca de las funciones utilizadas y la clase PLC, acudir al Anexo II: Programación SCADA.

4.3 Comunicación con el PLC [6]

Al no ser una aplicación creada específicamente para la interacción con PLC, el *AppDesigner* no tiene ninguna herramienta para conectarse directamente al autómata. Sin embargo, Matlab dispone de una gran cantidad de librerías y herramientas para suplir esta dificultad.

La comunicación con los PLC se realiza a través de MODBUS TCP/IP. En este protocolo es necesario especificar la dirección de memoria a la que se quiere acceder según los parámetros explicados anteriormente. Sin embargo, la librería "IP-S7-LINK SIMATIC S7" permite utilizar los mismos punteros para definir la dirección que el PLC. Esto se debe a que está parametrizada en función del modelo de los autómatas SIEMENS, facilitando en gran medida el proceso.

En primer lugar, se han de establecer los parámetros de comunicación (modelo y dirección IP) del servidor al que se quiere acceder. Tras esto, establecer la conexión. Una vez se ha establecido con éxito ya puede haber intercambio de información entre ambas partes.

Para leer información del PLC, se debe especificar el tipo de dato, la dirección de lectura y la longitud o cantidad de elementos a leer. Por ejemplo, para leer las salidas del PLC1 se utilizaría el código:

$$app.PLC(1).MB\_Con.ReadBoolean('Q0.0', 9)$$

Donde MB_Con es una de las propiedades de la clase PLC que almacena la información de conexión al autómata 1. La lectura empezaría en la dirección Q0.0 y acabaría en Q1.0, leyendo así 9 valores de tipo booleano.

El proceso para escribir información es muy similar, con la diferencia que, en lugar de especificar la longitud de lectura, se pasan los datos a escribir. Así, por ejemplo, si se quiere fijar el nuevo objetivo de producción en 300 piezas en el PLC2 se utilizaría el código:

$$app.PLC(i).MB\_Con.WriteInt16('DB16.DB\ 2", 300)$$

La dirección de memoria en este caso hace referencia al Bloque de Datos 16, en la posición 2. Este es el bloque reservado en el PLC para la comunicación con los maestros SCADA.

Una vez claros estos conceptos sólo queda especificar en qué momento del programa se usan.

En general, las operaciones de lectura se realizan de manera periódica cada medio segundo. Algunas variables se salen de esta norma y sólo se leen en determinadas circunstancias.

Las operaciones de escritura se llevan a cabo cuando se quiere ejecutar alguna orden o modificar algún parámetro en el PLC, por tanto, ocurren únicamente cuando el usuario interacciona con el sistema.

Por último, cabe destacar que el PLC1 actúa como "servidor de comunicaciones". No hay conexión directa entre el centro de control y el HMI, así que los intercambios de información entre estos elementos se hacen a través del autómata. Un ejemplo de esto es el estado de las alarmas de los distintos sistemas.

4.4 Interfaz gráfica

Como se puede observar en la ilustración 18, la interfaz está dividida en cuatro pestañas, cada una de las cuales ofrece distintas funcionalidades. En este apartado se explicará brevemente cada una de ellas.

Cabe destacar que el botón de emergencia, Start, encendido y modo son accesibles desde todas las pestañas. Esto es especialmente importante para las emergencias, ya que deben ser fácilmente accionables en cualquier momento.

4.4.1 Pestaña Principal

Desde esta pestaña se monitoriza el funcionamiento automático del proceso y se tiene acceso a gran parte de las funciones manuales. Desde aquí se puede obtener información acerca del paso actual de cada robot, así como un diagnóstico avanzado sobre los posibles errores y fallos que pueda tener el sistema. También se puede cambiar el paso de cualquiera de los manipuladores, ordenar que acudan a una posición específica, restablecer el sistema y descartar la pieza actual.

La aplicación lleva integrada un archivo con el contenido de cada fallo. Con esta información, el SCADA es capaz de transformar la matriz binaria de fallos en texto. Así se evita tener que transmitir largas cadenas de texto por los canales de comunicación.

Cabe destacar que para las operaciones de descarte y restablecimiento del sistema se ha implementado una “llave de seguridad”. Aunque ya se presupone que el operario que tenga acceso a este SCADA estará suficientemente cualificado para manipular el sistema sin riesgo, se le pide un factor extra de confirmación antes de llevar a cabo estas operaciones.

En la siguiente imagen se muestra la pestaña principal:

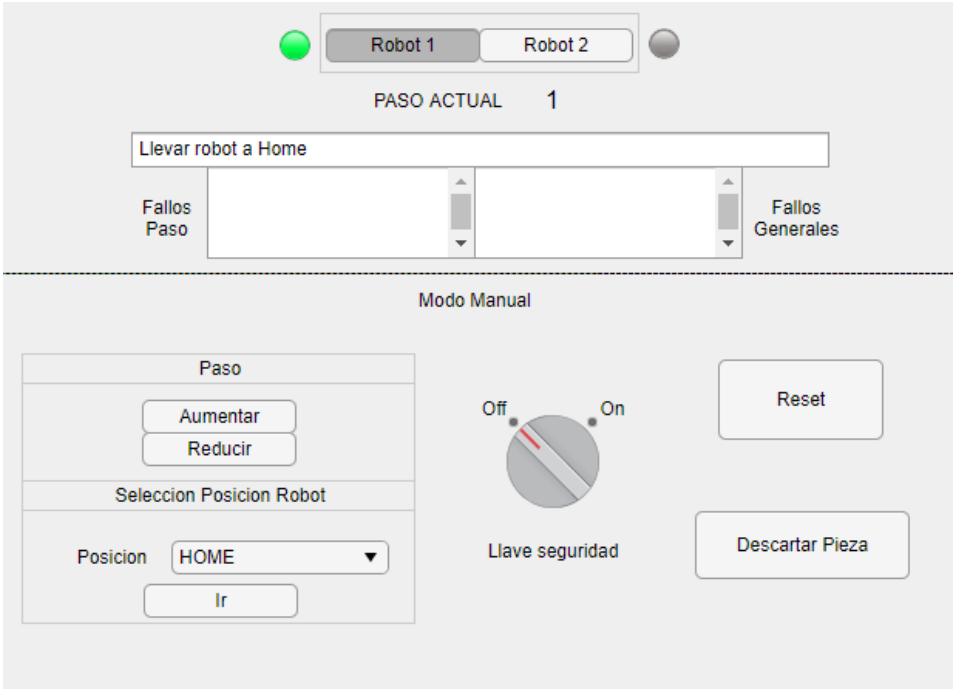


Ilustración 20 Pestaña principal. Fuente: Elaboración propia

4.4.2 Pestaña Estado del sistema: sensores, salidas y encoder

Desde esta pestaña se puede consultar rápidamente el estado de todos los sensores, las salidas de los autómatas y los encoder implicados en el proceso. También muestra información sobre qué modelo están produciendo los robots.

Toda la información mostrada en esta pestaña es leída directamente de los PLC, por tanto, permite contrastar el estado actual con el conocimiento sobre el funcionamiento del sistema. Esto puede ser muy útil a la hora de detectar y localizar problemas en el proceso, fallos en la programación o errores en la comunicación entre componentes.

A diferencia de su equivalente en el HMI, en esta pestaña los elementos no están organizados según su posición en los robots. A cambio de esta pérdida de información visual se gana en simplicidad y cantidad de información que se puede mostrar. Se le presupone al usuario de este sistema la capacidad de interpretar correctamente la información mostrada sin ayuda de elementos visuales, como podría ser un esquema del robot.

En la siguiente ilustración se muestra la pestaña de estado:

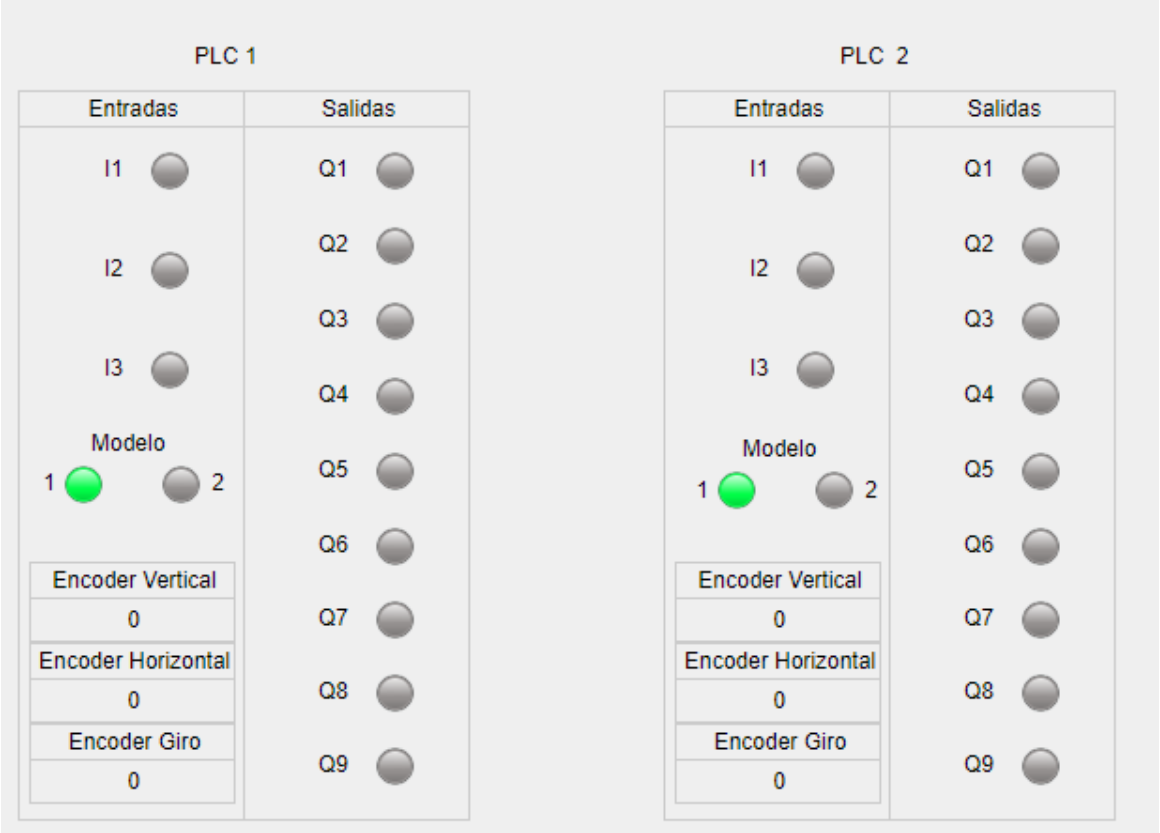


Ilustración 21 Pestaña Estado del Sistema. Fuente: Elaboración propia

4.4.3 Pestaña Control manual de movimientos

Esta pestaña permite el control directo de los movimientos del robot. Desde aquí se pueden accionar los movimientos verticales, horizontales y de giro (en los dos sentidos de cada movimiento) y se puede dar la orden de desactivar la ventosa y el compresor (soltar pieza). Además, se muestra información sobre los sensores de final de carrera de cada eje, los encoder y el accionamiento de la ventosa.

Para ejecutar los movimientos basta con seleccionar el robot deseado y apretar el botón del movimiento que se desea, o también se pueden usar las teclas W (vertical arriba), S (vertical abajo), A (horizontal avance), D (horizontal retroceso), Q (giro antihorario) y E (giro horario). Cada botón da la orden al PLC seleccionado de ejecutar el movimiento en cuestión y de parar todos los demás. Por tanto no permite desplazamiento multidireccional simultáneo.

Por motivos evidentes de seguridad y funcionamiento del sistema, esta pestaña sólo se puede usar en modo manual.

Por último, es bastante probable que esta pestaña no existiese en un sistema real, ya que seguramente desde la sala de control no se tenga una visibilidad suficiente de los robots como para controlarlos remotamente. Además, un robot manipulador real tiene un sistema de control mucho más complejo y se comanda desde una maneta especial que permite gestionar todos sus movimientos de manera más segura. No obstante, en el proyecto actual se considera que esta funcionalidad puede ser útil a la hora de calibrar cada una de las posiciones a las que deben acudir los prototipos, ya que permite un control sencillo y rápido de los movimientos.

En la siguiente ilustración se muestra la pestaña de control manual de movimientos:

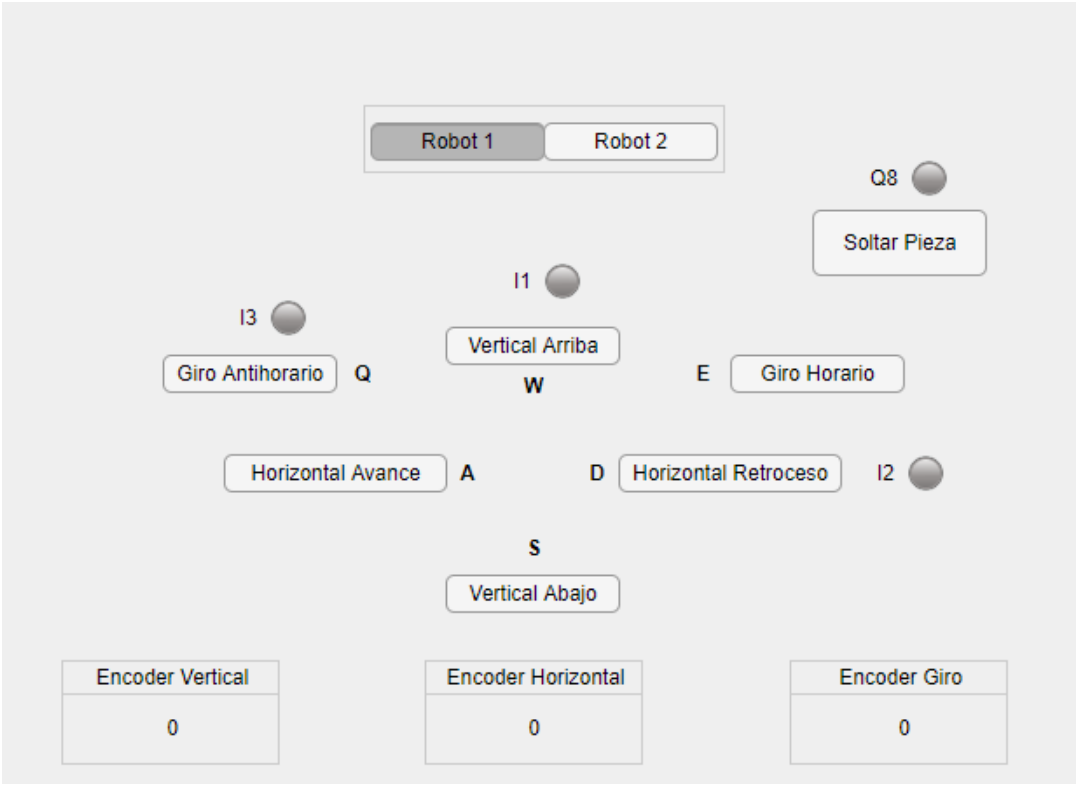


Ilustración 22 Pestaña Control manual de movimientos. Fuente: Elaboración propia

4.4.4 Pestaña Control de producción

Desde esta pestaña se tiene acceso al control de la producción y a datos sobre el proceso productivo.

La organización de la producción es una parte clave y compleja del funcionamiento de toda planta industrial. Hay muchas formas de enfocar este problema, dependiendo del tipo de producción, la demanda, la capacidad de almacenamiento, el material disponible, etc. En este caso, y teniendo en cuenta que este ámbito no entra dentro del objetivo del proyecto, se ha decidido hacer una producción simple por lotes. Sin embargo, esto se podría cambiar siguiendo, por ejemplo, una lista con las órdenes de la siguiente pieza.

De esta manera, desde la sala de control se puede elegir qué modelo se va a producir y cuál es el objetivo de producción (tamaño del lote). Una vez alcanzada esta cifra el sistema se detendrá, esperando a la siguiente orden.

Además, desde esta ventana también se puede visualizar la cantidad de piezas producidas y descartadas de cada modelo y los tiempos de ciclo de cada robot. El tiempo de ciclo mide los segundos que tarda el robot en completar todo el proceso (desde el primer paso hasta el último). También se pueden reiniciar las estadísticas si así se desea.

En la siguiente ilustración se muestra la pestaña de control de producción:

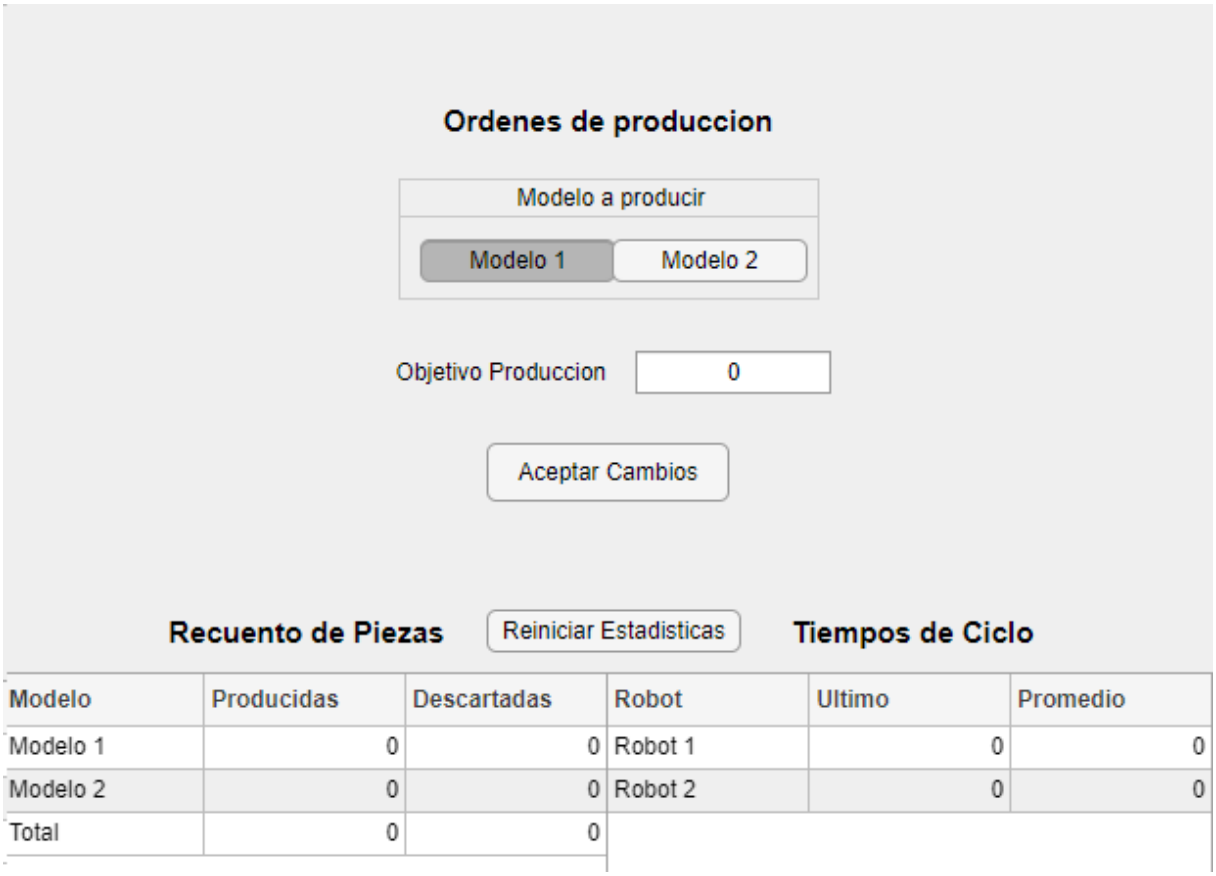


Ilustración 23 Pestaña Control de producción. Fuente: Elaboración propia

5. Programación HMI

5.1 Función

El HMI es un dispositivo táctil de control que se encuentra a pie de línea. Su objetivo principal es consultar y controlar el estado del proceso productivo. Sus principales funciones son:

- Monitorizar el sistema: aporta información sobre los sensores, el paso actual y el estado de la línea.
- Controles básicos: encender y apagar el sistema, iniciar la línea y cambiar el modo de funcionamiento (automático o manual).
- Controles manuales: permiten llevar a cabo operaciones especiales en el proceso. Cambiar el paso en el que se encuentran los robots, restablecer el sistema a condiciones iniciales, descartar piezas defectuosas y mover los manipuladores de forma manual.

Al encontrarse a pie de línea y ser accesible a personal no cualificado, las funcionalidades están más restringidas que en su equivalente en el centro de control. Está por tanto más enfocado a informar y arrancar o detener la línea, aunque también permite ciertas funciones manuales para solucionar pequeños problemas. Además, cabe destacar dos de los elementos que se encuentran en esta pantalla:

- El botón de emergencia. Como ya se ha explicado anteriormente, este botón representa a todas las setas de emergencia distribuidas a lo largo de la línea. En un sistema real, dicho botón se encontraría fuera de la pantalla, para que fuese más fácilmente accesible.
- La llave de seguridad. Este elemento cobra más importancia que en el SCADA, ya que es necesario accionarla para todas las funciones manuales, a excepción de cambiar el paso. En un sistema real, la llave sería externa y sólo la tendría personal cualificado (por ejemplo el jefe de línea o el ingeniero de planta). Con esto se asegura un cierto nivel de protección frente a accesos no autorizados.

En la siguiente imagen se muestra la pantalla principal, con sus utilidades:

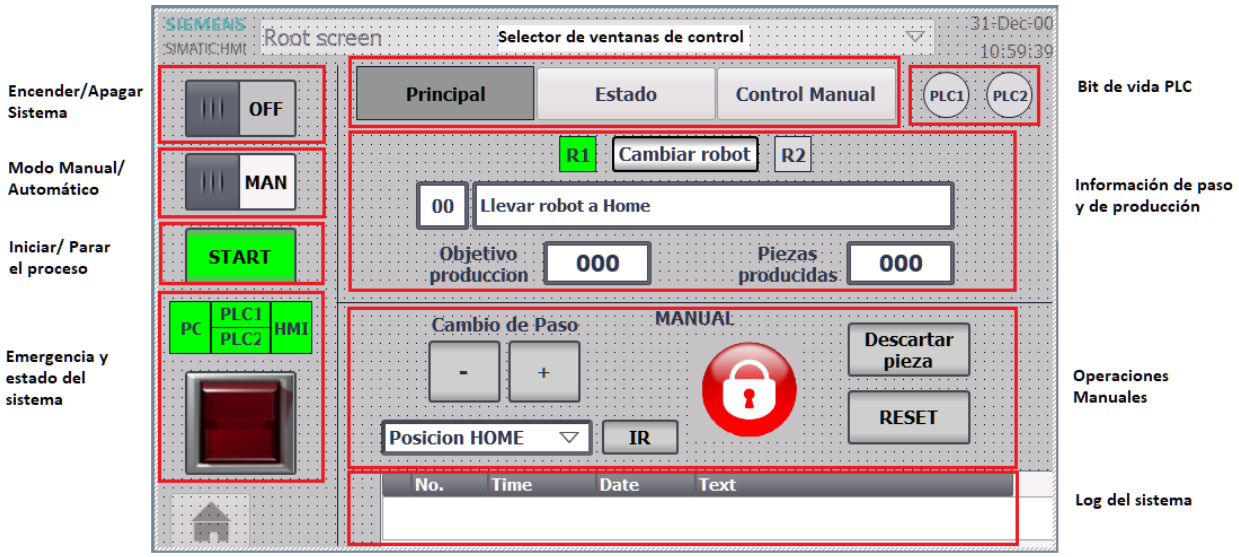


Ilustración 24 Vista general HMI. Fuente: Elaboración propia

5.2 Estructura del programa

A diferencia del *AppDesigner*, esta programación está completamente orientada a objetos gráficos. La aplicación ha de crearse únicamente a través de estos. Para compensar, cada elemento tiene opciones de configuración más amplias, que permiten desarrollar un programa de suficiente complejidad. De esta manera se sacrifica flexibilidad a cambio de sencillez. Además, el HMI tiene menos funciones y es menos complicado que su equivalente del centro de control, por tanto no hay problemas en este sentido.

Si se compara el siguiente esquema con el del SCADA (Ilustración 19) se puede observar que el funcionamiento de este es más sencillo:

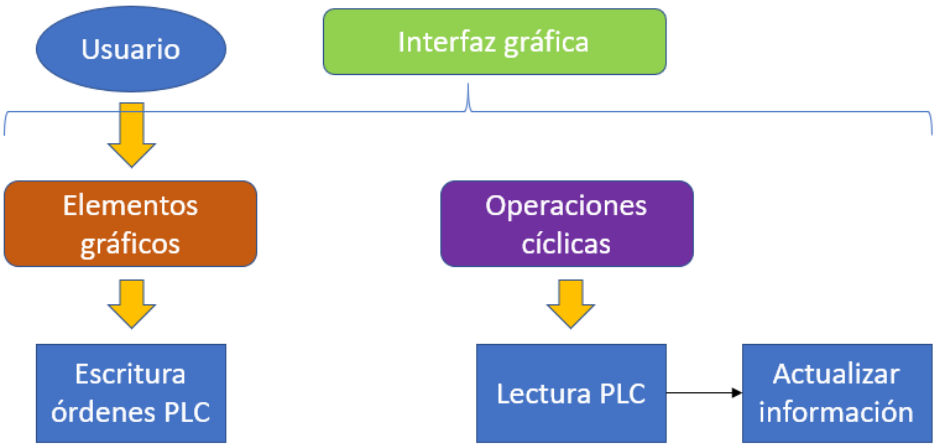


Ilustración 25 Funcionamiento HMI. Fuente: Elaboración propia

5.3 Comunicación con PLC

A diferencia de *AppDesigner*, el HMI y su software sí han sido diseñados para interacción directa con PLC. Esto implica que las comunicaciones ya cuentan con el formato específico necesario para conectar con un autómata SIEMENS, facilitando mucho el proceso.

Basta con configurar la conexión por ethernet a los dispositivos desde el propio *TIA Portal*, al igual que se conectan dos PLC, introduciendo la dirección IP de la pantalla. Una vez se ha completado este paso ya se puede especificar las variables de los PLC a las que se quiere acceder.

Al crearse la aplicación en el mismo proyecto en el que se han programado los PLC, el HMI puede acceder directamente a todas las variables disponibles en los autómatas. Esto facilita mucho el proceso, ya que ni siquiera hay que buscar cuál es la dirección de memoria.

La comunicación que se lleva a cabo es de tipo Profinet.

5.4 Interfaz gráfica

Como se puede observar en la ilustración 24, la interfaz está dividida en tres pestañas, cada una de las cuales ofrece distintas funcionalidades. En este apartado se explicará brevemente cada una de ellas.

Al igual que en la interfaz del centro de control, el botón de emergencia, start, encendido y modo son accesibles desde todas las pestañas.

5.4.1 Pestaña Principal

Muy similar a la pestaña principal del centro de control. Su función principal es poder monitorizar el funcionamiento automático de la línea y realizar algunas operaciones manuales para solucionar problemas. En este caso también se muestra el objetivo de producción y las piezas completadas, para que los operarios puedan consultarlo. Las dos principales diferencias de esta pestaña entre el HMI y el centro de control son:

- El HMI no cuenta con un diagnóstico avanzado. Se muestra el paso actual y la descripción de este, pero no los fallos. Se considera que no es necesario para la función de este dispositivo.
- Todas las operaciones manuales, excepto el cambio de paso, requieren el accionamiento de la llave de seguridad (En la interfaz actual está representada por el círculo rojo con candado).

En la siguiente ilustración se puede ver en detalle la pestaña principal:

Ilustración 26 Pestaña principal HMI. Fuente: Elaboración propia

5.4.2 Pestaña Estado del sistema: sensores, salidas y encoder

El objetivo de esta pestaña es mostrar de manera gráfica e intuitiva los sensores, las salidas del PLC y los encoder de posición de los robots. A diferencia de la otra interfaz gráfica, en el HMI estos indicadores están posicionados sobre una imagen del robot, para que sea más fácil localizarlos por personal menos cualificado o que no conoce tan bien el proceso. Esto puede ser especialmente útil para tareas de mantenimiento, como por ejemplo cambiar o limpiar un sensor. En la pestaña, los círculos representan los sensores y los cuadrados las salidas del PLC. Sólo se muestra el estado del robot seleccionado.

En la siguiente ilustración se muestra la pestaña de estado del sistema:

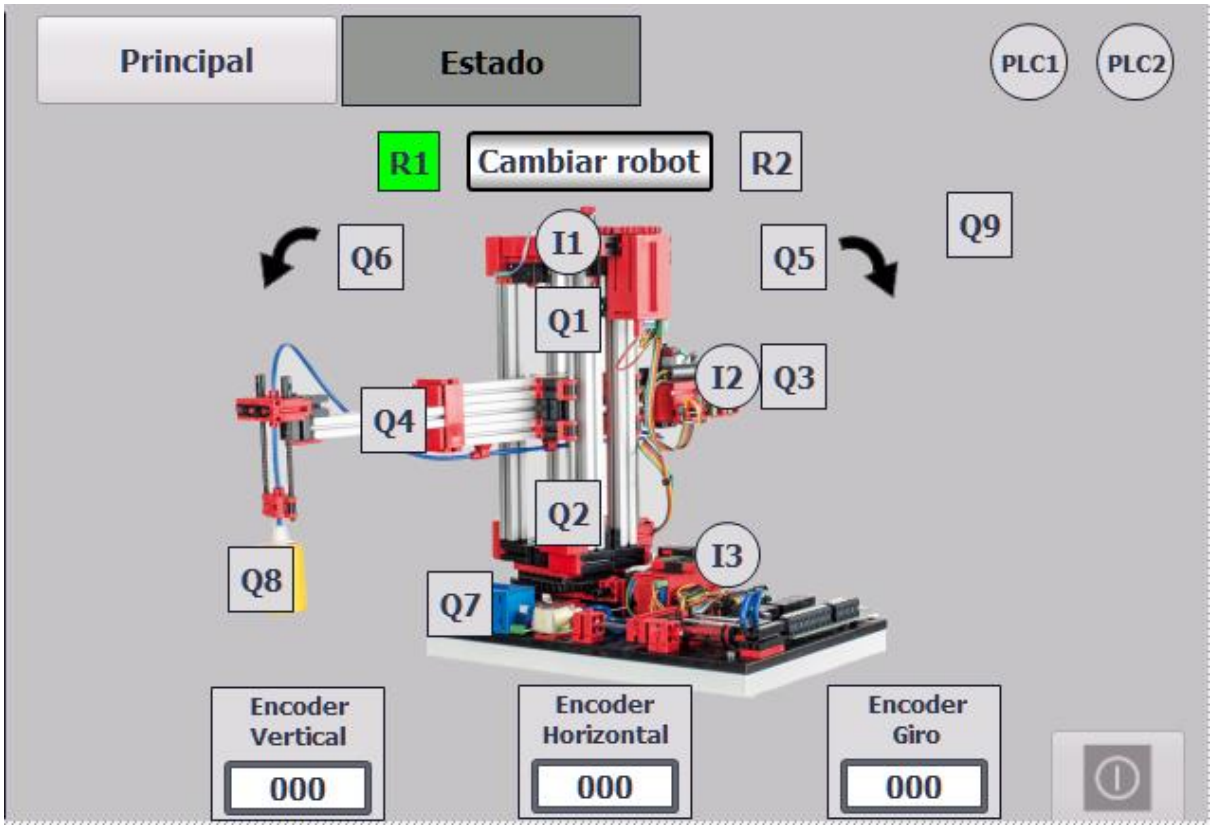


Ilustración 27 Pestaña estado sistema HMI. Fuente: Elaboración propia

5.4.3 Pestaña Control manual de movimientos

Esta pestaña permite controlar de manera manual y directa los movimientos del robot. Aunque tiene más sentido aquí que en la otra interfaz gráfica, porque el robot sí que está a la vista, lo más probable es que en un sistema real tampoco se implementase, por los motivos ya explicados.

A esta pestaña sólo se puede acceder en modo manual y con la llave de seguridad. El funcionamiento es el mismo que el de la interfaz gráfica del centro de control, a excepción del control con las teclas.

En la siguiente ilustración se muestra la pestaña de control manual:

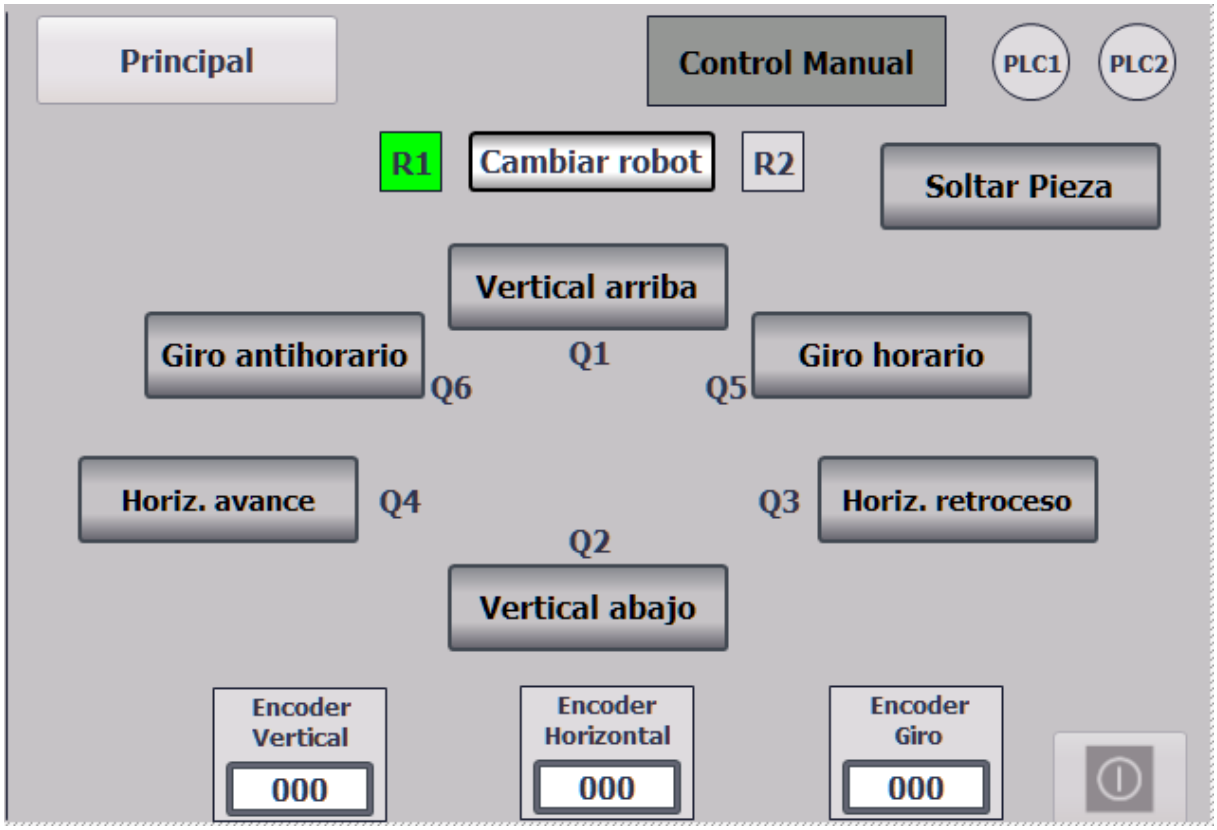


Ilustración 28 Pestaña control manual HMI. Fuente: Elaboración propia

6. Adaptar el sistema a una línea real

Este proyecto ha sido diseñado para una pequeña maqueta con dos prototipos de robot, dos PLC y un HMI. En este apartado se van a exponer algunos de los cambios que se podrían aplicar para adaptar el sistema a una línea productiva real. Además, se propondrá un diseño general de la línea, con nuevos elementos y medidas de seguridad. El objetivo es mostrar la utilidad del trabajo realizado y poder llevar a cabo un presupuesto más completo y similar al de un proyecto real.

El diseño completo de una línea real es un proceso complejo, elaborado y minucioso, que depende de una gran cantidad de factores, como por ejemplo, el peso de las piezas, el proceso a realizar, el material, la geometría o las especificaciones de calidad. En resumidas cuentas, es un proceso que podría ocupar un TFM entero. Por tanto, y teniendo en cuenta el carácter orientativo de este apartado, se ha realizado el diseño siguiendo unas especificaciones de carácter general. El resultado no podría ser ejecutado directamente, pues no está completo, pero sirve como primera aproximación para dar una idea de cómo podría ser el proceso y para aportar una idea de la magnitud en lo referente al apartado económico.

6.1 Cambios en el programa

Aunque se ha tratado diseñar el programa de una manera similar a lo que podría ser la programación de una línea real, es evidente que hay ciertos aspectos que se tendrían que cambiar para adaptarlo.

6.1.1 Un único PLC

En primer lugar, en un entorno industrial una línea de este tamaño estaría seguramente controlada por un único PLC. Los autómatas utilizados en la industria permiten conectar muchas más entradas y salidas y tienen una mayor potencia de procesado. Teniendo esto en cuenta, la estructura del programa se vería ligeramente alterada. En lugar de usar los dos PLC, el programa de ambos manipuladores se integraría en uno solo. Como estructura de programación se podría, por ejemplo, dividir el programa en secuencias. Cada secuencia incluiría su propia función principal (Como la función Main, en este caso) y todas las funciones necesarias para su correcto funcionamiento. Cada robot y cada estación tendrían asignadas una secuencia distinta. En la siguiente figura se muestra una comparativa entre las dos estructuras:

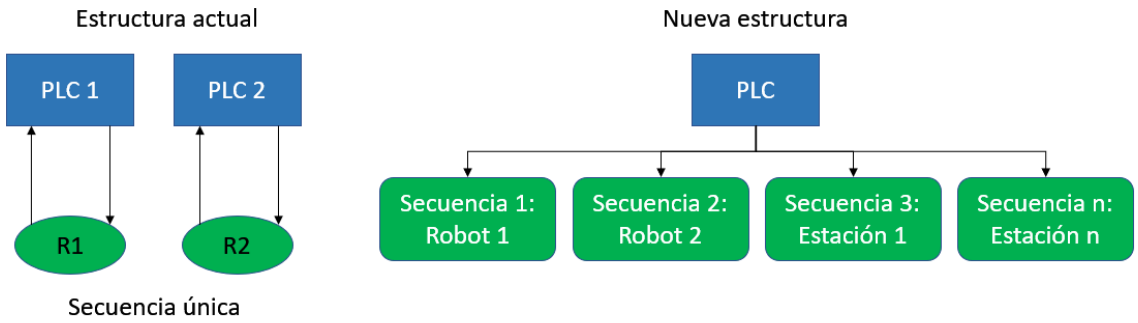


Ilustración 29 Comparación entre estructuras. Fuente: Elaboración propia

Este cambio también implicaría modificaciones en la configuración de las comunicaciones con el SCADA y el HMI y eliminaría la necesidad de comunicación PLC-PLC.

6.1.2 Sensores de pieza

Por cuestiones de seguridad y eficiencia del proceso sería recomendable incluir sensores de pieza, tanto en las estaciones como en los manipuladores. De esta manera cada uno de los robots sería capaz de comprobar que ha cogido la pieza correctamente, y el PLC podría saber dónde se encuentran las piezas en cada momento. Esto permitiría añadir protocolos de seguridad para evitar que los robots intentasen depositar piezas en estaciones ya ocupadas, o que tratasen de coger de una estación vacía.

Para implementar estos sensores en el programa bastaría con añadirlos como condiciones de paso. Por ejemplo, en el paso 6 del R1, que es el de levantar la pieza desde la entrada a línea, se podría comprobar que el manipulador detecta la pieza y que además no hay pieza en la estación inicial. En caso de que no fuese así, el robot volvería al paso 3 para volver a iniciar el proceso de cogida. En la siguiente ilustración se muestra la diferencia entre el programa actual (izquierda) y la modificación (derecha). *PiezaOK* hace referencia al sensor del robot y *Entrada_linea_Pieza* a la estación inicial.

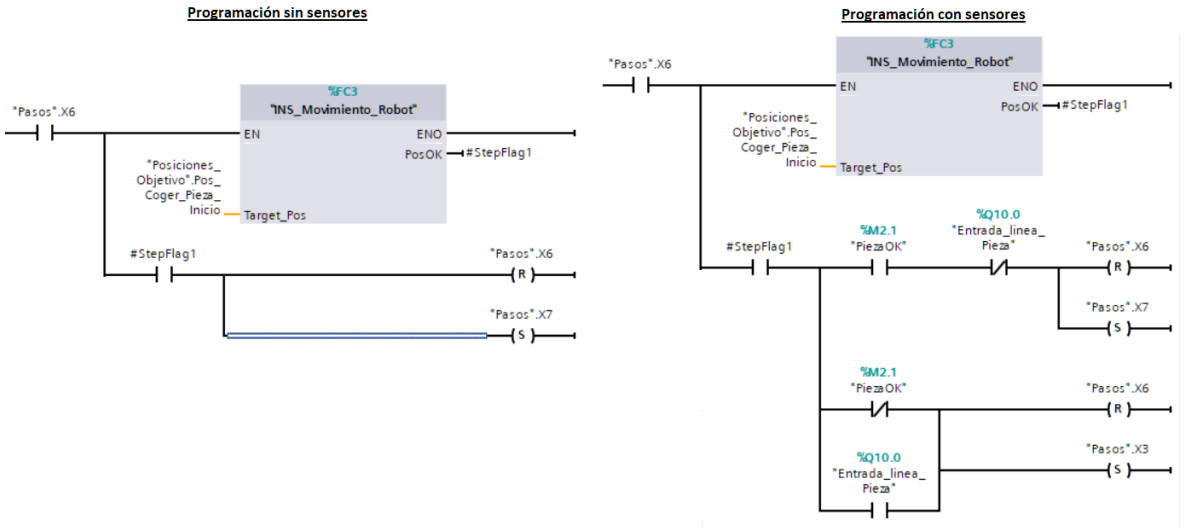


Ilustración 30 Cambios del programa al añadir sensores. Fuente: Elaboración propia

Con este procedimiento también se podría eliminar la comunicación entre robots en lo referente a la pieza en la estación intermedia R1R2, ya que el sensor detectaría la pieza sin necesidad de que R1 informase a R2 y viceversa.

6.1.3 Ciberseguridad PLC

Aunque las redes industriales suelen estar aisladas y protegidas del exterior, es recomendable añadir al menos un nivel básico de seguridad informática en el PLC para evitar un acceso no autorizado al sistema de control, ya que se trata de un sistema potencialmente peligroso. Los PLC Siemens incluyen la opción de aplicar ciertas medidas de cifrado y protección, que podrían ser suficientes para el proyecto actual.

6.1.4 Control de los robots

El control de los robots industriales es muy complejo. Se trata de máquinas con mucha fuerza y que permiten gran cantidad de movimientos distintos de alta precisión a velocidades elevadas. Por este motivo, el control se traslada de los PLC a la propia CPU del robot (ilustración 31). Desde aquí se programan los movimientos, las velocidades y se controlan los motores, los frenos, la refrigeración y el resto de los sistemas.

Por tanto, en un sistema real, el PLC se limita a especificar las órdenes que ha de ejecutar el robot en cada momento. Para ello se comunica con la CPU y le indica, por ejemplo, que debe moverse a la posición x, activar el mecanismo de cogida o accionar el freno de emergencia.

En el caso de este programa bastaría con modificar las funciones de Instrucción con bloques de comunicación que se encargasen de enviar dicha orden a los manipuladores.

Ilustración 31 CPU de un robot KUKA. Fuente: Página web KUKA Robots

6.1.5 Mecanismos de seguridad

Las medidas de seguridad son una parte crucial de cualquier proceso industrial. Un fallo de seguridad puede desembocar en pérdidas económicas e incluso en daños a los trabajadores de la empresa. Por ese motivo estos mecanismos deben de ser completos y confiables, cubriendo, en la medida de lo posible, todas las posibles situaciones de emergencia que se puedan producir.

Debido a las limitaciones físicas, los mecanismos de seguridad del proyecto se han limitado a dos setas de emergencia. Sin embargo, en una línea real se añadirían muchos más, como por ejemplo radares de proximidad, barreras láser y un perímetro de seguridad. Más adelante se propondrá un diseño de estos mecanismos.

Ya que se trata de un sistema crítico, estos elementos suelen estar programados en un PLC de seguridad, distinto del PLC normal, que está unido a los elementos de seguridad mediante una red de

comunicaciones especial y más robusta que la principal. Este PLC está además protegido por una firma de seguridad, para evitar modificaciones no autorizadas.

6.2 Elementos de la línea

Además de los cambios en el programa y la sustitución de los elementos actualmente presentes en la línea, para adaptar el sistema a un proceso productivo real hay que añadir nuevos componentes para mejorar la seguridad y la efectividad del sistema. En este apartado se propone un diseño general con los elementos más importantes. Los productos a elegir se clasifican en los siguientes grupos:

- PLC.
- Robots manipuladores.
- Sensores.
- Mecanismos de seguridad.

Para poder continuar con el diseño se van a plantear una serie de hipótesis, que se utilizarán como especificaciones para elegir los elementos:

- Las piezas son de acero con algunos apliques de plástico, y tienen una dimensión aproximada de 1 x 0.5 m y un peso máximo de 100 kg.
- Cada una de las estaciones, excepto las de entrada y salida, contarán con 3 sensores de pieza y 2 para detectar los apliques.
- Las estaciones de entrada y salida contarán con sensores de barrera láser.
- Las manos de los brazos robóticos contarán con 3 sensores de pieza.
- Las estaciones R1M1 y R2M2 son estaciones de operario, a las cuales el trabajador accederá para introducir alguna pieza de manera manual.
- Los procesos llevados a cabo en las estaciones R1M1, R2M2 y R1R2 no forman parte del diseño expuesto y tendrían que ser añadidos en un futuro diseño.

6.2.1 PLC

Como ya se ha explicado anteriormente, para una línea industrial de este tamaño se propone el uso de un único PLC. De esta manera se reduce la complejidad del sistema, se evita la necesidad de comunicaciones entre autómatas y se centraliza más el control del proceso. Además, para poder conectar todos los nuevos elementos hace falta un PLC con una mayor potencia de procesamiento y con más entradas y salidas.

En la industria actual, la mayoría de los elementos están conectados por medio de ethernet industrial. Esto permite la comunicación entre componentes a una velocidad mayor y con un estándar de cableado establecido a nivel mundial. Por este motivo, los PLC industriales utilizan módulos de entrada y salida con conexión ethernet Profinet.

Para poder utilizar el mismo programa con la menor cantidad posible de modificaciones lo ideal es seguir usando el *TIA Portal* como plataforma de software. Esto permitirá reutilizar la mayor parte del código. Por este motivo se va a elegir un PLC de la marca SIEMENS. Los PLC de esta marca permiten

además la programación de los elementos de seguridad en el mismo autómata, de manera aislada, sin necesidad de tener uno especial.

El modelo elegido es el "SIMATIC S7-1511-1PN". El fabricante ofrece un pack con todos los elementos necesarios para el funcionamiento por un precio de 1258,75 €. Además de este pack, una vez diseñado todo el sistema y teniendo claro las entradas y salidas necesarias habría que considerar si hace falta añadir módulos adicionales de entradas y salidas.

También se propone adquirir, con el fin de modernizar el sistema y adaptarlo a las nuevas tecnologías de la información, el módulo "Pasarela IoT". Este módulo permite crear una pasarela entre la nube, los sistemas informáticos y los sistemas de producción.

El HMI utilizado será el especificado al principio del proyecto. El modelo KTP700 Basic PN, con un precio de 680.02 €

Ilustración 32 PLC 1511. Fuente: Página web Siemens

6.2.2 Robot

La elección de los robots depende en gran medida del tipo de piezas que se vayan a utilizar en el proceso. Las dimensiones, el peso e incluso el material son factores determinantes a la hora de determinar cuál es el manipulador ideal. En este caso, y de manera arbitraria, se va a suponer que las piezas producidas serán de chapa de acero y tendrán un peso máximo de 100 kg.

El robot elegido es el IRB 6620 de ABB. Se ha elegido este modelo por su peso de carga (150kg) y el alcance de su brazo robótico (2,2 m). Tiene un precio de 37.500 €.

Este robot se controla y programa desde su propia CPU, como ya se ha explicado anteriormente.

Ilustración 33 Brazo robótico ABB IRB 6620. Fuente: Página web ABB

Además del propio brazo robótico será necesario diseñar la mano manipuladora, que es la encargada de coger las piezas. El diseño de estos componentes se hace a medida y depende completamente de la geometría de la pieza, así que se deja como parte pendiente.

6.2.3 Sensores

Los sensores son una parte fundamental de cualquier proceso automático, pues dan la habilidad al autómata de conocer el estado del sistema, permitiendo que éste ejecute las órdenes específicas para cada caso. Además, también son muy útiles para poder mejorar la seguridad, ya que permiten hacer comprobaciones sobre el proceso. Por ejemplo, se puede detectar si una pieza ha sido cogida correctamente o si el robot deja de detectar la pieza que transporta antes de tiempo.

Con las especificaciones expuestas anteriormente se seleccionarán los elementos que mejor se adapten a las necesidades del sistema. En este caso se necesitan sensores para detectar la presencia de piezas metálicas con apliques de plástico. Este detalle es especialmente importante a la hora de seleccionar los sensores.

Estaciones de trabajo

En primer lugar, para las estaciones seleccionaremos 3 sensores para confirmar que la pieza está en su sitio, y dos sensores extra para comprobar que se han añadido correctamente los apliques de plástico.

Dado que las piezas son metálicas, se debería evitar sensores de tipo fotocélula para detectar la pieza, ya que la superficie brillante puede afectar a la lectura. Además, este tipo de material suele ir recubierto de una película protectora de aceite, para protegerlo de la corrosión, por tanto es recomendable evitar sensores de contacto, que se pueden ensuciar y estropearse a la larga. Por estas razones se han elegido sensores de proximidad inductivos. Las piezas de plástico no tienen estas restricciones, por tanto se seleccionan sensores fotoeléctricos.

Los sensores elegidos son:

- "XS112B3P A L2" de Schneider, sensor inductivo con alcance de detección de 4mm y con un precio unitario de 29,78 €.
- "XUB5BP ANL2" de Schneider, sensor fotoeléctrico de proximidad con alcance de detección de 0,6 m y precio unitario de 49,36 €.

Estaciones de entrada/salida

Para las estaciones de entrada y salida, que conectan con cintas transportadoras, la posición de la pieza es menos importante. Por tanto basta con un sensor tipo barrera láser para detectar la presencia de la pieza. Para hacer el sistema más robusto se pondrán dos barreras próximas, de manera que la pieza tenga que activar las dos una vez está en la posición correcta. Para esto la mejor opción vuelve a ser sensores fotoeléctricos, esta vez con configuración réflex. En esta configuración, el sensor apunta su rayo hacia una superficie reflectante y detecta si hay un objeto en medio o no.

El sensor elegido es el "XUB1BP ANL2" de Schneider, sensor fotoeléctrico de tipo réflex con un alcance de detección de 4m y un precio unitario de 43,82 €.

Robots

Para las manos de los brazos manipuladores será necesario comprobar que se detecta pieza y que además está en la posición correcta. Para esto hay que tener en cuenta las mismas restricciones que en las estaciones. Los sensores elegidos son los mismos, del tipo inductivo. ("XS112B3P A L2").

6.2.4 Elementos de seguridad

A lo largo de este documento se ha recalcado repetidamente la importancia de las medidas de seguridad en una línea productiva. En este apartado se recogen los que se han considerado más importantes, haciendo una selección de productos. No obstante, antes de exponer los resultados se ha de explicar cómo se ha organizado la seguridad. En la siguiente ilustración se puede observar una representación del diseño de la línea:

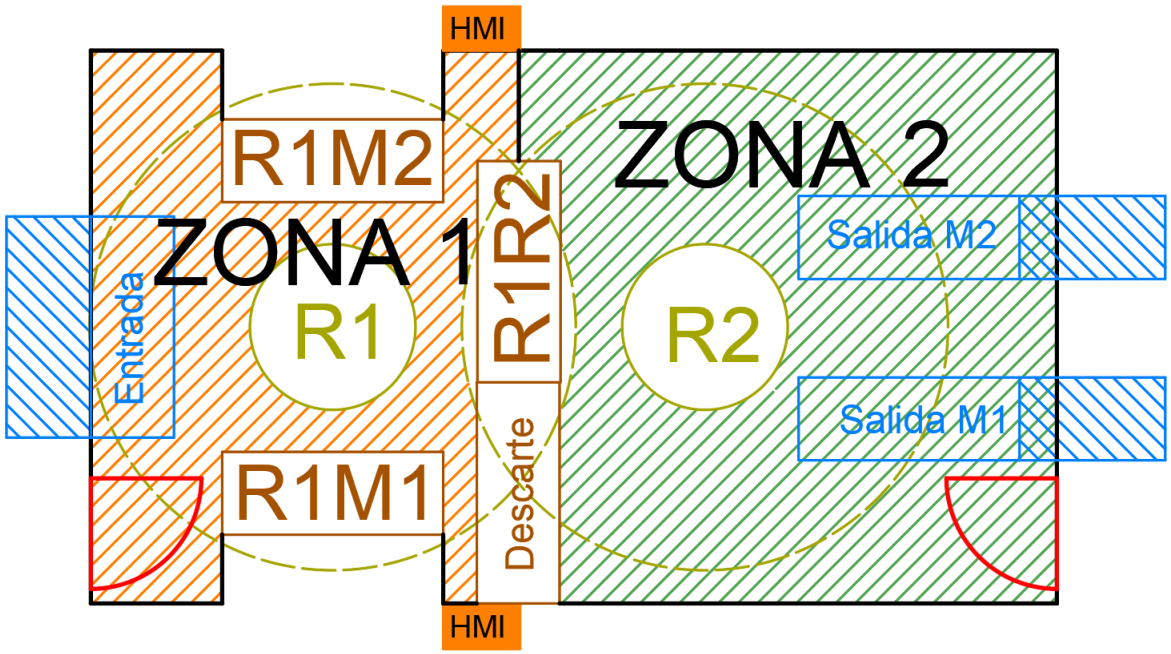


Ilustración 34 Esquema línea con zonas. Fuente: Elaboración propia

Como se puede observar, la línea está dividida en dos zonas de seguridad. Una zona de seguridad es una unidad de "territorio" en la cual las emergencias actúan. Por ejemplo, si se abre una puerta en la zona 1, toda esa parte de la línea se para de manera inmediata, pero la zona 2 puede seguir trabajando. Esto se aplica con todos los elementos de seguridad, a excepción de las setas de emergencia, que detienen todo el proceso.

Entre ambas zonas, unidas a través de la estación R1R2, debe existir un mecanismo de emergencia que actúe como separación. Una opción es que haya una cortina láser que detecte los traspasos no autorizados. De esta manera si un operario entra en la zona 1 e intenta pasar a la zona 2, la emergencia saltará también en esta.

Una vez aclarado, se puede recopilar una lista de los elementos necesarios:

- Setas de emergencia distribuidas por la línea.
- En las estaciones de operario se necesitará una botonera de comandos y un escáner láser de seguridad.
- Una puerta de acceso para cada zona.
- Barrera láser para la separación entre zonas.

Setas de emergencia

Han de ser accesibles desde cualquier punto de la línea, fácilmente accionables y visibles. Al apretarla todo el sistema entra en emergencia y se detiene el movimiento de la maquinaria.

El modelo elegido para las setas es "XALK178" de Schneider, con un precio unitario de 62,94 €. Se adquirirán 7 unidades, dispuestas como se muestra en la siguiente ilustración:

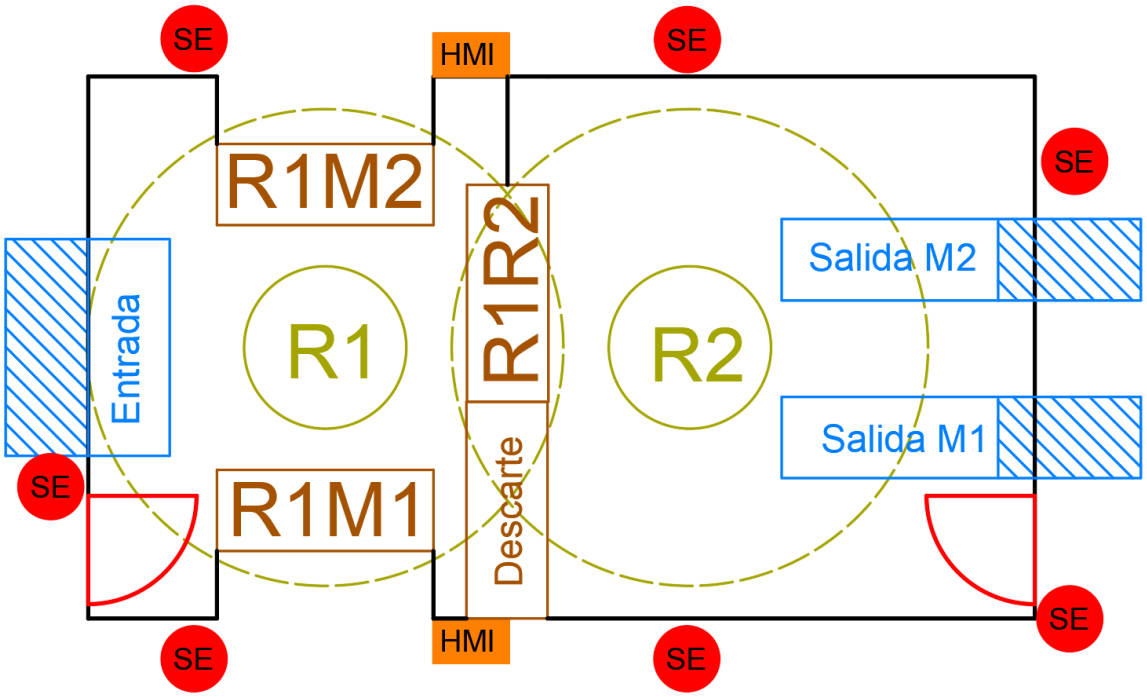


Ilustración 35 Esquema línea con Setas. Fuente: Elaboración propia

Botoneras

Accesibles desde las estaciones de operario. Sirven para restablecer los radares de presencia y para comandos específicos, como por ejemplo activar el Start. Su posición se muestra en la siguiente ilustración, representado como un cuadrado gris con las letras BOT.

El modelo elegido para las botoneras es "XALD324" de Schneider, con un precio unitario de 84,38 €. Se requieren 2 unidades.

Escáner láser de seguridad

En las zonas que son utilizadas por los robots y los operarios debe haber un procedimiento extra de seguridad para asegurar que el acceso a la misma no se produce de manera simultánea. De esta manera, si el operario está dentro de la zona, el robot esperará, y si es el robot el que está dentro y el operario entra, saltará la emergencia. Con este fin se ha seleccionado un escáner láser que es capaz de detectar la presencia de objetos y personas en una zona.

El modelo seleccionado es "S30A-4011BA" de SICK, con un precio unitario de 4.200 €. Se requieren 2 unidades, dispuestas como se muestra en la siguiente ilustración (zona amarilla frente a las estaciones de operario):

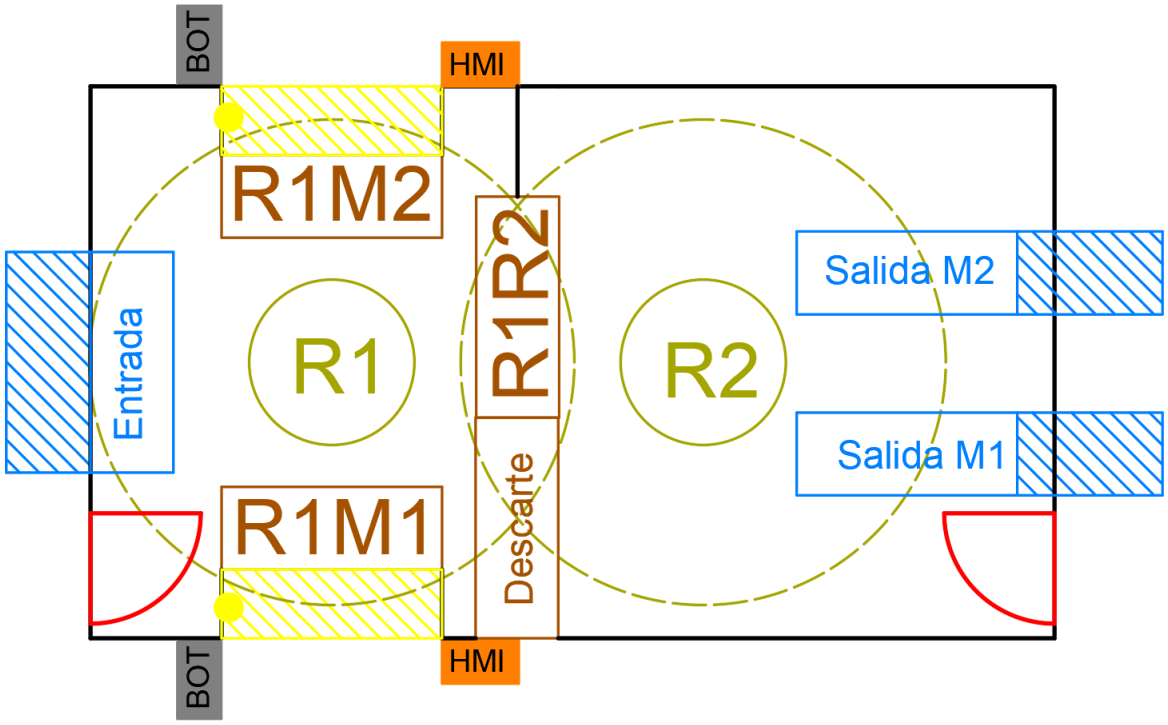


Ilustración 36 Esquema línea con escáner láser. Fuente: Elaboración propia

Barrera láser

Elemento de separación entre zonas. Hará saltar la emergencia si se detecta un traspaso no autorizado. El modelo elegido es el "C4000 Advanced Atex II 3G" de SICK, con un precio unitario de 12,97 €. Se requieren 2 unidades para formar la cortina.

Puerta de acceso

Las líneas de producción que utilizan robots móviles están protegidas por un perímetro de seguridad. Para acceder a este perímetro se ha de entrar a través de una puerta especial. Esta puerta está bloqueada durante el funcionamiento automático de la línea y sólo se desbloquea al poner el modo manual o activar la emergencia. Una vez abierta, la zona a la que se ha accedido queda en emergencia, y no puede volverse a activar hasta que la puerta ha sido cerrada y el sistema reiniciado.

El modelo elegido es el "MGB-L1H-APA-R-110458" de Euchner, con un precio unitario de 954 €.

Cabe destacar que el producto como tal es la cerradura. La propia puerta y el perímetro se encargan bajo demanda, pero no tienen un coste elevado.

El esquema completo de la línea, con todos los elementos, se muestra en la siguiente ilustración:

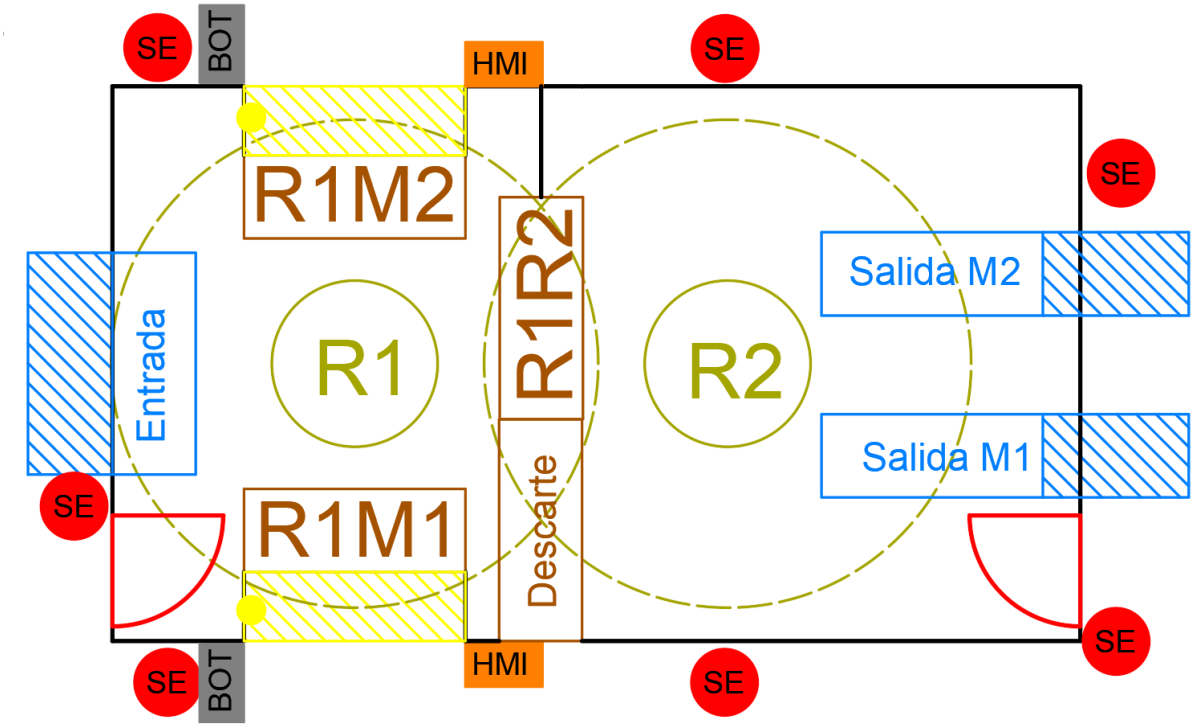


Ilustración 37 Esquema línea completa. Fuente: Elaboración propia

7. Pliego de condiciones

El objetivo del presente pliego de condiciones es el de recoger y exponer las condiciones generales y técnicas que afectan de manera directa a este proyecto. Con el fin de crear un documento más realista, este pliego hará referencia al diseño preliminar expuesto en el apartado anterior. Por tanto, y teniendo en cuenta el alcance de este trabajo, se pueden diferenciar dos partes:

- Pliego de condiciones generales: en él se incluye una descripción general del proyecto y el marco legislativo en el que se encuentra.
- Pliego de especificaciones técnicas: recoge las características de los equipos necesarios para el desarrollo del trabajo y los seleccionados en el diseño preliminar.

Finalmente se expondrán también una serie de directrices e información para el uso de las aplicaciones de control.

7.1 Pliego de condiciones generales

El objetivo de este proyecto es el desarrollo y programación de un sistema de control para una línea industrial compuesta por dos robots manipuladores. Dicho sistema consta de tres partes: el programa que controla el robot, un HMI que proporciona información sobre la línea y permite cierto control sobre ella, y un SCADA que opera desde el centro de control de la planta industrial, con control y diagnóstico avanzado sobre todos los aspectos del proceso. Además, se propone un diseño preliminar de los elementos más importantes de la línea.

Las leyes más importantes a tener en cuenta en caso de continuar el diseño y montaje de la línea productiva son[7] [8]:

- Real Decreto 1215/1997, de 18 de julio, por e que se establecen las disposiciones mínimas de seguridad y salud para la utilización por los trabajadores de los equipos de trabajo.
- Real decreto 842/2002, de 2 de agosto, por el que se aprueba el Reglamento electrotécnico para baja tensión.
- Ley 31/1995, de 8 de noviembre, de prevención de Riesgos Laborales.
- Real Decreto 1580/2006, de 22 de diciembre, por el que se regula la compatibilidad electromagnética de los equipos eléctricos y electrónicos.
- Norma IEC 1131-3 que establece la estandarización en la programación a nivel industrial.
- UNE-EN 62061:2005: Seguridad de las máquinas. Seguridad funcional de sistemas de mando eléctricos, electrónicos y electrónicos programables relativos a la seguridad.

7.2 Pliego de especificaciones técnicas

En este apartado se recogen las especificaciones técnicas del material necesario para llevar a cabo el presente TFM, así como las de los equipos seleccionados durante el diseño preliminar de la línea.

7.2.1 Ordenador personal

Durante el desarrollo del proyecto se requiere el uso simultáneo de programas bastante demandantes a nivel de hardware y de las simulaciones de los distintos sistemas. Por tanto, es recomendable el uso

de un PC de potencia y capacidad de procesamiento suficiente. Las características del ordenador utilizado son:

- Procesador Intel i7-7700HQ 2.8GHz, 4 Núcleos.
- 12 GB de memoria RAM.
- Disco duro SSD de 128 GB.

Y el software utilizado:

- Windows 10.
- TIA Portal v13 con la extensión PLCSim.
- Matlab R2020a con la extensión AppDesigner.
- NetToPLCSim.

7.2.2 PLC

El PLC seleccionado en el diseño preliminar es el modelo SIMATIC S7-1511-PN, con las siguientes especificaciones técnicas:

- Tiempo de procesamiento para operaciones binarias: 60 ns.
- Memoria de trabajo: 150 KB de programa y 1 MB para datos.
- Tensión de alimentación: 24 V.
- Corriente consumida: 0,7 A
- Potencia consumida: 5,5 W-
- Interfaz Profinet.
- Entradas/salidas analógicas integradas: 5 entradas y 2 salidas.
- Entradas/salidas digitales integradas: 16 entradas y 16 salidas.

7.2.3 HMI

Las características del HMI KTP700 Basic PN son:

- Tamaño de la pantalla: 7 pulgadas (154,1 x 85,9 mm).
- Resolución: 800 x 480 pixel.
- Número de colores: 65536.
- Tensión de alimentación: 24 V.
- Corriente consumida: 230 mA.
- Potencia consumida: 5,5 W.
- Memoria para datos: 10MB.

7.2.4 Brazo robótico

El robot elegido para el proyecto es el modelo IRB 6620 de ABB. Sus características más importantes son:

- Alcance: 2,2 m.
- Carga máxima: 150 kg.
- Número de ejes: 6
- Tensión de alimentación: 200 – 600 V, 50/60 Hz.

- Dimensiones de la base: 1007 x 760 mm.
- Peso: 900 Kg.

7.2.5 Sensores

Característica	XS112B3P A L2	XUB5BP ANL2	XUB1BP ANL2
Alcance de detección	4 mm	0,6 m	4 m
Tensión de alimentación (V)	[10, 36]		
Frecuencia de conmutación (Hz)	2500	500	500
Rango de temperaturas (ºC)	[-25, 70]		
Grado de protección	IP68	-	-

Tabla 7 Especificaciones técnicas de los sensores

7.2.6 Escáner láser

Modelo S30A-4011BA:

- Rango del campo de protección: 4 m.
- Rango del campo de advertencia: 49 m.
- Ángulo de escaneo: 190º.
- Resolución: 70 mm.
- Tiempo de respuesta: 60 ms.
- Tensión de alimentación: 24 V.
- Corriente consumida: 0.8 A.

7.2.7 Barrera láser

Modelo C4000 Advanced Atex II 3G:

- Resolución 30 mm.
- Altura: 1500 mm.
- Rango de escaneo: 21 m.
- Tiempo de respuesta: 24 ms.
- Tensión de alimentación: 24 V.

7.2.8 Puerta de seguridad

Modelo MGB-L1H-APA-R-110458:

- Peso: 0,75 Kg.
- Grado de protección: IP 65.
- Fuerza de bloqueo: 2000 N
- Tensión de alimentación: 24 V.
- Corriente consumida: 200 mA.
- Tiempo de respuesta 50 ms.

7.3 Manual de las interfaces HMI/SCADA

Las directrices a tener en cuenta a la hora del uso de las aplicaciones de control son las siguientes:

- El primer paso para realizar cualquier acción en el sistema es encenderlo (Botón ON).
- Para que el proceso funcione de manera automática, el sistema debe estar en modo AUTO y se debe apretar START. El sistema se puede detener con el botón STOP, o cambiando el sistema a modo manual.
- Para realizar cualquier operación manual, como el cambio de paso, descarte de piezas, reset o control manual de la posición, se debe activar el modo MANUAL. Si una vez activado este modo, la operación deseada sigue sin estar disponible, se deberá accionar la llave de seguridad.
- Apretar el botón de emergencia detendrá todo el proceso. Una vez esto ocurra, la interfaz mostrará qué sistema ha hecho saltar la alarma. Cuando se haya quitado la emergencia, para desbloquear el autómata, se debe hacer un Reset.
- Una vez alcanzado el objetivo de producción, el proceso se detendrá hasta que se vuelva a dar una nueva orden de producción.
- La pestaña de Estado muestra un resumen del estado de los sensores y actuadores del sistema.
- Para comprobar que la comunicación con los PLC está activa, se pueden consultar los bits de vida de cada autómata. Éstos deben de estar parpadeando. Una luz fija significa un fallo de comunicación.
- Para más información a cerca de las interfaces, consultar las ilustraciones 18 (Para el SCADA) y 24 (para el HMI)
- **Sólo para la aplicación SCADA:** Desde la pestaña de producción se puede seleccionar el modelo y la cantidad de piezas a producir. Una vez seleccionadas, se ha de apretar el botón “Aplicar cambios”. Desde esta pestaña se pueden consultar también las estadísticas de piezas completadas y descartadas y de los tiempos de ciclo.

8. Presupuesto

A la hora de redactar un presupuesto sobre este proyecto se ha decidido utilizar los datos aportados en el apartado anterior (Adaptar el sistema a una línea real). Se considera que de esta manera el presupuesto gana más significado, y se asimilará más al de un proyecto profesional.

Se distinguen tres apartados:

- Software: Recoge los costes de los programas utilizados.
- Hardware: Recoge los costes de la maquinaria utilizada.
- Recursos humanos.

8.1 Software

El coste de las licencias de los programas utilizados es el siguiente:

Modelo	Descripción	Precio/unidad	Unidades	Precio total
PLCSim Advanced V3.0	Simulador PLC	2.000,00 €	1	2.000,00 €
SIMATIC Step 7 Prof. v16	Licencia TIA Portal	1.512,00 €	1	1.512,00 €
Matlab	Licencia Matlab	2.000,00 €	1	2.000,00 €
			Total	5.512,00 €

Tabla 8 Presupuesto Software. Fuente: Elaboración propia

8.2 Hardware

El coste de la maquinaria y los elementos previamente explicados necesarios para la línea puede dividirse en:

SENSORES				
Modelo	Descripción	Precio/unidad	Unidades	Precio total
XS112B3P A L2	Inductivo	29,78 €	15	446,70 €
XUB5BP ANL2	Fotoeléctrico de proximidad	49,36 €	6	296,16 €
XUB1BP ANL2	Fotoeléctrico reflex	43,82 €	2	87,64 €
			Total	830,50 €

Tabla 9 Presupuesto Sensores. Fuente: Elaboración propia

ELEMENTOS DE SEGURIDAD				
Modelo	**Descripción**	**Precio/unidad**	**Unidades**	**Precio total**
XALK178	Seta de emergencia	62,94 €	7	440,58 €
XALD324	Botonera	84,38 €	2	168,76 €
S30A-4011BA	Escáner láser	4.200,00 €	2	8.400,00 €
C4000 Advanced Atex II 3G	Barrera láser	12,97 €	2	25,94 €
MGB-L1H-APA-R· 110458	Cerradura puerta acceso	954,00 €	2	1.908,00 €
			Total	10.943,28 €

Tabla 10 Presupuesto Elementos seguridad. Fuente: Elaboración propia

OTROS ELEMENTOS				
Modelo	**Descripción**	**Precio/unidad**	**Unidades**	**Precio total**
Pasarela IoT	Módulo comunicación	159,20 €	1	159,20 €
S7-1511-1PN Kit	PLC	1.258,75 €	1	1.258,75 €
IRB 6620 ABB	Robot manipulador	37.500,00 €	2	75.000,00 €
KTP700 Basic PN	HMI	680,02 €	2	1.360,04 €
			Total	77.777,99 €

Tabla 11 Presupuesto Elementos línea. Fuente: Elaboración propia

Siendo el total de los elementos de hardware:

Elemento	Precio
Sensores	830,50 €
Seguridad	10.943,28 €
Otros	77.777,99 €
Total	89.551,77 €

Tabla 12 Presupuesto Total hardware. Fuente: Elaboración propia

8.3 Recursos humanos

El coste de contratación de un ingeniero técnico industrial se ha estimado, realizando un pequeño estudio de la oferta actual, en 32 €/h. Con esta información y la cantidad de horas dedicadas a cada una de las partes del proyecto, el coste total se puede desglosar de la siguiente manera:

Actividad	Horas	Precio/hora	Precio total
Programación PLC + HMI	70	32,00 €	2.240,00 €
Programación SCADA	50	32,00 €	1.600,00 €
Diseño de línea	10	32,00 €	320,00 €
Análisis proceso	10	32,00 €	320,00 €
Documentación	15	32,00 €	480,00 €
		Total	4.960,00 €

Tabla 13 Presupuesto Ingeniero técnico. Fuente: Elaboración propia

8.4 Total

La suma total de los costes de este proyecto es:

Elemento	Coste
Hardware	89.551,77 €
Software	5.512,00 €
RH	4.960,00 €
Total	100.023,77 €

Tabla 14 Presupuesto final. Fuente: Elaboración propia

9. Conclusiones

Una vez completado este proyecto, se puede concluir que todos los objetivos iniciales se han cumplido de manera satisfactoria. La realización de este trabajo ha servido para asentar y ampliar los conocimientos sobre la automatización en el campo de la industria, experimentar el proceso de desarrollo de un proyecto de mayor complejidad y demostrar la experiencia conseguida durante los inicios de la etapa laboral.

Ha quedado demostrado que una línea productiva es un sistema de una gran complejidad, con muchos elementos a tener en cuenta. La programación de estos automatismos es crítica, y hace falta una gran experiencia y cuidado para evitar errores y posibles accidentes. Además, queda justificado que un sistema de diagnóstico completo es esencial para el mantenimiento y el correcto funcionamiento de estos sistemas.

Con el resultado obtenido del presupuesto, y teniendo en cuenta que con esa cantidad no se cubriría el proceso completo, se puede concluir que se trata de elementos productivos de un coste elevado, lo cual recalca la importancia de las fases de análisis y diseño, puesto que un error en éstas puede desembocar en un gran desembolso de capital por parte de la empresa.

A nivel personal, opino que este proyecto me ha permitido prepararme para enfocar mejor el mundo laboral, pues he tenido que esforzarme para poder demostrar mi valía como ingeniero y mi capacidad para aportar soluciones válidas a problemas reales.

Bibliografía

Documentación consultada

Páginas web:

- SIEMENS: https://new.siemens.com/global/en.html
- Foros técnicos SIEMENS: https://support.industry.siemens.com/tf/
- Fischertechnik: https://www.fischertechnik.de/en
- Documentación Matlab: https://es.mathworks.com/help/
- ABB: https://new.abb.com/
- Schneider: https://www.se.com/
- SICK: https://www.sick.com
- Euchner: https://www.euchner.de/en-us/
- KUKA: https://www.kuka.com/
- NetToPLCsim: http://nettoplcsim.sourceforge.net/

Documentos:

- Manual PLC SIEMENS S7 1200 - V4.3.0 - 02/2019.
- Manual NetToPLCsim - Network extension for PLCsim - Thomas Wiens - Versión 1.2.4 -Febrero 2018.
- Informe motor Artitecnic v2.0 – Raúl Simarro Fernández – DISA-UPV – Julio 2013.
- Robot manipulador de vacío FischerTechnik - Raúl Simarro Fernández – DISA-UPV.

Referencias

[1]. Documentación robot manipulador: Informe motor Artitecnic v2.0 – Raúl Simarro Fernández – DISA-UPV – Julio 2013 [Fecha de consulta marzo del 2020].

[2]. Documentación del motor eléctrico: Informe motor Artitecnic v2.0 – Raúl Simarro Fernández – DISA-UPV – Julio 2013 [Fecha de consulta junio del 2020].

[3]. Manual PLC S7 1200: Manual PLC SIEMENS S7 1200 - V4.3.0 - 02/2019. [Fecha de consulta febrero – junio 2020]

[4]. Profinet:

PI North America. Profinet technology [en línea]. profinet.com [fecha de consulta junio del 2020]. Disponible en <https://us.profinet.com/technology/profinet/>

[5]. Modbus:

Modbus Organization. MODBUS Messaging on TCP/IP Implementation Guide V1.0b [en línea]. Modbus.org, octubre de 2006. [fecha de consulta junio del 2020]. Disponible en <http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf>

[6]. Documentación IP-S7-LINK SIMATIC S7: IPS7LnkNet.Advanced.Matlab Documentation. Traeger.de, mayo de 2017 [fecha de consulta abril 2020].

[7] Agencia Estatal Boletín oficial del estado [en línea]. Gobierno de España. [fecha de consulta junio del 2020]. Disponible en <https://www.boe.es/>

[8] UNE, Normalización española [en línea]. UNE [Fecha de consulta junio del 2020]. Disponible en <https://www.une.org/>

Ilustraciones:

Ecuaciones

Ilustraciones y tablas del documento

Anexo 1: Programación PLC

Índice Anexo I

Condiciones de cambio de paso para el PLC 1

(El del PLC2 es muy similar, pero con más pasos de reserva):

PASO	Descripción	ERROR 1	ERROR 2	Siguiente Paso	Funciones llamadas
1	Llevar robot a Home	HomePosOK		2	Posicion_Home
2	Esperar orden de nueva pieza	NuevaPieza		3	
3	Ir a la posicion de aproximacion a pieza.	PosOK		4	Movimiento_Robot
4	Bajada a pieza y Activar compresor	PosOK	Temporizador _Compresor	5	Movimiento_Robot
5	Coger pieza	ONVentosaOK		6	Coger_Pieza
6	Levantar pieza	PosOK		7	Movimiento_Robot
7	Ir a posicion de dejar la pieza (Segun modelo).	PosOK		!Pieza_Defectuosa -8 Pieza_Defectuosa - 25	Movimiento_Robot
8	Bajada vertical a dejar pieza	PosOK		9	Movimiento_Robot
9	Dejar pieza en estacion intermedia	OFFVentosaOK		10	Dejar_Pieza
10	Subida vertical a posicion de aproximacion	PosOK		11	Movimiento_Robot
11	Ir a posicion de espera (Segun modelo)	PosOK		12	Movimiento_Robot
12	Esperar proceso completado	Temporizador _proceso		13	
13	Ir a posicion de aproximacion a pieza (Segun modelo)	PosOK		14	Movimiento_Robot
14	Bajada a pieza y activar compresor	PosOK	Temporizador _Compresor	15	Movimiento_Robot
15	Coger Pieza	ONVentosaOK		16	Coger_Pieza

Ilustración Anexo PLC- 1 Pasos PLC1 (1)

16	Levantar Pieza	PiezaOK	PosOK	17	Movimiento_Robot
17	Ir a posicion aproximacion dejada final (Al robot 2)	PosOK		!Pieza_Defectuosa -18 Pieza_Defectuosa - 25	Movimiento_Robot
18	RESERVA				
19	RESERVA				
20	RESERVA				
21	RESERVA				
22	Bajada vertical a dejar pieza	PosOK		23	Movimiento_Robot
23	Dejar pieza en posicion final	OFFVentosaOK		24	Dejar_Pieza
24	FIN DE CICLO: Subida vertical a aproximacion pieza.	PosOK		1	Movimiento_Robot
25	PIEZA DEF. Ir a Posicion descartar	PosOK		26	Movimiento_Robot
26	PIEZA DEF. Bajada vertical a dejar pieza	PosOK		27	Movimiento_Robot
27	PIEZA DEF. Dejar pieza	OFFVentosaOK		28	Dejar_Pieza
28	PIEZA DEF. Subida vertical	PosOK		1	Movimiento_Robot
29	RESERVA				
30	RESERVA				

Ilustración Anexo PLC- 2 Pasos PLC1 (2)

Matriz de fallos PLC 1

(La del PLC2 es muy similar pero con más pasos de reserva):

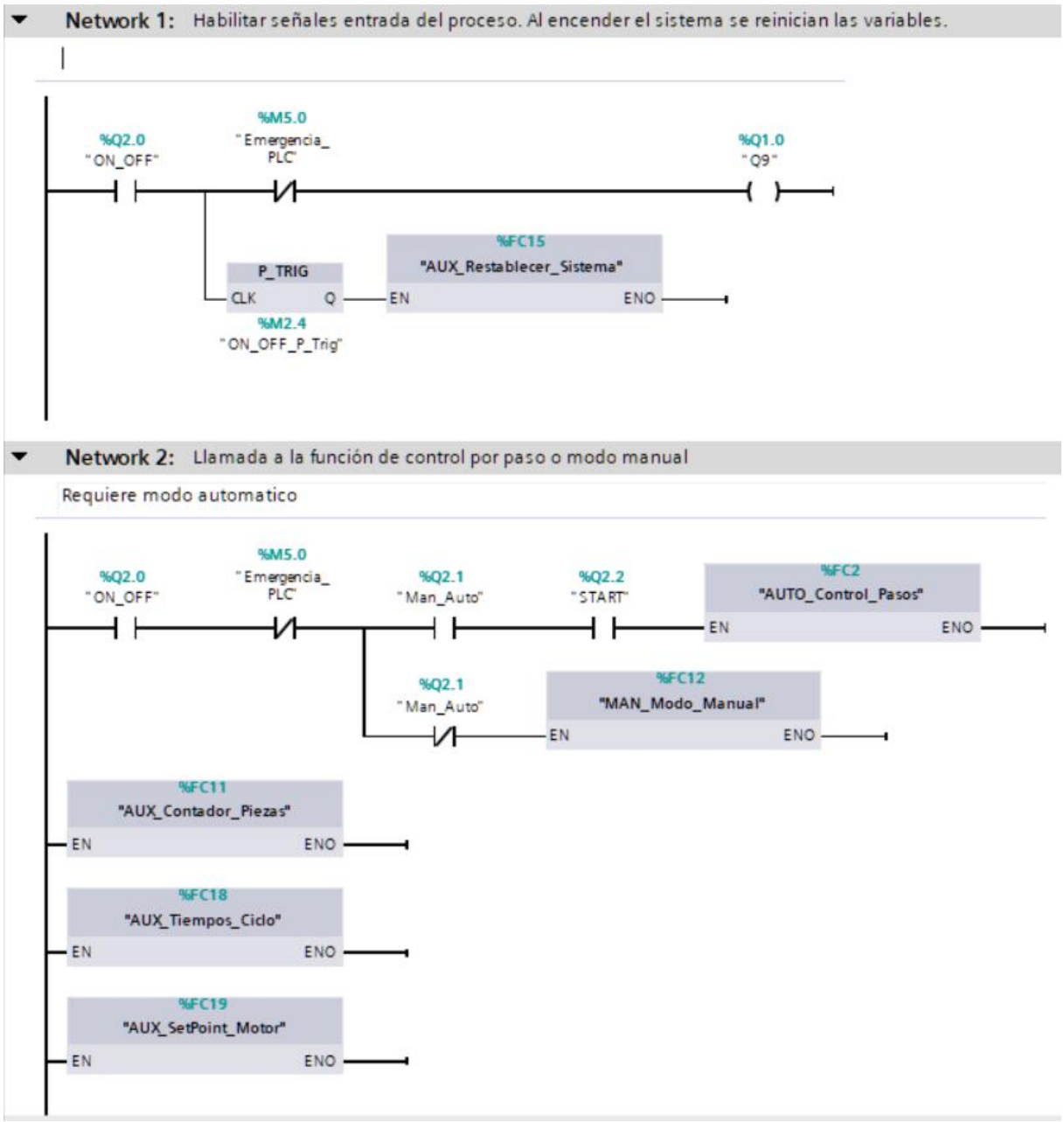
PASO\FALLO	0	1	2	3
1	. HomePosOK			
2	. NuevaPieza			
3	. PosOK			
4	. Temporizador_Compresor	. PosOK		
5	. ON_VentosaOK			
6	. PosOK			
7	. PosOK (M1)	. PosOK (M2)		
8	. PosOK			
9	. OFFVentosaOK			
10	. PosOK			
11	. PosOK (M1)	. PosOK (M2)		
12	. Temporizador_proceso			
13	. PosOK (M1)	. PosOK (M2)		
14	. Temporizador_Compresor	. PosOK		
15	. ON_VentosaOK			
16	. PosOK			
17	. PosOK	* R1_Pieza_Depositada_R2		
18	-			
19	-			
20	-			
21	-			
22	. PosOK			
23	. OFFVentosaOK			
24	. PosOK			
25	. PosOK			
26	. PosOK			
27	. OFFVentosaOK			
28	. PosOK			
29	-			
30	-			
Comunicaciones	Fallo bit vida PLC 2			
General	* Emergencia_PLC	* Emergencia_MAT	* Emergencia_HMI	. RESET

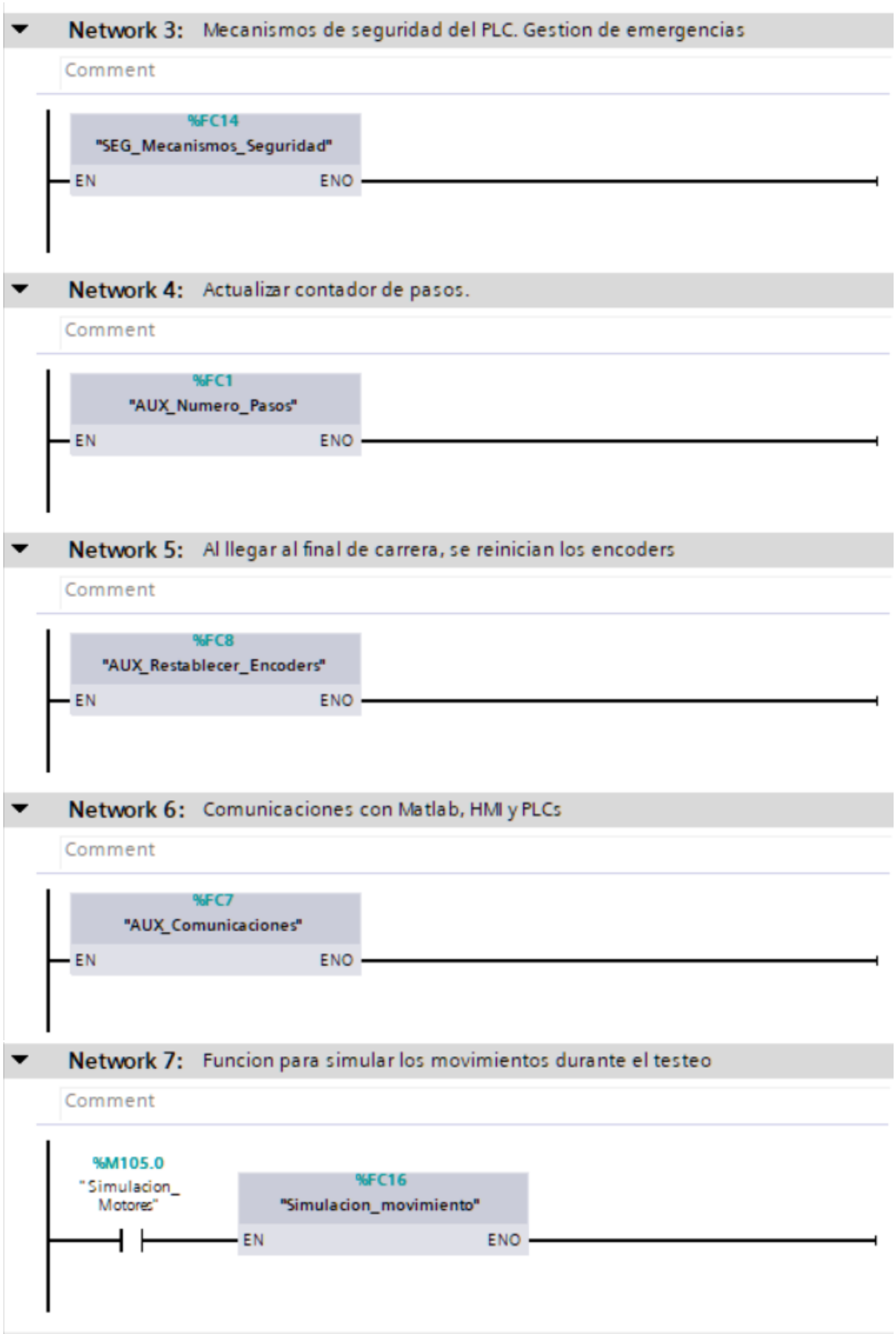
Ilustración Anexo PLC- 3 Matriz de fallos PLC1

Bloques de programación

En este apartado se muestra el programa desarrollado en TIA Portal para los PLC. Para ello se adjuntarán capturas de cada uno de los bloques de programación. Casi todo el código es igual para ambos PLC, y las partes que se diferencian siguen la misma lógica, cambiando sólo algunas variables, por tanto sólo se mostrará la programación del PLC1.

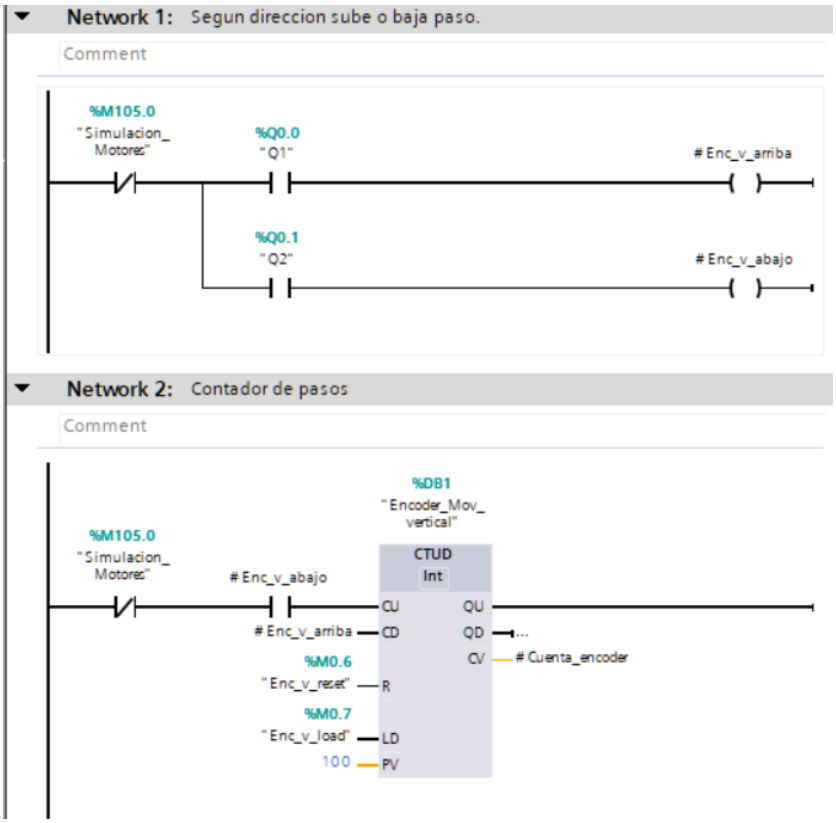
[OB1] Main

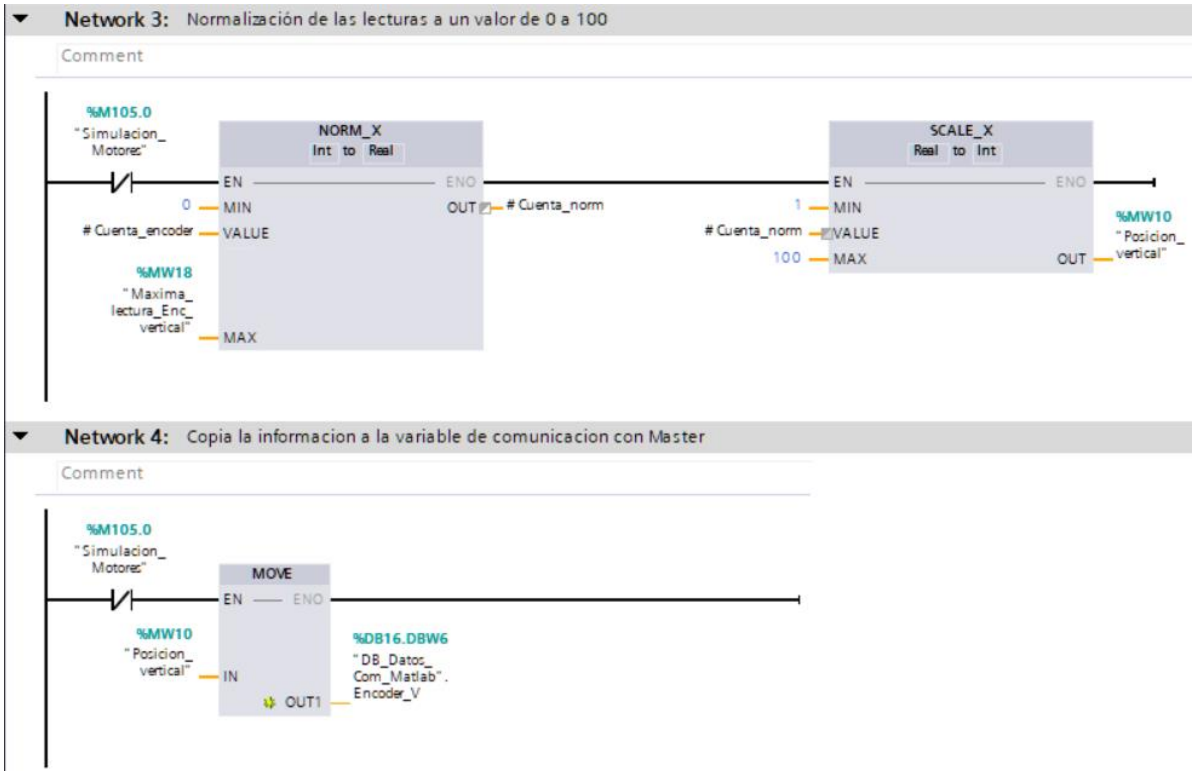



Network 3: Mecanismos de seguridad del PLC. Gestion de emergencias
Comment
%FC14
"SEG_Mecanismos_Seguridad"
EN
ENO
Network 4: Actualizar contador de pasos.
Comment
%FC1
"AUX_Numero_Pasos"
EN
ENO
Network 5: Al llegar al final de carrera, se reinician los encoders
Comment
%FC8
"AUX_Restablecer_Encoders"
EN
ENO
Network 6: Comunicaciones con Matlab, HMI y PLCs
Comment
%FC7
"AUX_Comunicaciones"
EN
ENO
Network 7: Funcion para simular los movimientos durante el testeo
Comment
%M105.0
"Simulacion_
Motores"
%FC16
"Simulacion_movimiento"
EN
ENO

[OB40] OB_HI_B1_vertical, [OB41] OB_HI_B3_Horizontal, [OB42] OB_HI_B5_giro

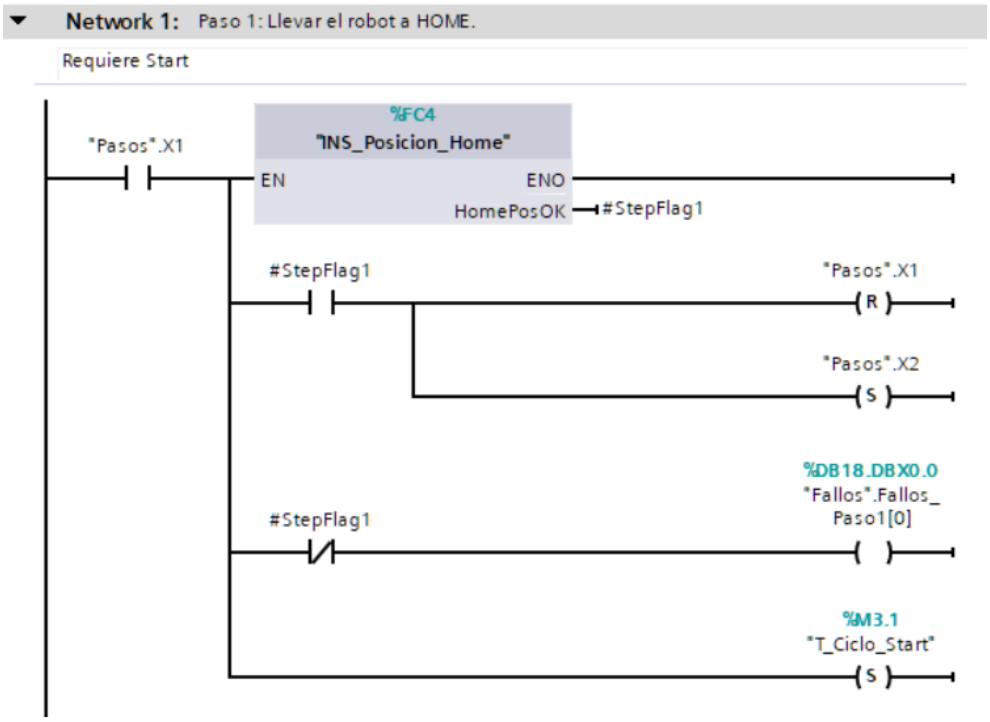
Los bloques horizontal y giro son iguales, cambiando únicamente las entradas Q

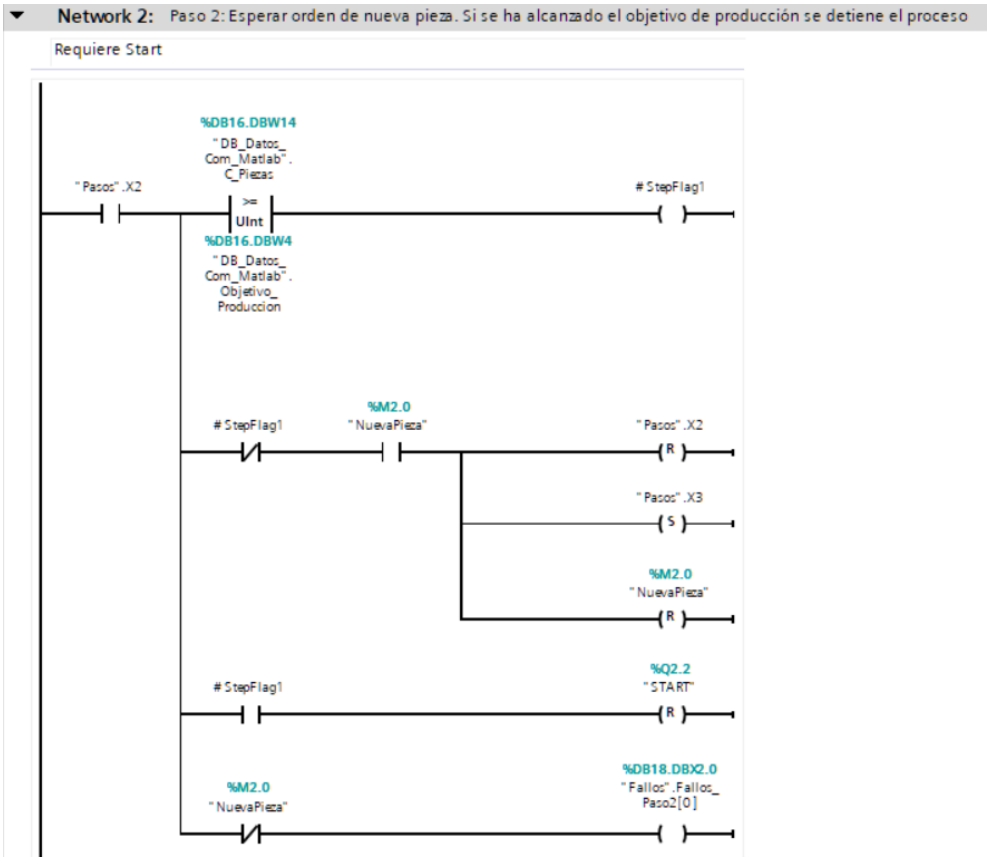


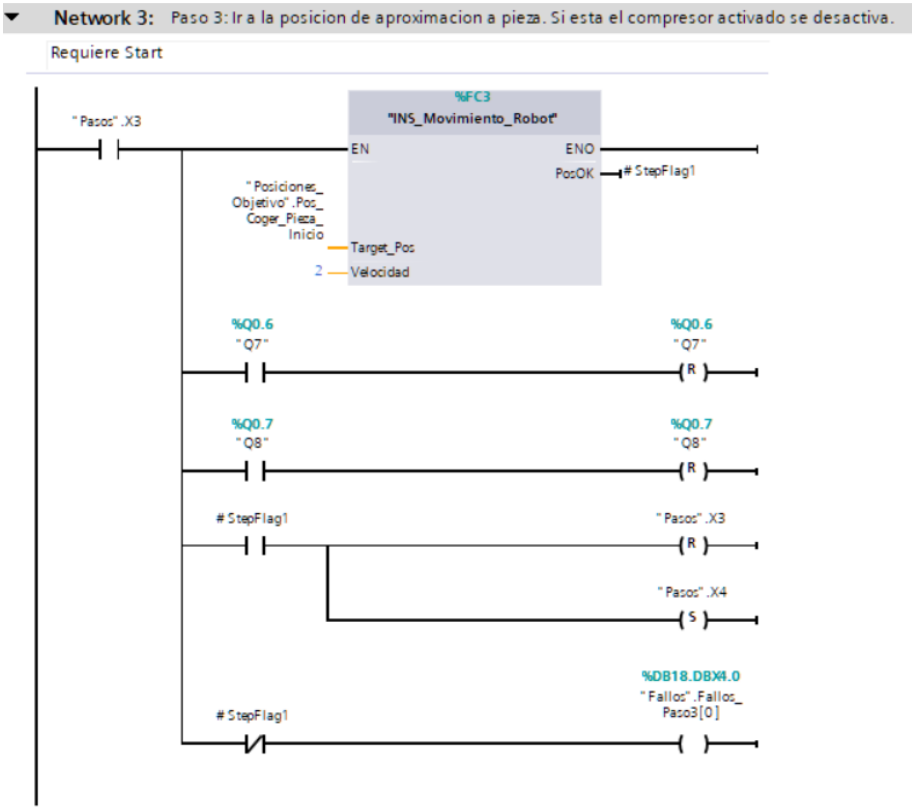


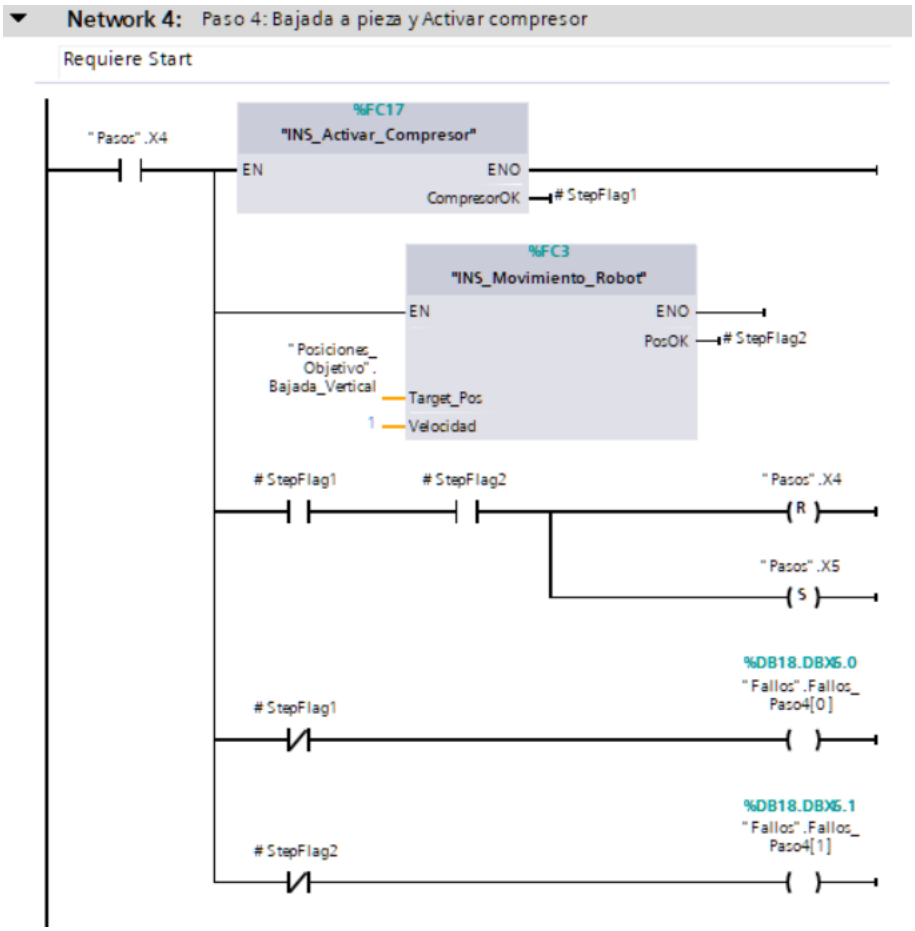
[FC2] AUTO_Control_Pasos

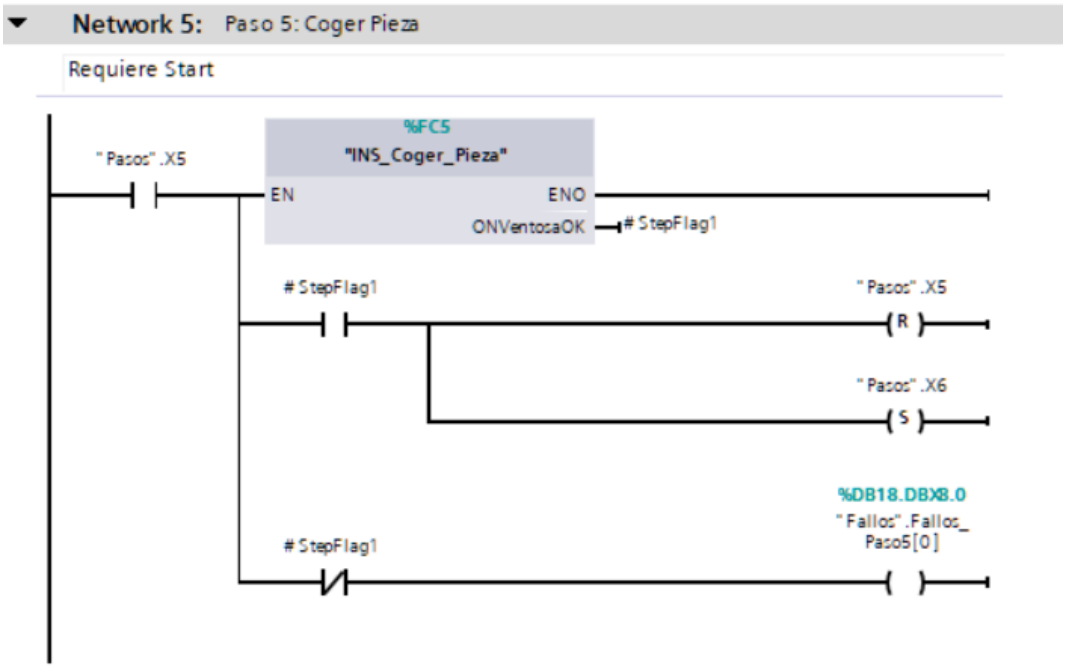
Los pasos no mostrados son de reserva y no contienen información relevante.

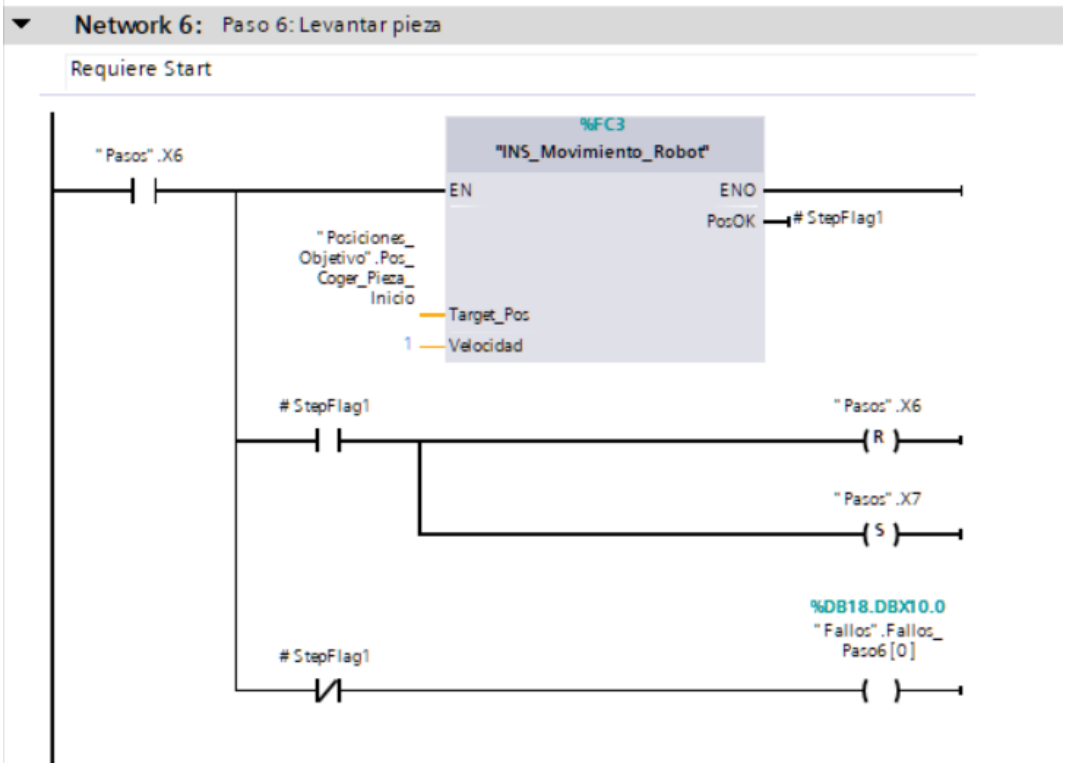


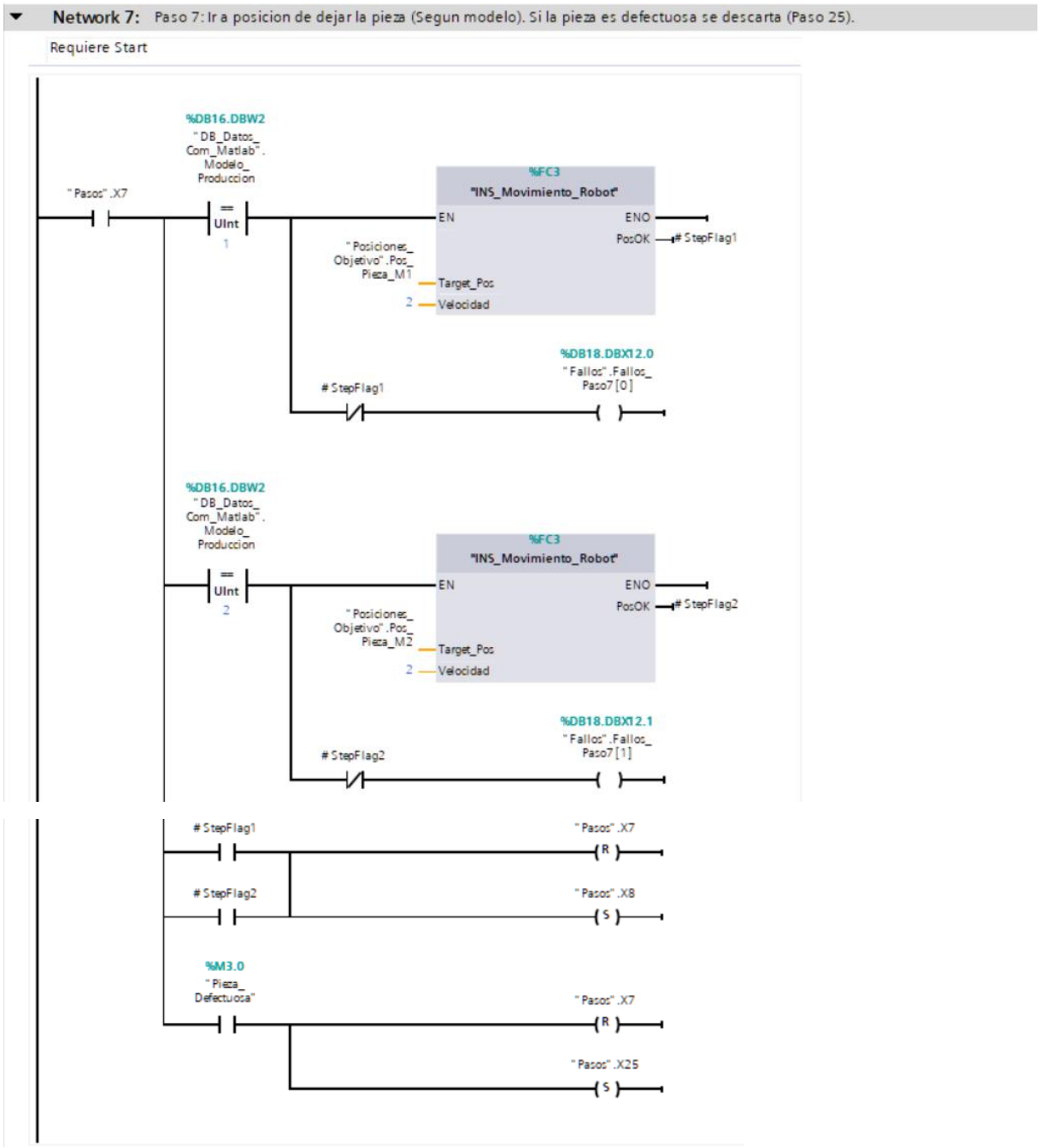


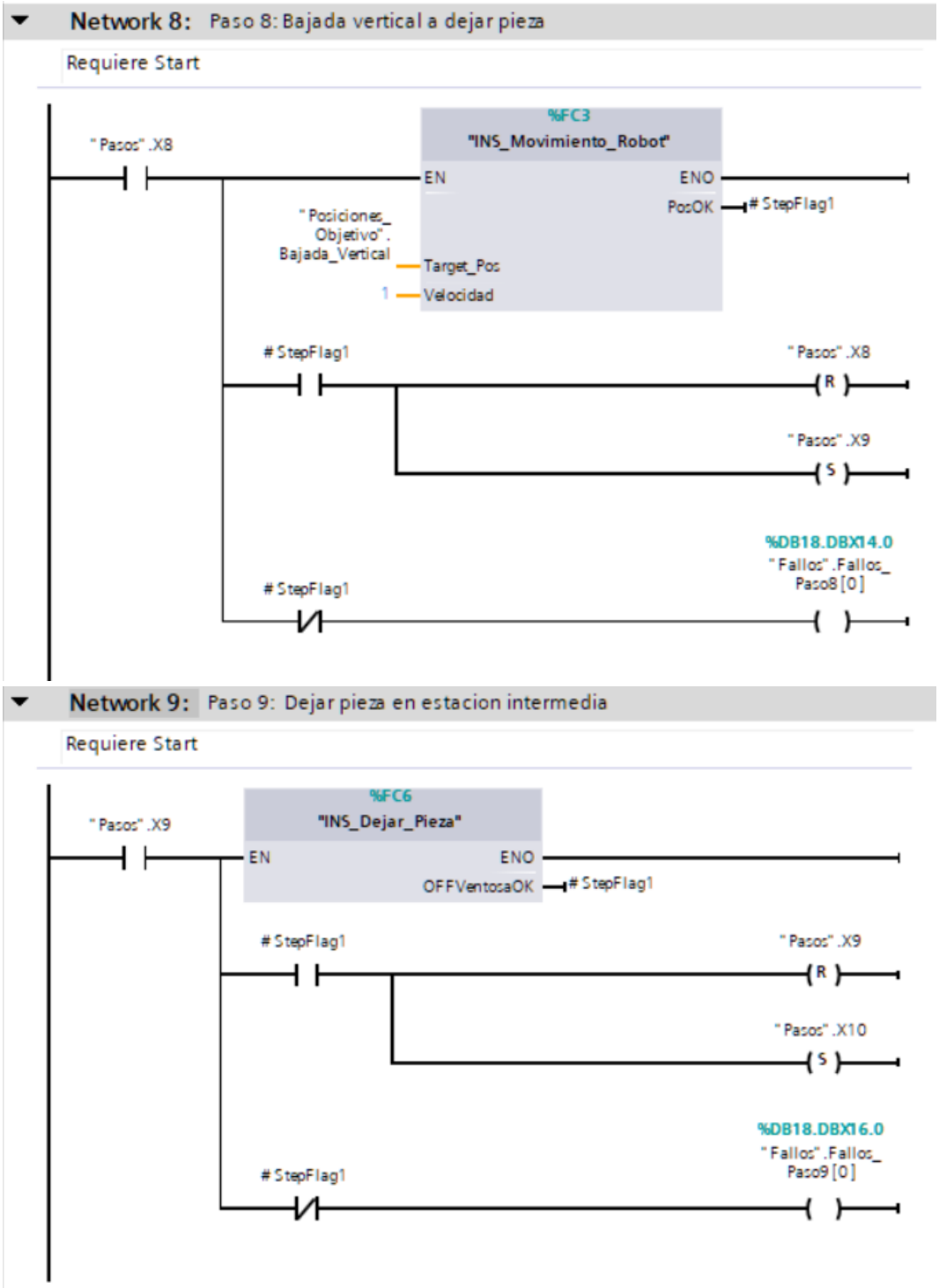

Network 3: Paso 3: Ir a la posicion de aproximacion a pieza. Si esta el compresor activado se desactiva.
Requiere Start
"Pasos".X3
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag1
"Posiciones_Objetivo".Pos_Coger_Pieza_Inicio
Target_Pos
2
Velocidad
%Q0.6
"Q7"
%Q0.7
"Q8"
#StepFlag1
"Pasos".X3
"Pasos".X4
%DB18.DBX4.0
"Fallos".Fallos_Paso3[0]

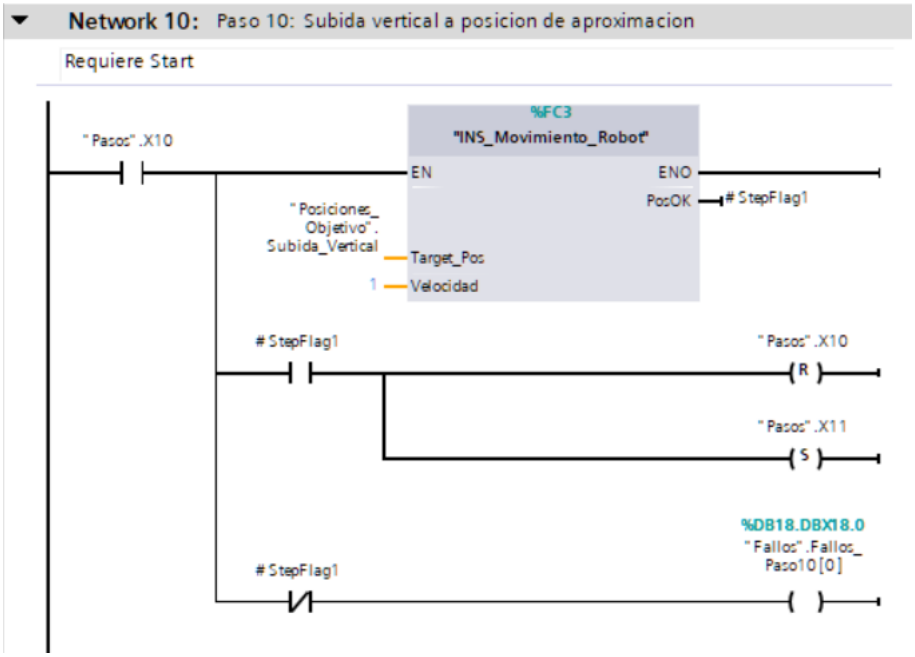

Network 4: Paso 4: Bajada a pieza y Activar compresor
Requiere Start
"Pasos".X4
%FC17
"INS_Activar_Compresor"
EN
ENO
CompresorOK
#StepFlag1
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag2
"Posiciones_Objetivo".Bajada_Vertical
Target_Pos
1
Velocidad
#StepFlag1
#StepFlag2
"Pasos".X4
"Pasos".X5
%DB18.DBX5.0
"Fallos".Fallos_Paso4[0]
%DB18.DBX5.1
"Fallos".Fallos_Paso4[1]

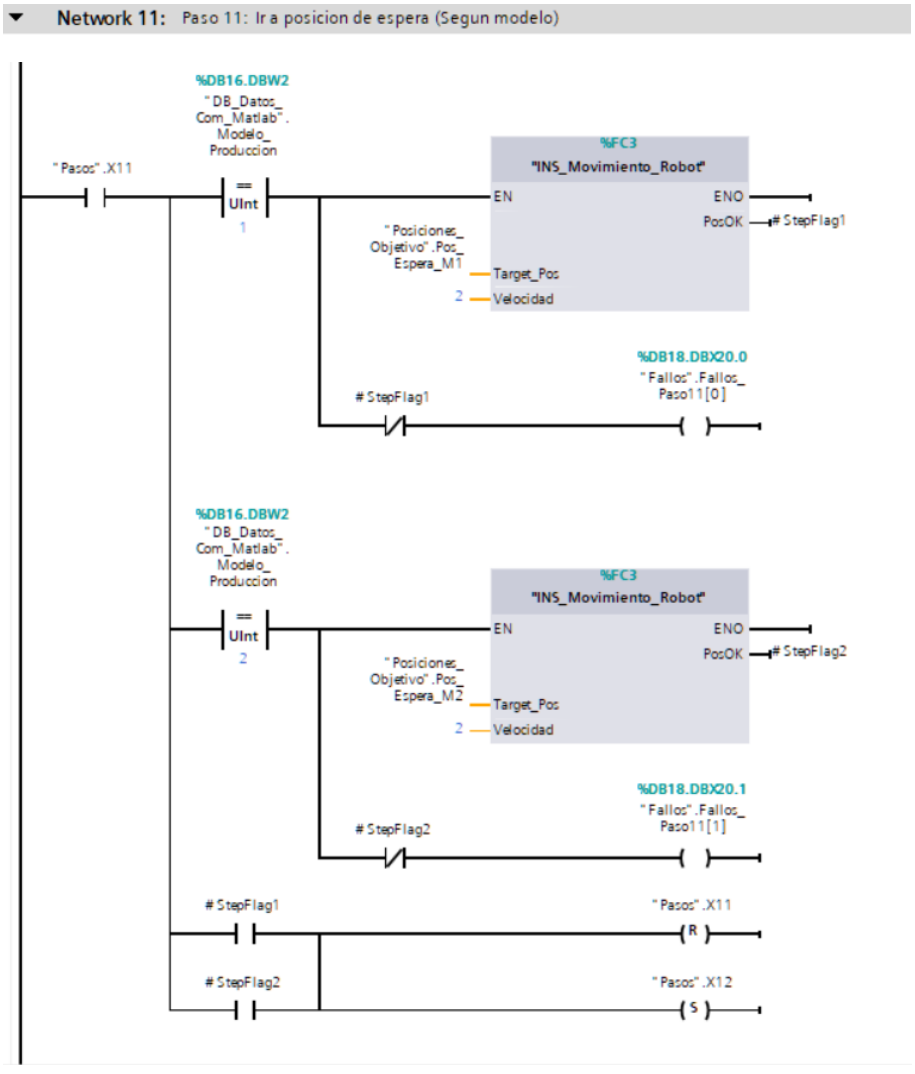

Network 5: Paso 5: Coger Pieza
Requiere Start
%FC5
"INS_Coger_Pieza"
"Pasos".X5
EN
ENO
ONVentosaOK
#StepFlag1
#StepFlag1
"Pasos".X5
R
"Pasos".X6
S
%DB18.DBX8.0
"Fallos".Fallos_Paso5[0]
#StepFlag1

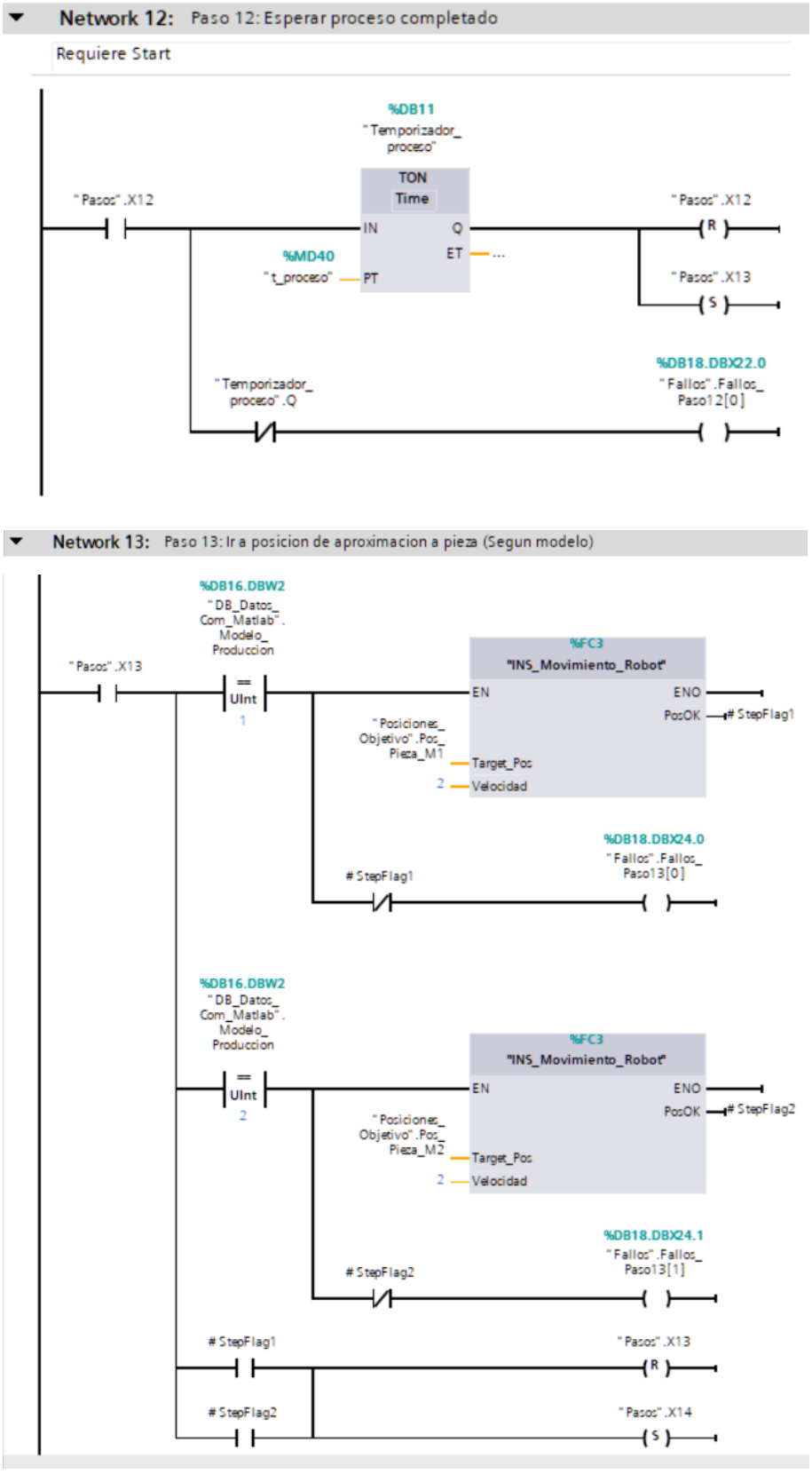

Network 6: Paso 6: Levantar pieza
Requiere Start
%FC3
"INS_Movimiento_Robot"
"Pasos".X6
EN
ENO
PosOK
#StepFlag1
"Posiciones_Objetivo".Pos_Coger_Pieza_Inicio
Target_Pos
1
Velocidad
#StepFlag1
"Pasos".X6
R
"Pasos".X7
S
%DB18.DBX10.0
"Fallos".Fallos_Paso6[0]
#StepFlag1

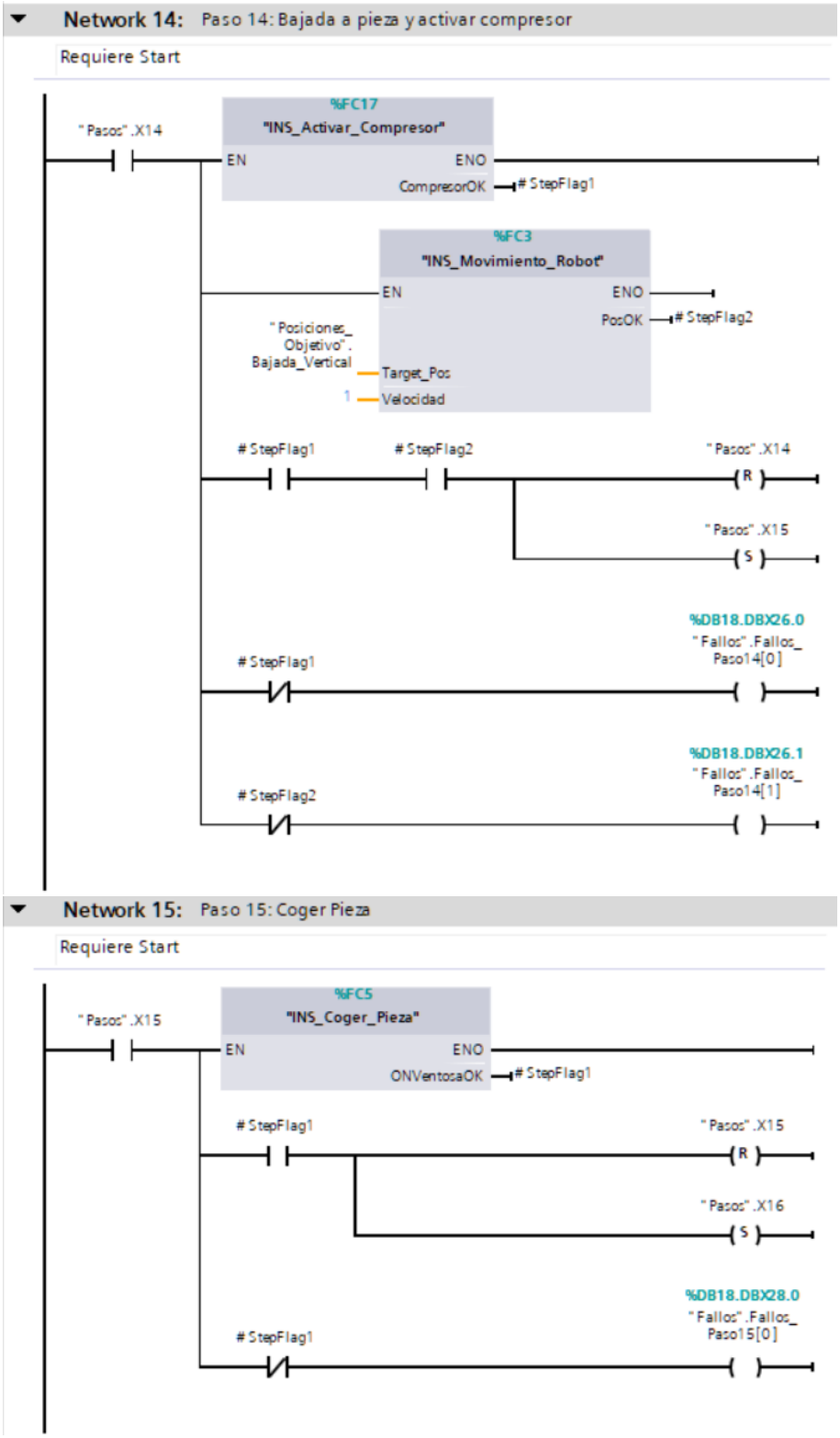

Network 7: Paso 7: Ir a posicion de dejar la pieza (Segun modelo). Si la pieza es defectuosa se descarta (Paso 25).
Requiere Start
%DB16.DBW2
"DB_Datos_Com_Matlab".Modelo_Produccion
"Pasos".X7
==
UInt
1
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag1
"Posiciones_Objetivo".Pos_Pieza_M1
Target_Pos
2
Velocidad
%DB18.DBX12.0
"Fallos".Fallos_Paso7[0]
#StepFlag1
%DB16.DBW2
"DB_Datos_Com_Matlab".Modelo_Produccion
==
UInt
2
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag2
"Posiciones_Objetivo".Pos_Pieza_M2
Target_Pos
2
Velocidad
%DB18.DBX12.1
"Fallos".Fallos_Paso7[1]
#StepFlag2
#StepFlag1
"Pasos".X7
R
#StepFlag2
"Pasos".X8
S
%M3.0
"Pieza_Defectuosa"
"Pasos".X7
R
"Pasos".X25
S

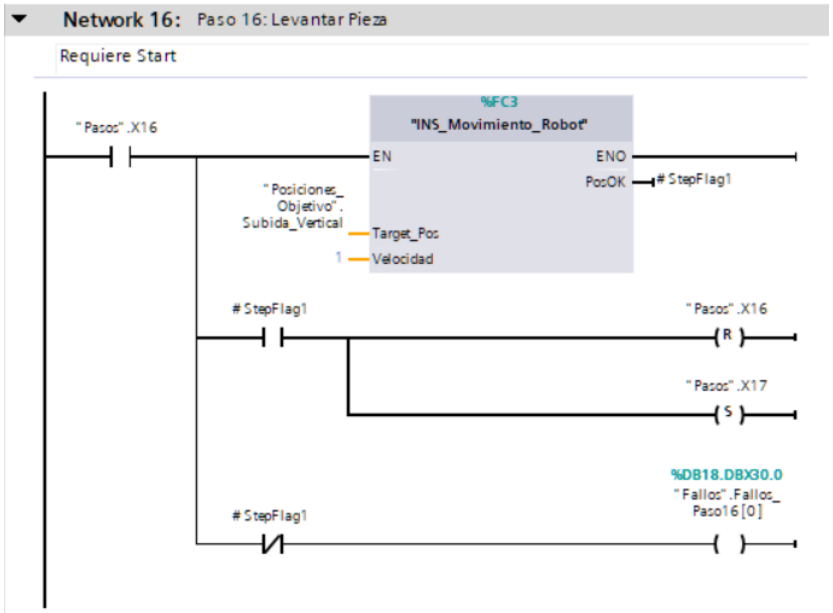

Network 8: Paso 8: Bajada vertical a dejar pieza
Requiere Start
"Pasos".X8
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag1
"Posiciones_Objetivo".Bajada_Vertical
Target_Pos
1
Velocidad
#StepFlag1
"Pasos".X8
R
"Pasos".X9
S
%DB18.DBX14.0
"Fallos".Fallos_Paso8[0]
#StepFlag1
Network 9: Paso 9: Dejar pieza en estacion intermedia
Requiere Start
"Pasos".X9
%FC6
"INS_Dejar_Pieza"
EN
ENO
OFFVentosaOK
#StepFlag1
#StepFlag1
"Pasos".X9
R
"Pasos".X10
S
%DB18.DBX16.0
"Fallos".Fallos_Paso9[0]
#StepFlag1

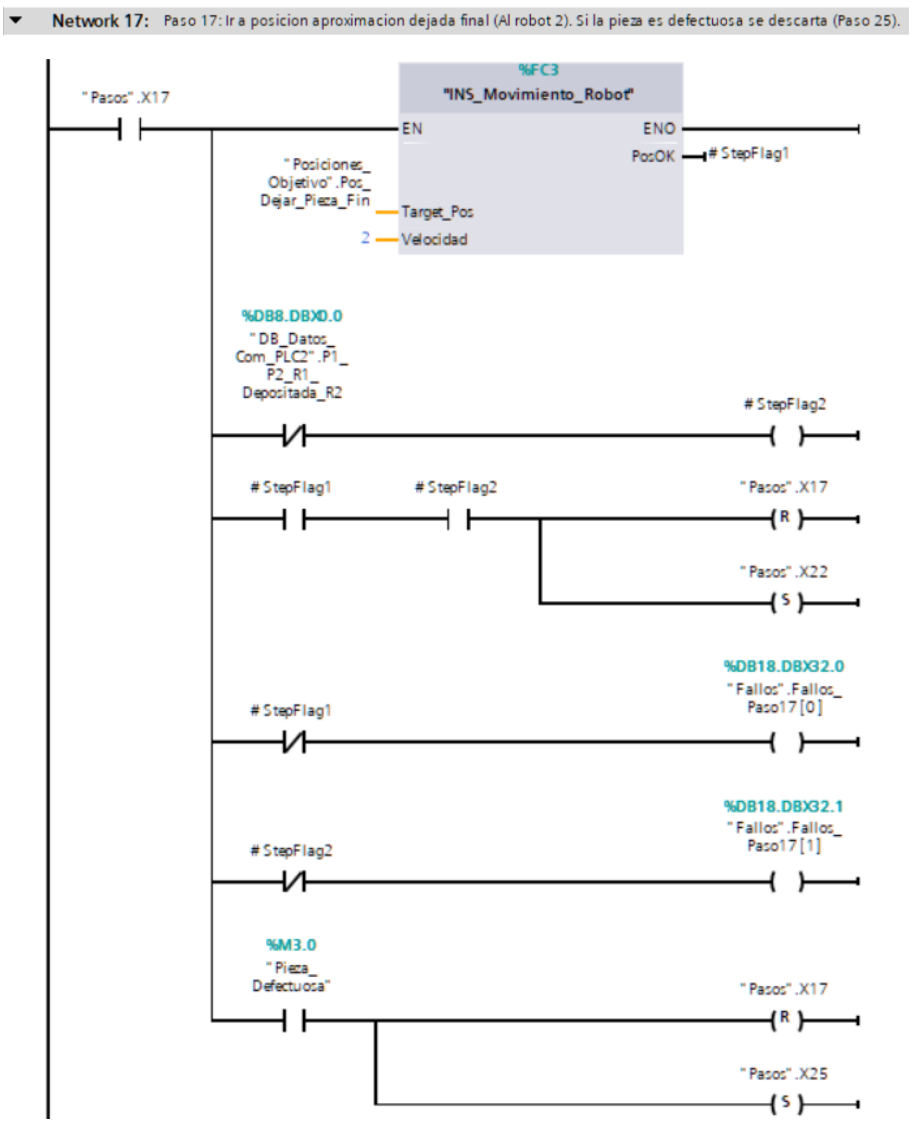

Network 10: Paso 10: Subida vertical a posicion de aproximacion
Requiere Start
"Pasos".X10
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
StepFlag1
"Posiciones_Objetivo".Subida_Vertical
Target_Pos
1
Velocidad
StepFlag1
"Pasos".X10
R
"Pasos".X11
S
%DB18.DBX18.0
"Fallos".Fallos_Paso10[0]
StepFlag1

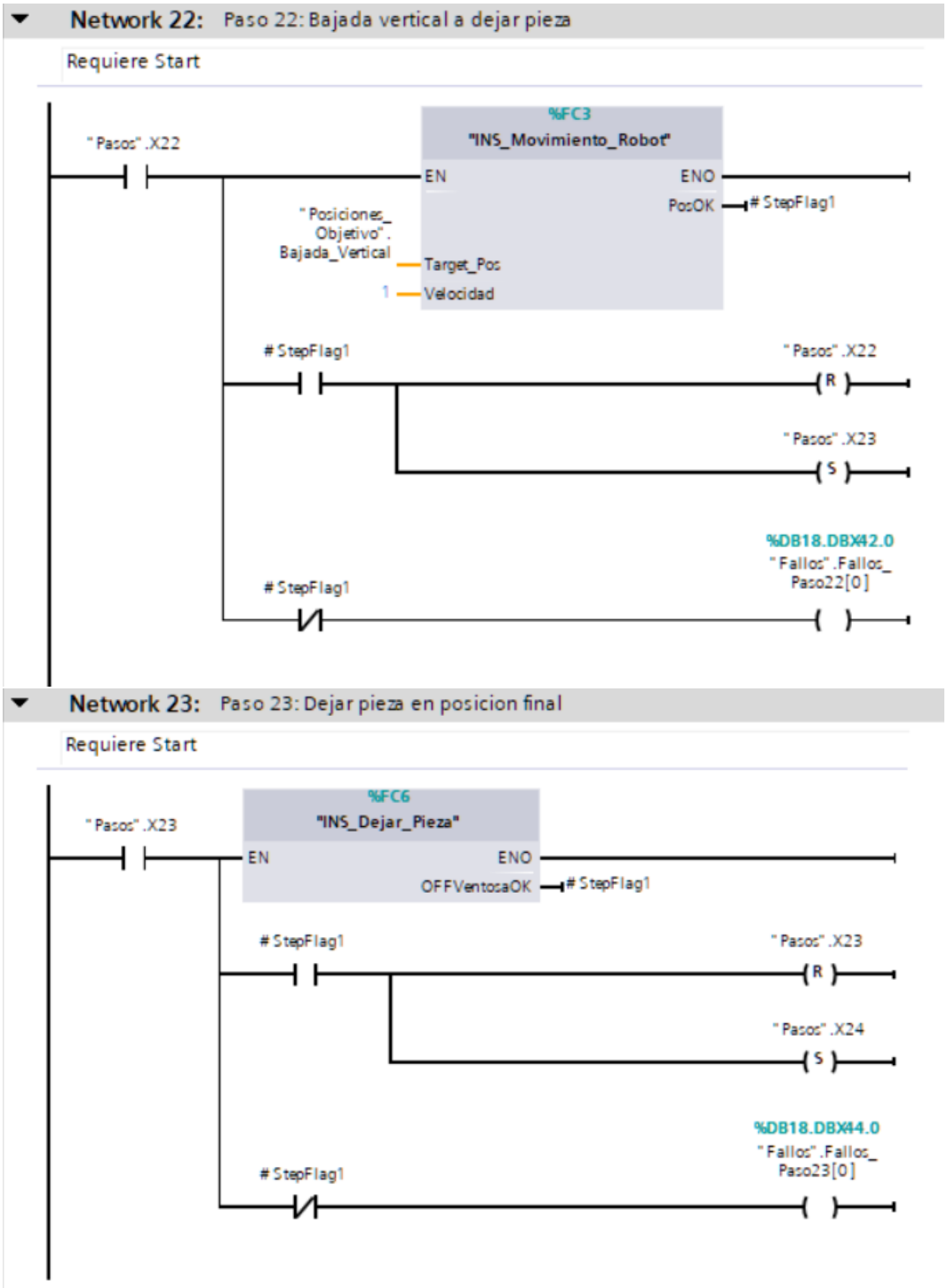

Network 11: Paso 11: Ir a posicion de espera (Segun modelo)
%DB16.DBW2
"DB_Datos_Com_Matlab".Modelo_Produccion
"Pasos".X11
==
UInt
1
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
StepFlag1
"Posiciones_Objetivo".Pos_Espera_M1
Target_Pos
2
Velocidad
%DB18.DBX20.0
"Fallos".Fallos_Paso11[0]
StepFlag1
%DB16.DBW2
"DB_Datos_Com_Matlab".Modelo_Produccion
==
UInt
2
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
StepFlag2
"Posiciones_Objetivo".Pos_Espera_M2
Target_Pos
2
Velocidad
%DB18.DBX20.1
"Fallos".Fallos_Paso11[1]
StepFlag2
StepFlag1
"Pasos".X11
R
StepFlag2
"Pasos".X12
S

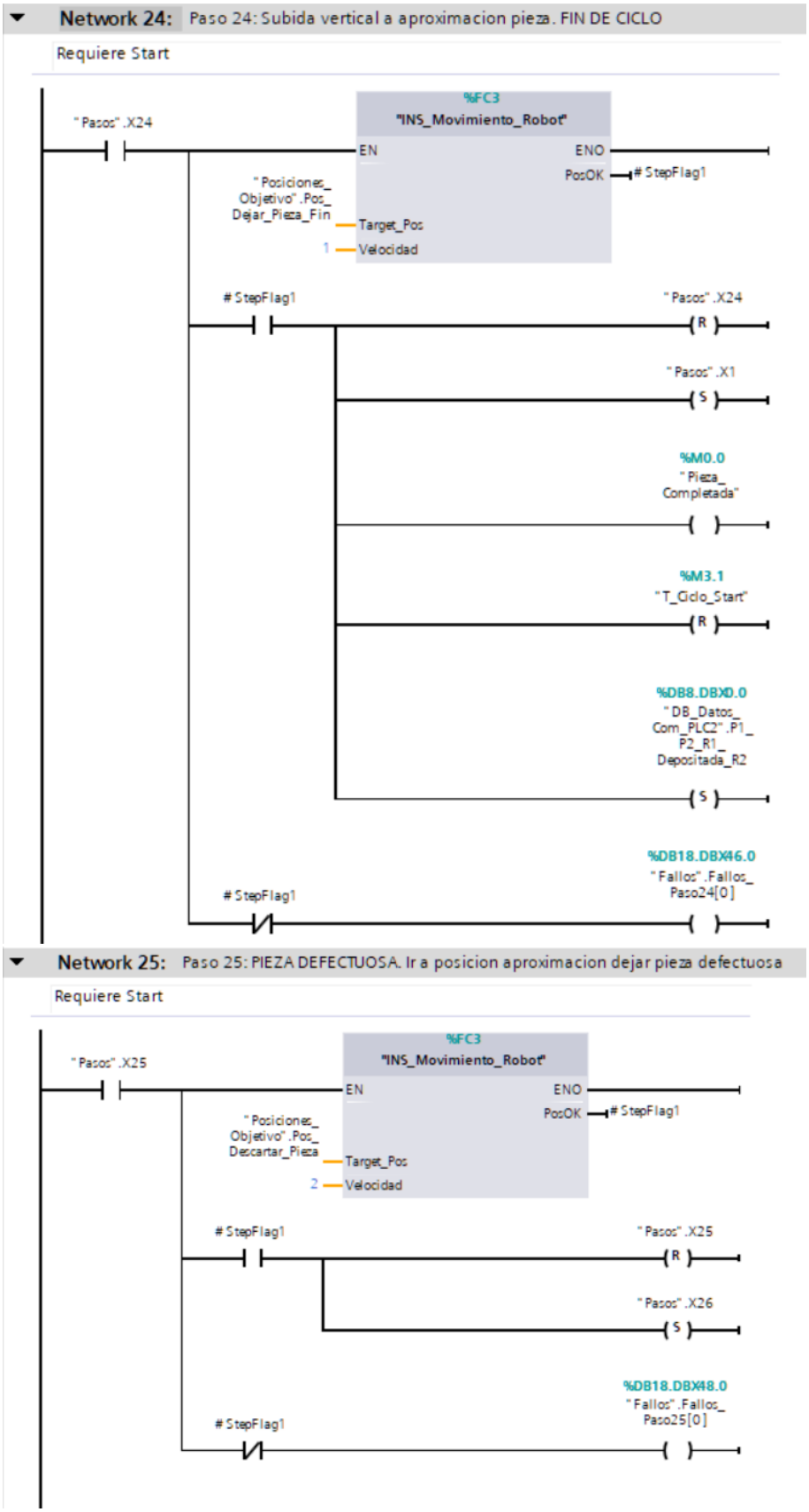

Network 12: Paso 12: Esperar proceso completado
Requiere Start
%DB11
"Temporizador_proceso"
TON
Time
"Pasos".X12
IN
Q
ET
%MD40
"t_proceso"
PT
"Pasos".X12
R
"Pasos".X13
S
%DB18.DBX22.0
"Fallos".Fallos_Paso12[0]
"Temporizador_proceso".Q
Network 13: Paso 13: Ir a posicion de aproximacion a pieza (Segun modelo)
%DB16.DBW2
"DB_Datos_Com_Matlab".Modelo_Produccion
"Pasos".X13
==
UInt
1
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag1
"Posiciones_Objetivo".Pos_Pieza_M1
Target_Pos
2
Velocidad
%DB18.DBX24.0
"Fallos".Fallos_Paso13[0]
#StepFlag1
%DB16.DBW2
"DB_Datos_Com_Matlab".Modelo_Produccion
==
UInt
2
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag2
"Posiciones_Objetivo".Pos_Pieza_M2
Target_Pos
2
Velocidad
%DB18.DBX24.1
"Fallos".Fallos_Paso13[1]
#StepFlag2
#StepFlag1
"Pasos".X13
R
#StepFlag2
"Pasos".X14
S

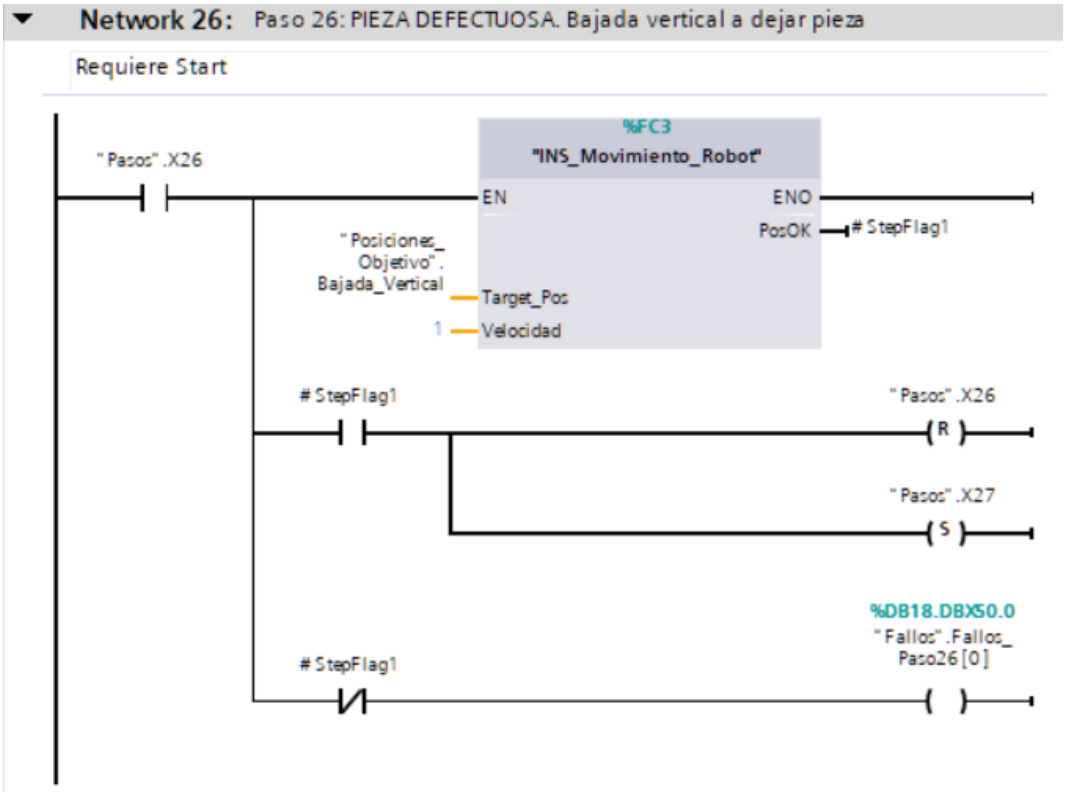

Network 14: Paso 14: Bajada a pieza y activar compresor
Requiere Start
"Pasos".X14
%FC17
"INS_Activar_Compresor"
EN
ENO
CompresorOK
#StepFlag1
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag2
"Posiciones_Objetivo".Bajada_Vertical
Target_Pos
1
Velocidad
#StepFlag1
#StepFlag2
"Pasos".X14
R
"Pasos".X15
S
%DB18.DBX26.0
"Fallos".Fallos_Paso14[0]
#StepFlag1
%DB18.DBX26.1
"Fallos".Fallos_Paso14[1]
#StepFlag2
Network 15: Paso 15: Coger Pieza
Requiere Start
"Pasos".X15
%FC5
"INS_Coger_Pieza"
EN
ENO
ONVentosaOK
#StepFlag1
#StepFlag1
"Pasos".X15
R
"Pasos".X16
S
%DB18.DBX28.0
"Fallos".Fallos_Paso15[0]
#StepFlag1

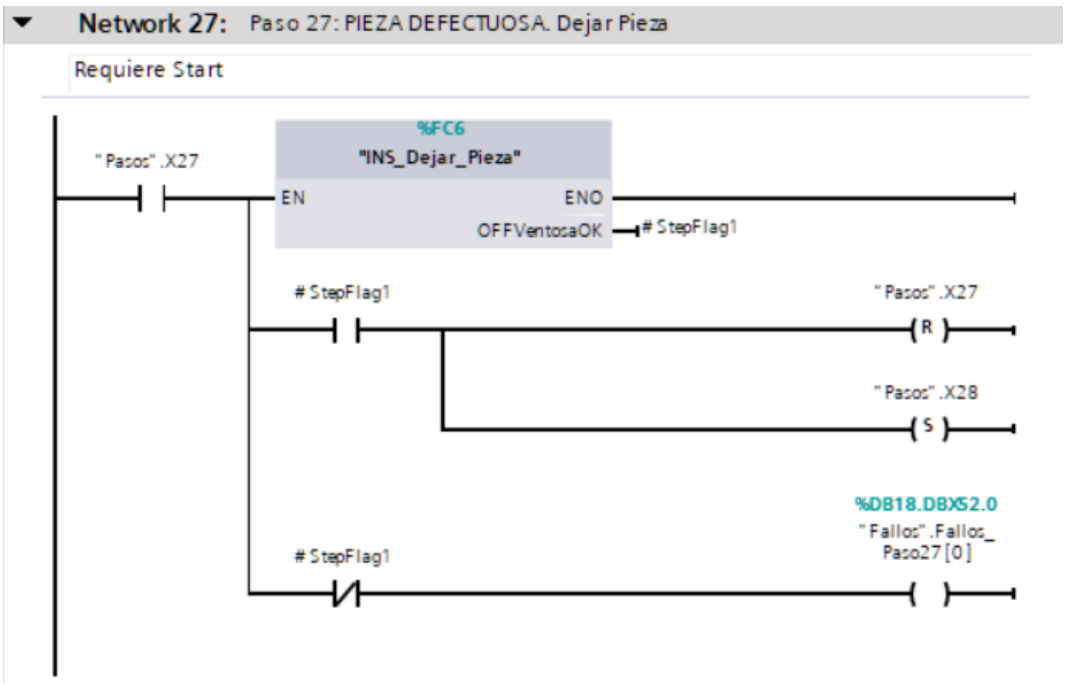

Network 16: Paso 16: Levantar Pieza
Requiere Start
%FC3
"INS_Movimiento_Robot"
"Pasos".X16
EN
ENO
PosOK
#StepFlag1
"Posiciones_Objetivo".Subida_Vertical
Target_Pos
1
Velocidad
#StepFlag1
"Pasos".X16
R
"Pasos".X17
S
%DB18.DBX30.0
"Fallos".Fallos_Paso16[0]
#StepFlag1

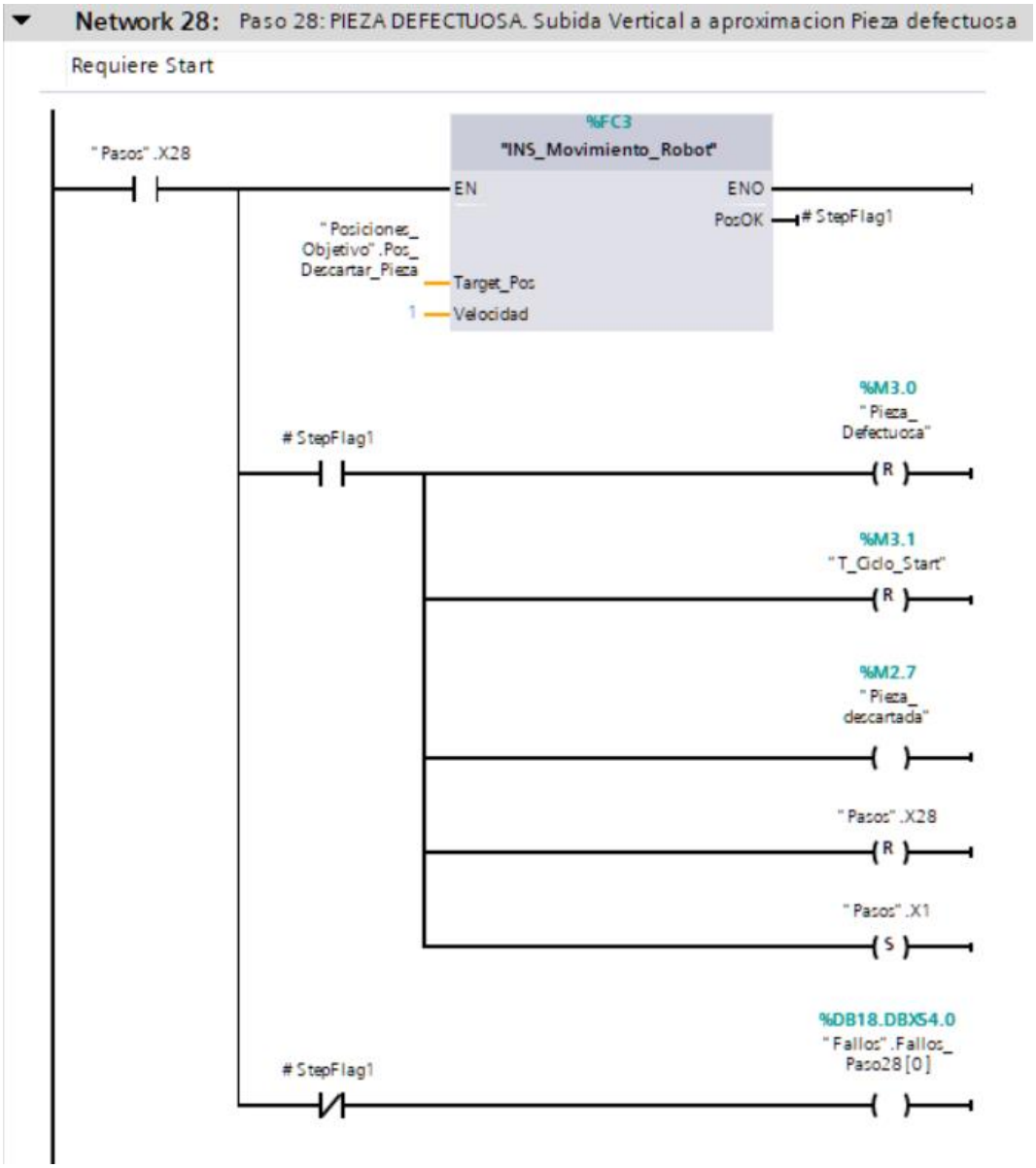

Network 17: Paso 17: Ir a posicion aproximacion dejada final (Al robot 2). Si la pieza es defectuosa se descarta (Paso 25).
%FC3
"INS_Movimiento_Robot"
"Pasos".X17
EN
ENO
PosOK
#StepFlag1
"Posiciones_Objetivo".Pos_Dejar_Pieza_Fin
Target_Pos
2
Velocidad
%DB8.DBX0.0
"DB_Datos_Com_PLC2".P1_P2_R1_Depositada_R2
#StepFlag2
#StepFlag1
#StepFlag2
"Pasos".X17
R
"Pasos".X22
S
%DB18.DBX32.0
"Fallos".Fallos_Paso17[0]
#StepFlag1
%DB18.DBX32.1
"Fallos".Fallos_Paso17[1]
#StepFlag2
%M3.0
"Pieza_Defectuosa"
"Pasos".X17
R
"Pasos".X25
S


Network 22: Paso 22: Bajada vertical a dejar pieza
Requiere Start
"Pasos".X22
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag1
"Posiciones_Objetivo".Bajada_Vertical
Target_Pos
1
Velocidad
#StepFlag1
"Pasos".X22
R
"Pasos".X23
S
%DB18.DBX42.0
"Fallos".Fallos_Paso22[0]
#StepFlag1
Network 23: Paso 23: Dejar pieza en posicion final
Requiere Start
"Pasos".X23
%FC6
"INS_Dejar_Pieza"
EN
ENO
OFFVentosaOK
#StepFlag1
#StepFlag1
"Pasos".X23
R
"Pasos".X24
S
%DB18.DBX44.0
"Fallos".Fallos_Paso23[0]
#StepFlag1

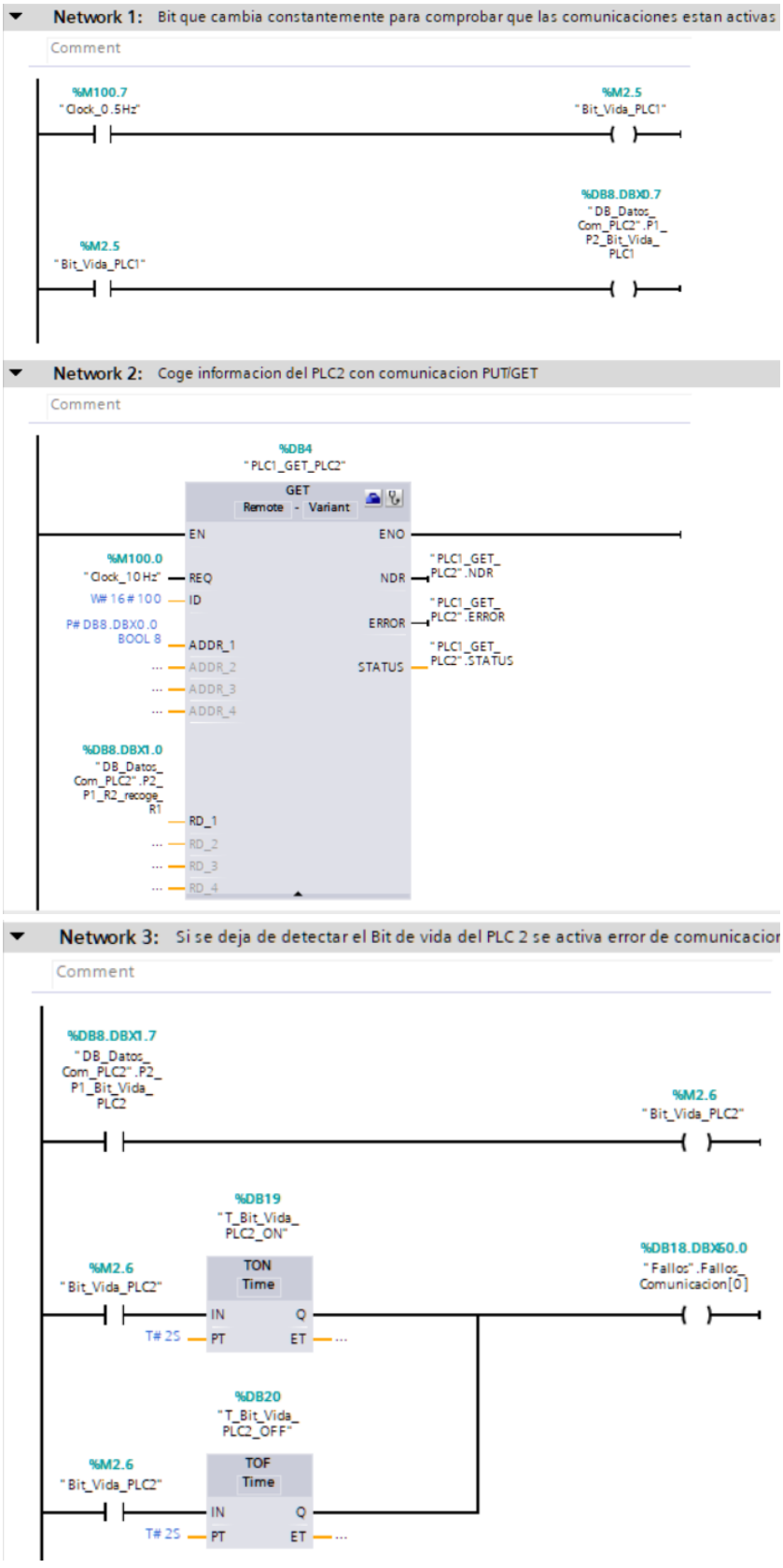

Network 24: Paso 24: Subida vertical a aproximacion pieza. FIN DE CICLO
Requiere Start
"Pasos".X24
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag1
"Posiciones_Objetivo".Pos_Dejar_Pieza_Fin
Target_Pos
1
Velocidad
#StepFlag1
"Pasos".X24
R
"Pasos".X1
S
%M0.0
"Pieza_Completada"
%M3.1
"T_Ciclo_Start"
%DB8.DBX0.0
"DB_Datos_Com_PLC2".P1_P2_R1_Depositada_R2
%DB18.DBX46.0
"Fallos".Fallos_Paso24[0]
Network 25: Paso 25: PIEZA DEFECTUOSA. Ir a posicion aproximacion dejar pieza defectuosa
Requiere Start
"Pasos".X25
"Posiciones_Objetivo".Pos_Descartar_Pieza
2
"Pasos".X26
%DB18.DBX48.0
"Fallos".Fallos_Paso25[0]

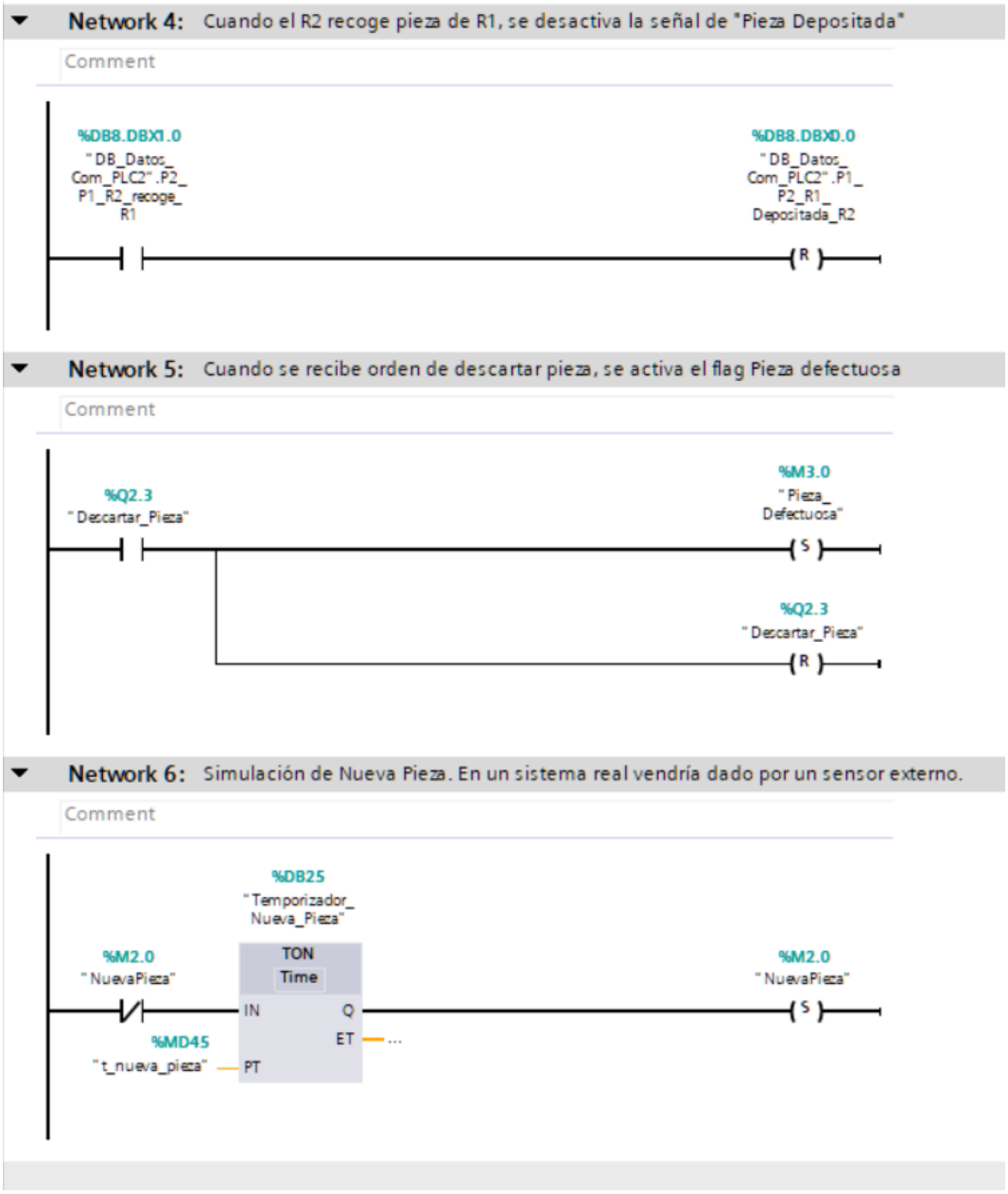

Network 26: Paso 26: PIEZA DEFECTUOSA. Bajada vertical a dejar pieza
Requiere Start
"Pasos".X26
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag1
"Posiciones_Objetivo".Bajada_Vertical
Target_Pos
1
Velocidad
#StepFlag1
"Pasos".X26
R
"Pasos".X27
S
%DB18.DBX50.0
"Fallos".Fallos_Paso26[0]
#StepFlag1


Network 27: Paso 27: PIEZA DEFECTUOSA. Dejar Pieza
Requiere Start
"Pasos".X27
%FC6
"INS_Dejar_Pieza"
EN
ENO
OFFVentosaOK
#StepFlag1
#StepFlag1
"Pasos".X27
R
"Pasos".X28
S
%DB18.DBX52.0
"Fallos".Fallos_Paso27[0]
#StepFlag1

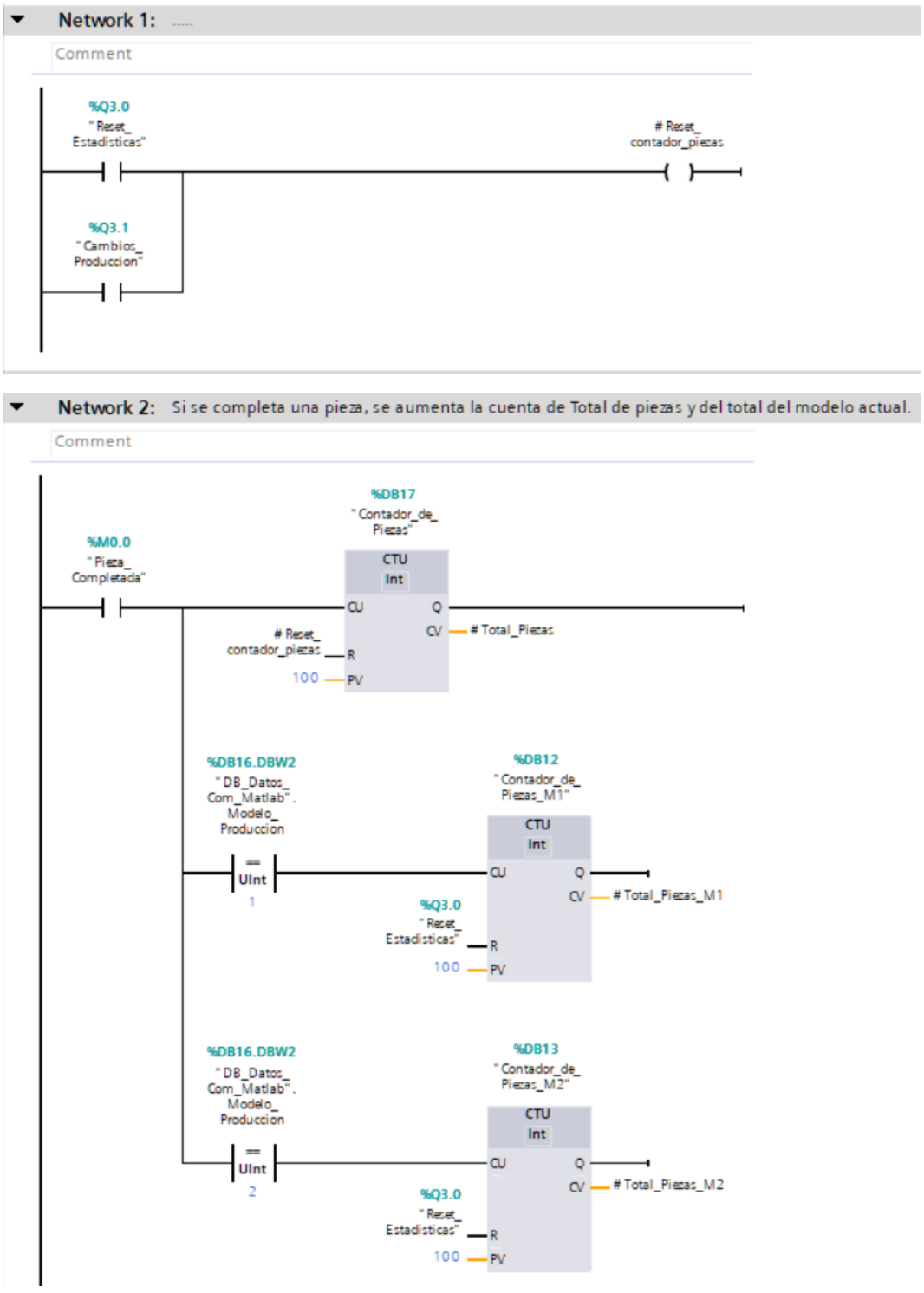

Network 28: Paso 28: PIEZA DEFECTUOSA. Subida Vertical a aproximacion Pieza defectuosa
Requiere Start
"Pasos".X28
%FC3
"INS_Movimiento_Robot"
EN
ENO
PosOK
#StepFlag1
"Posiciones_Objetivo".Pos_Descartar_Pieza
Target_Pos
1
Velocidad
%M3.0
"Pieza_Defectuosa"
#StepFlag1
R
%M3.1
"T_Ciclo_Start"
R
%M2.7
"Pieza_descartada"
"Pasos".X28
R
"Pasos".X1
S
%DB18.DBX54.0
"Fallos".Fallos_Paso28[0]
#StepFlag1

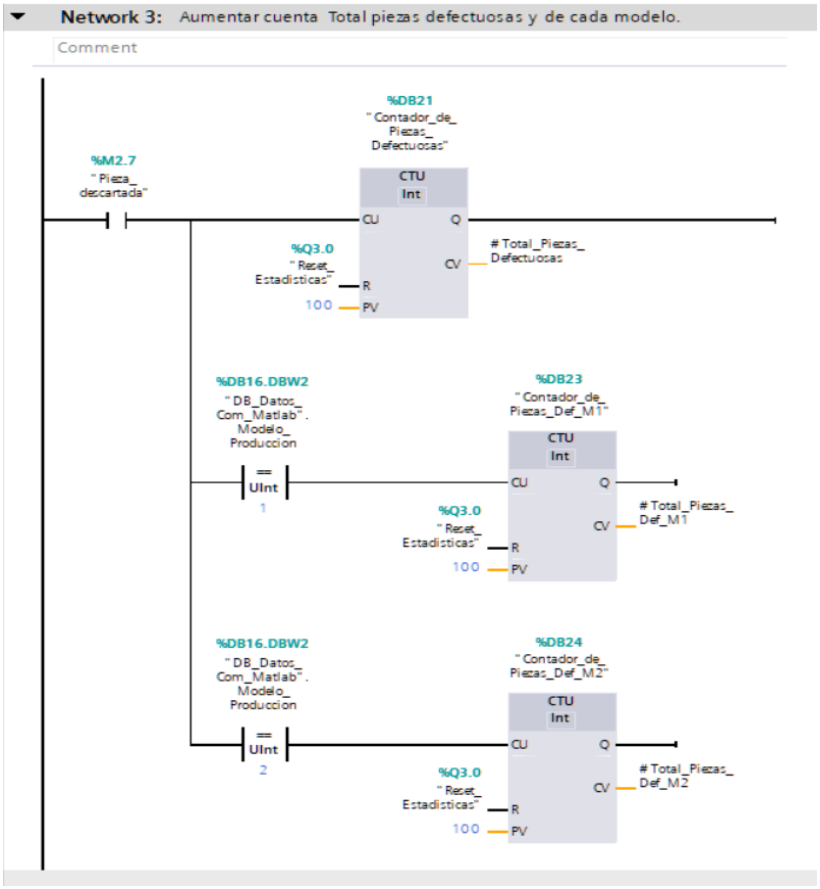
[FC7] AUX_Comunicaciones

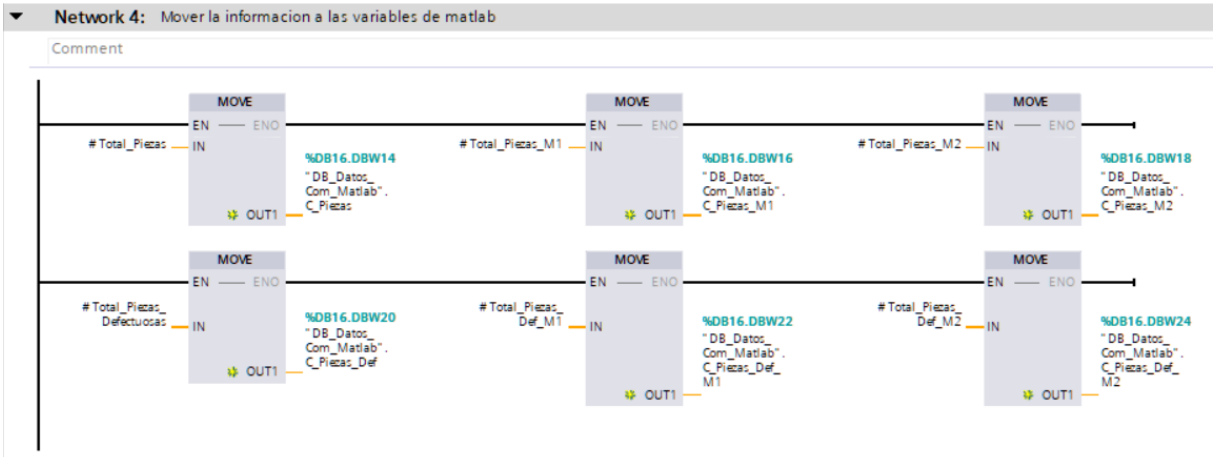


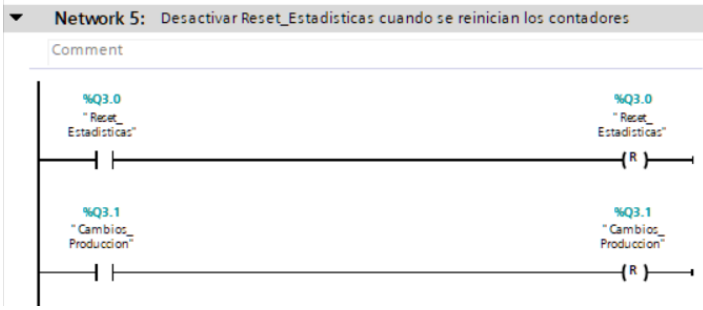

Network 4: Cuando el R2 recoge pieza de R1, se desactiva la señal de "Pieza Depositada"
Comment
%DB8.DBX1.0
"DB_Datos_Com_PLC2".P2_P1_R2_recoge_R1
%DB8.DBX0.0
"DB_Datos_Com_PLC2".P1_P2_R1_Depositada_R2
R
Network 5: Cuando se recibe orden de descartar pieza, se activa el flag Pieza defectuosa
Comment
%Q2.3
"Descartar_Pieza"
%M3.0
"Pieza_Defectuosa"
S
%Q2.3
"Descartar_Pieza"
R
Network 6: Simulación de Nueva Pieza. En un sistema real vendría dado por un sensor externo.
Comment
%DB25
"Temporizador_Nueva_Pieza"
%M2.0
"NuevaPieza"
TON
Time
IN
Q
ET
...
%MD45
"t_nueva_pieza"
PT
%M2.0
"NuevaPieza"
S

[FC11] AUX_Contador_Piezas

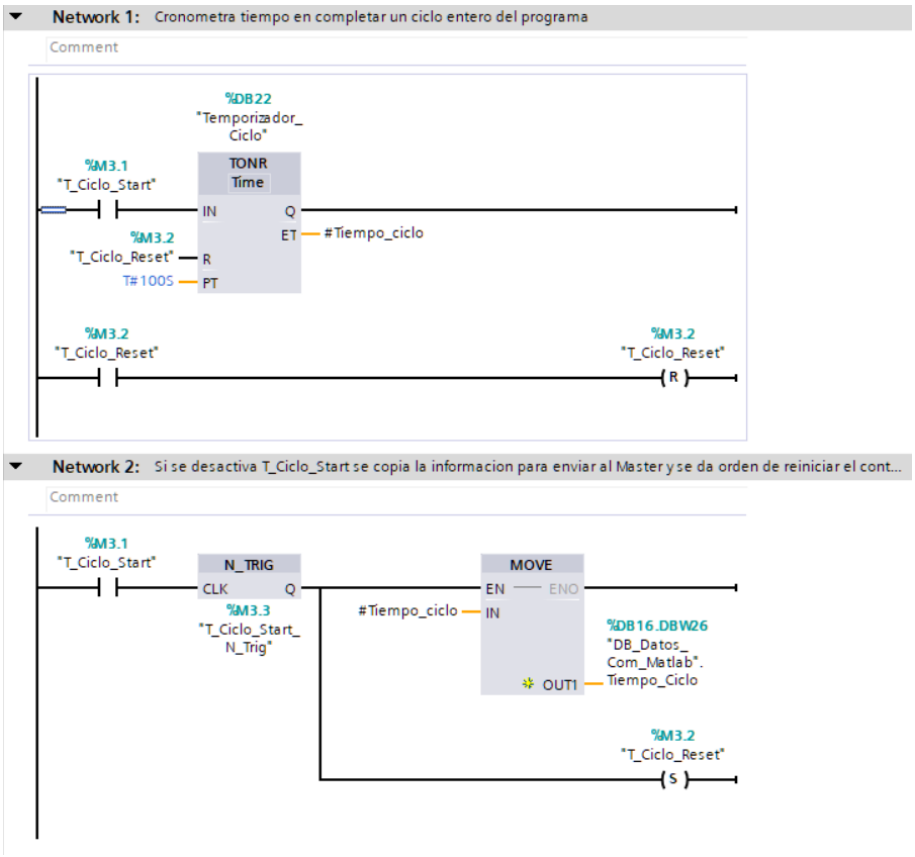



Network 3: Aumentar cuenta Total piezas defectuosas y de cada modelo.
Comment
%DB21
"Contador_de_Piezas_Defectuosas"
%M2.7
"Pieza_descartada"
CTU
Int
CU
Q
CV
R
PV
%Q3.0
"Reset_Estadisticas"
100
Total_Piezas_Defectuosas
%DB16.DBW2
"DB_Datos_Com_Matlab".Modelo_Produccion
==
UInt
1
%DB23
"Contador_de_Piezas_Def_M1"
Total_Piezas_Def_M1
2
%DB24
"Contador_de_Piezas_Def_M2"
Total_Piezas_Def_M2

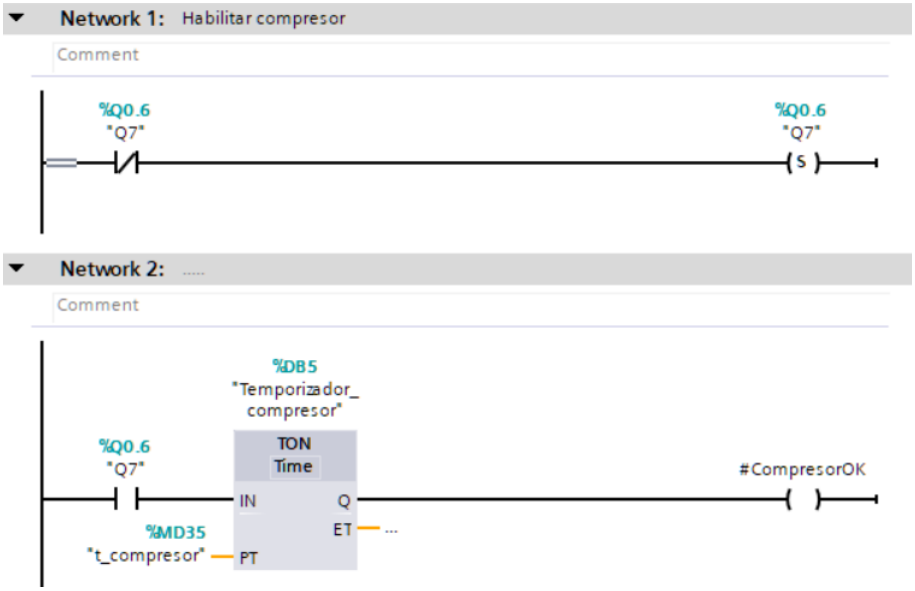

Network 4: Mover la informacion a las variables de matlab
Comment
MOVE
EN
ENO
IN
OUT1
Total_Piezas
%DB16.DBW14
"DB_Datos_Com_Matlab".C_Piezas
Total_Piezas_M1
%DB16.DBW16
"DB_Datos_Com_Matlab".C_Piezas_M1
Total_Piezas_M2
%DB16.DBW18
"DB_Datos_Com_Matlab".C_Piezas_M2
Total_Piezas_Defectuosas
%DB16.DBW20
"DB_Datos_Com_Matlab".C_Piezas_Def
Total_Piezas_Def_M1
%DB16.DBW22
"DB_Datos_Com_Matlab".C_Piezas_Def_M1
Total_Piezas_Def_M2
%DB16.DBW24
"DB_Datos_Com_Matlab".C_Piezas_Def_M2


Network 5: Desactivar Reset_Estadisticas cuando se reinician los contadores
Comment
%Q3.0
"Reset_Estadisticas"
R
%Q3.1
"Cambios_Produccion"

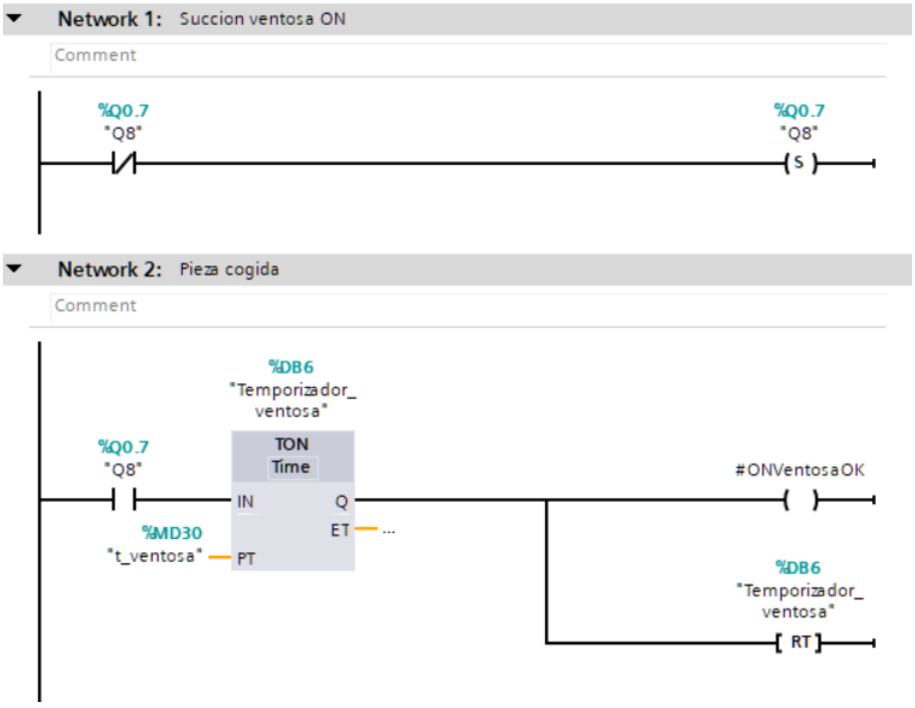
[FC18] AUX_Tiempos_Ciclo



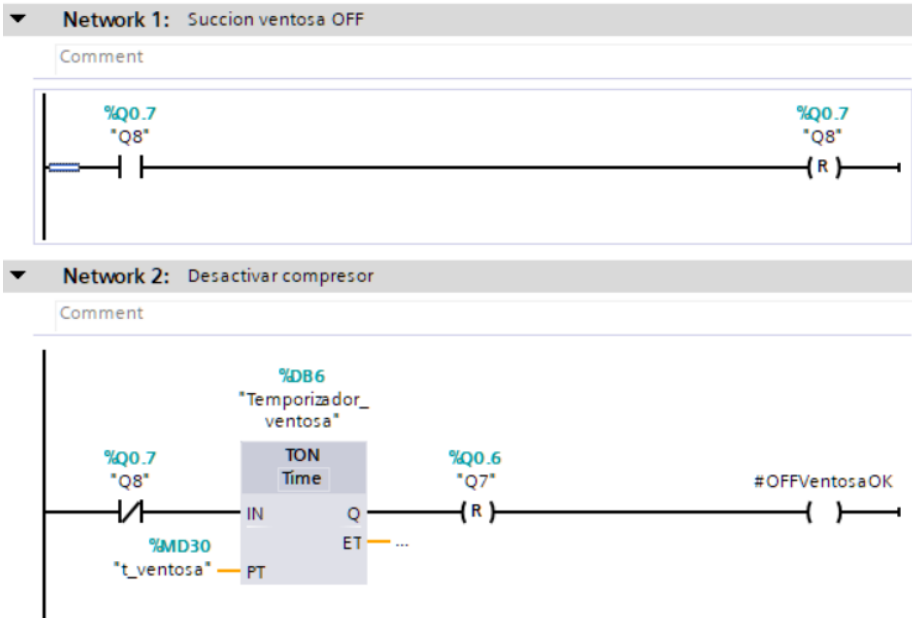
[FC17] INS_Activar_Compresor



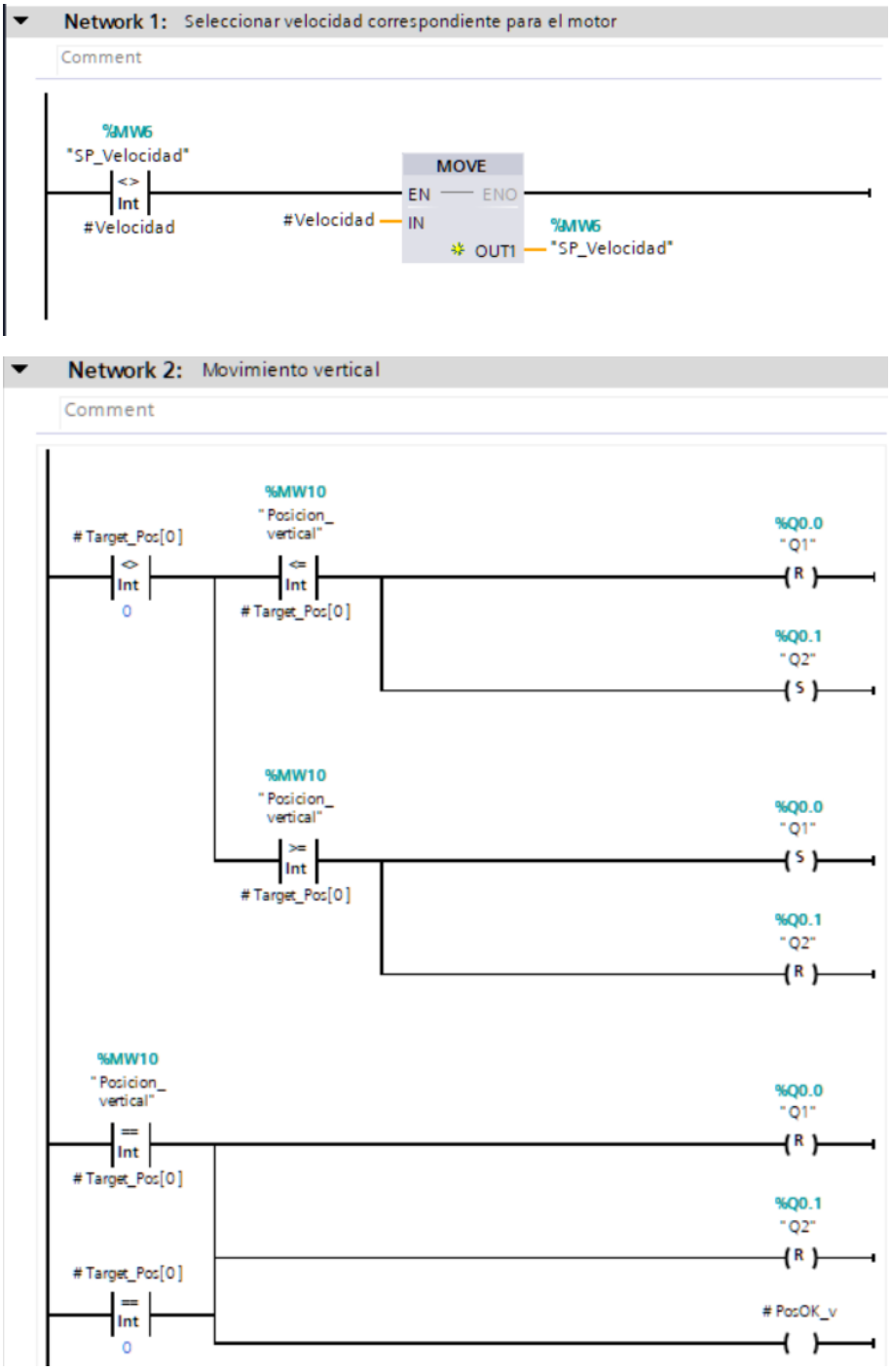
[FC5] INS_Coger_Pieza



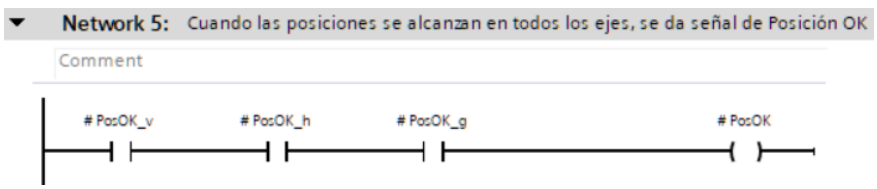
[FC6] INS_Dejar_Pieza



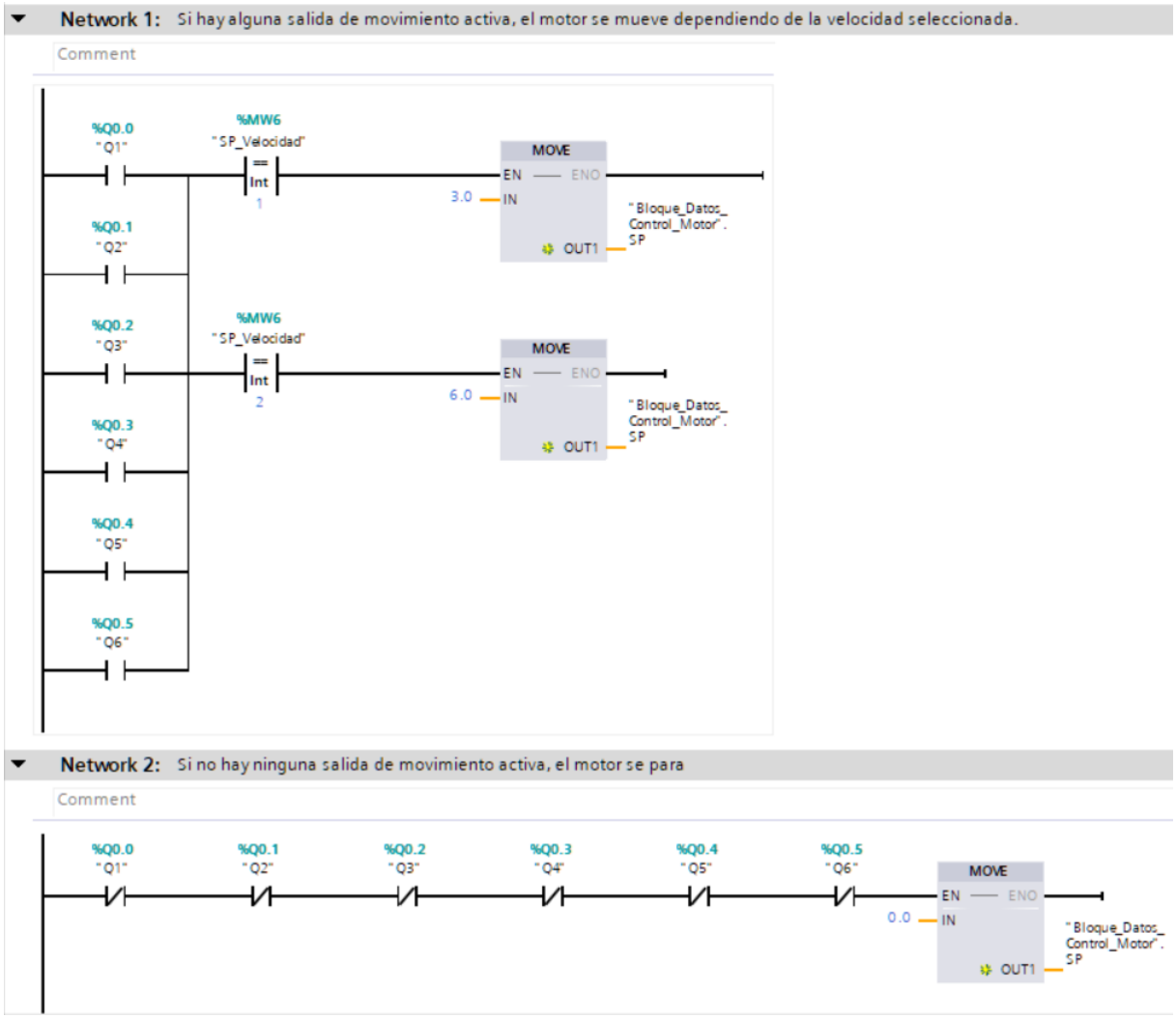
[FC3] INS_Movimiento_Robot



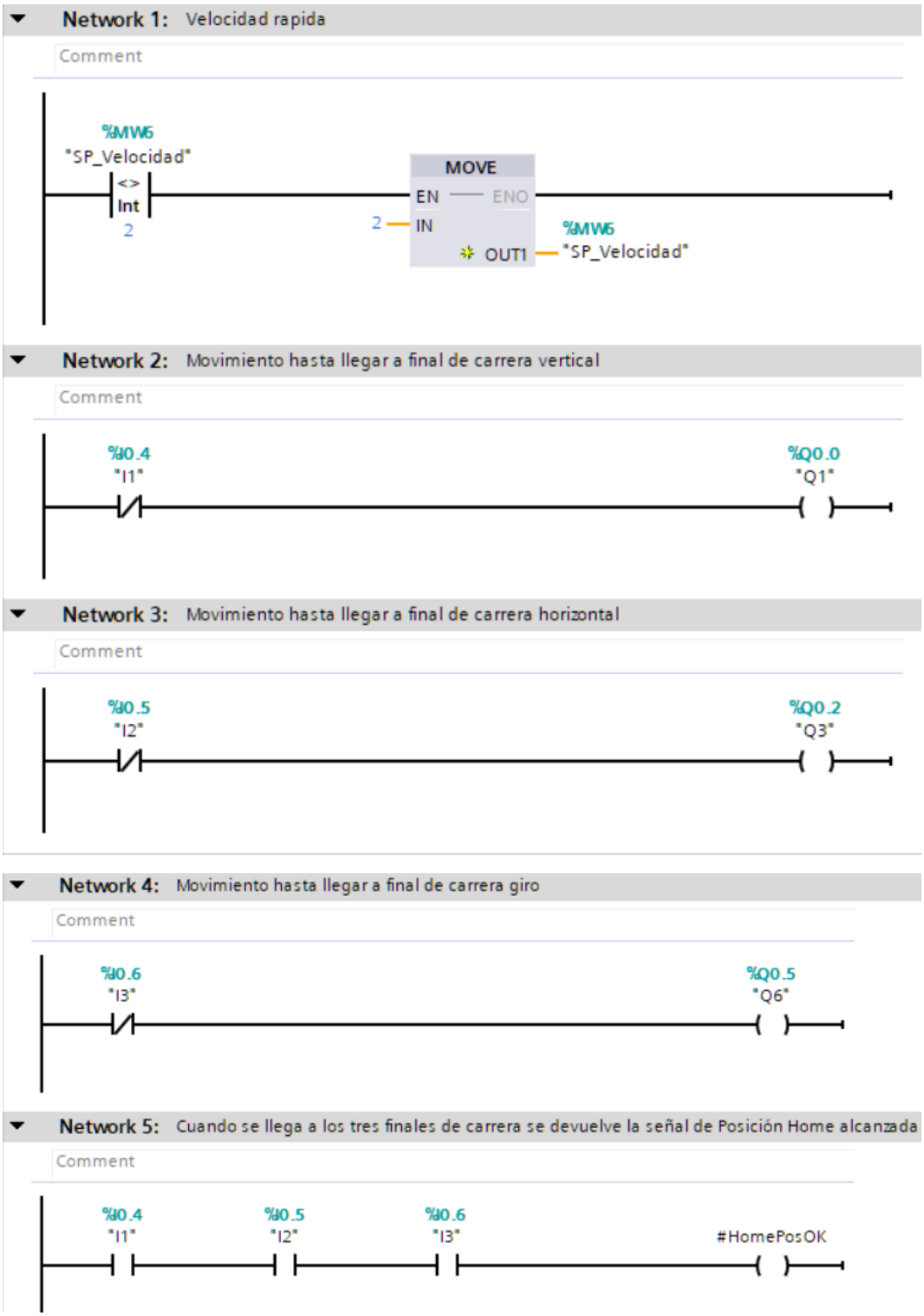
Los segmentos de movimiento horizontal y giro son iguales, pero cambiando las variables correspondientes.



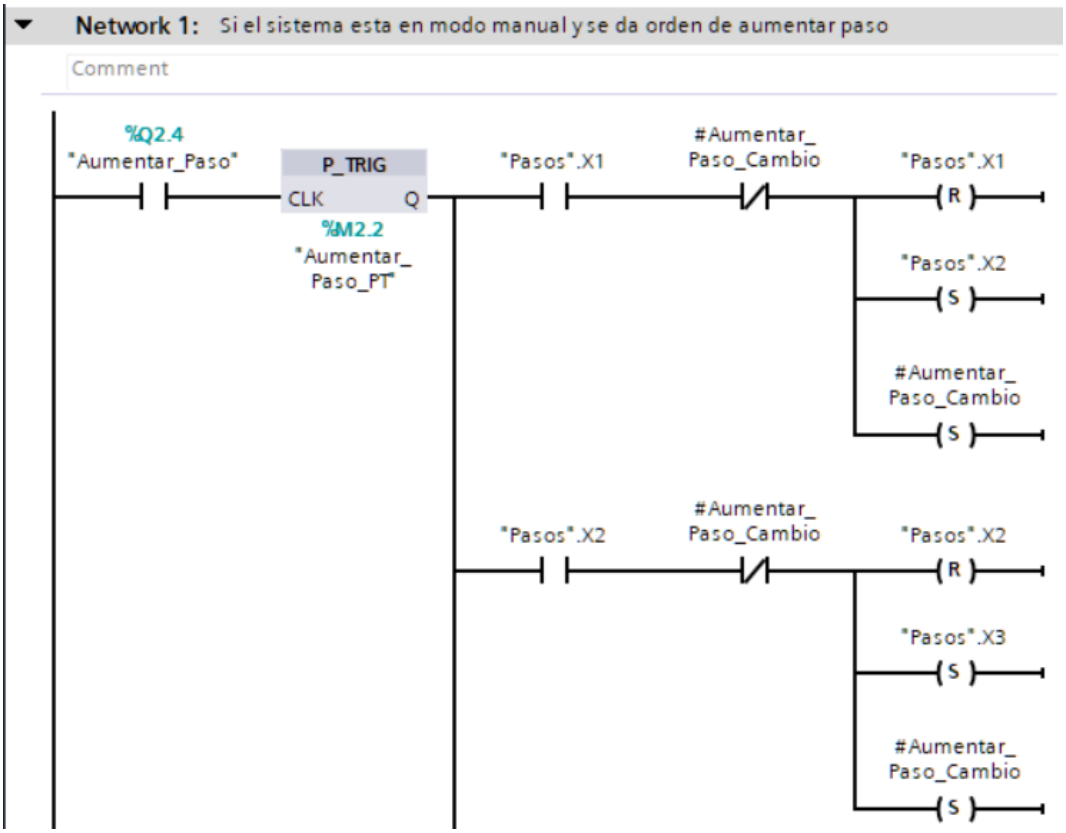
[FC19] AUX_SetPoint_Motor



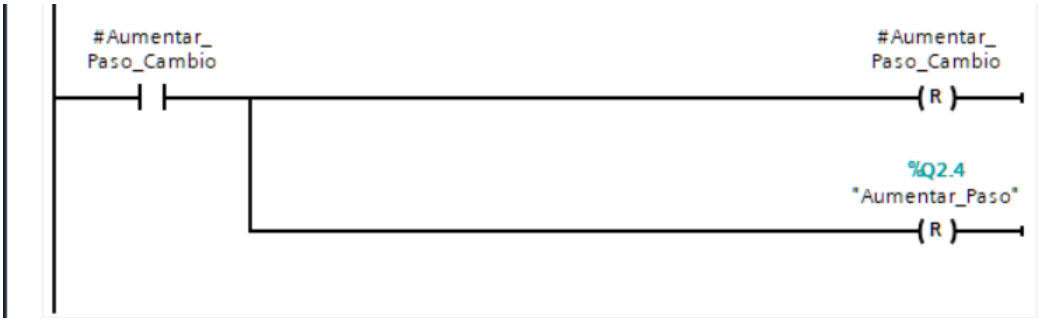
[FC4] INS_Posicion_Home

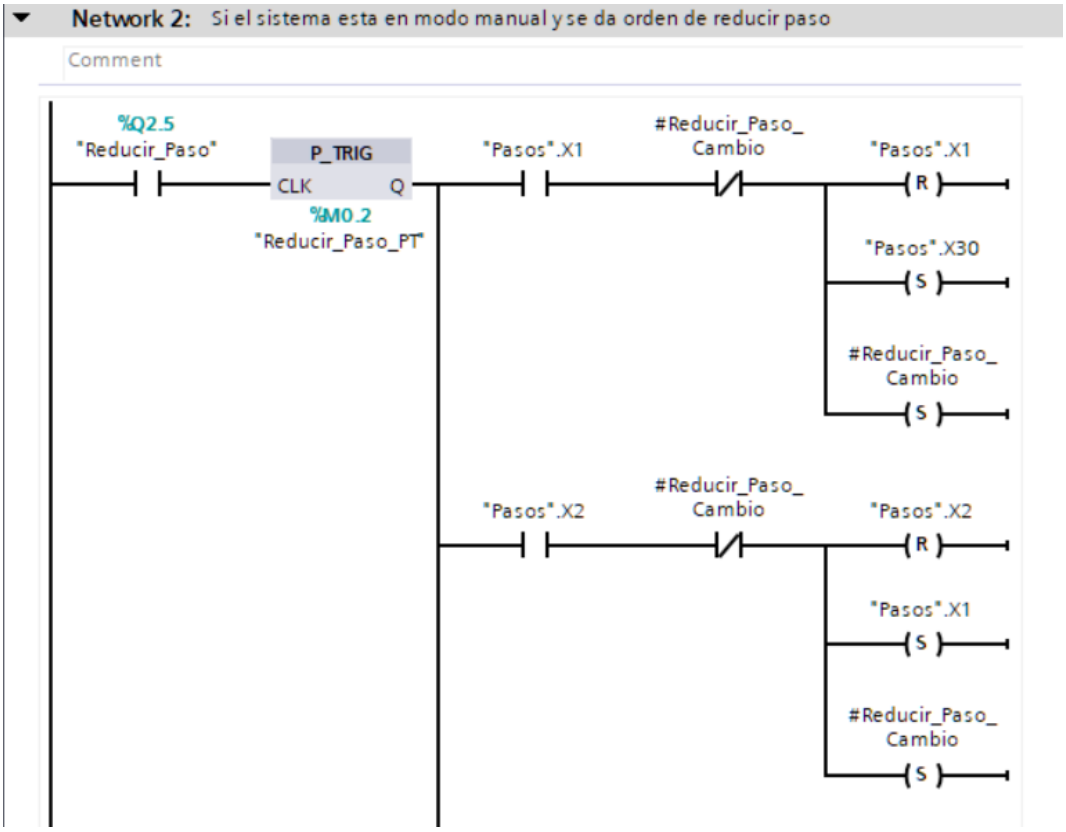


[FC9] MAN_Cambiar_Paso

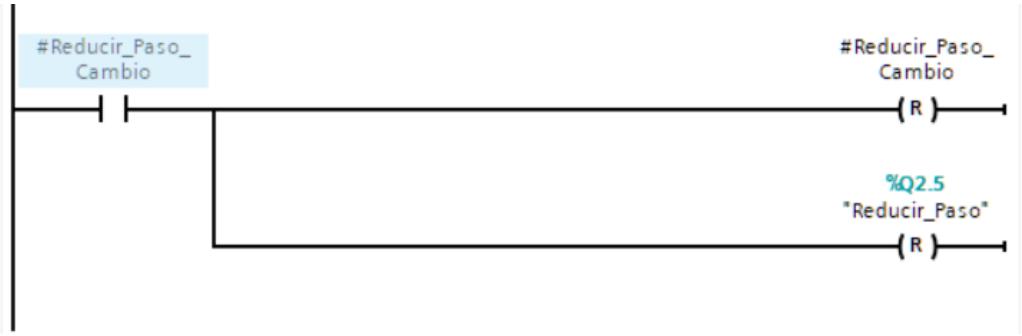


…Continúa para cada paso…

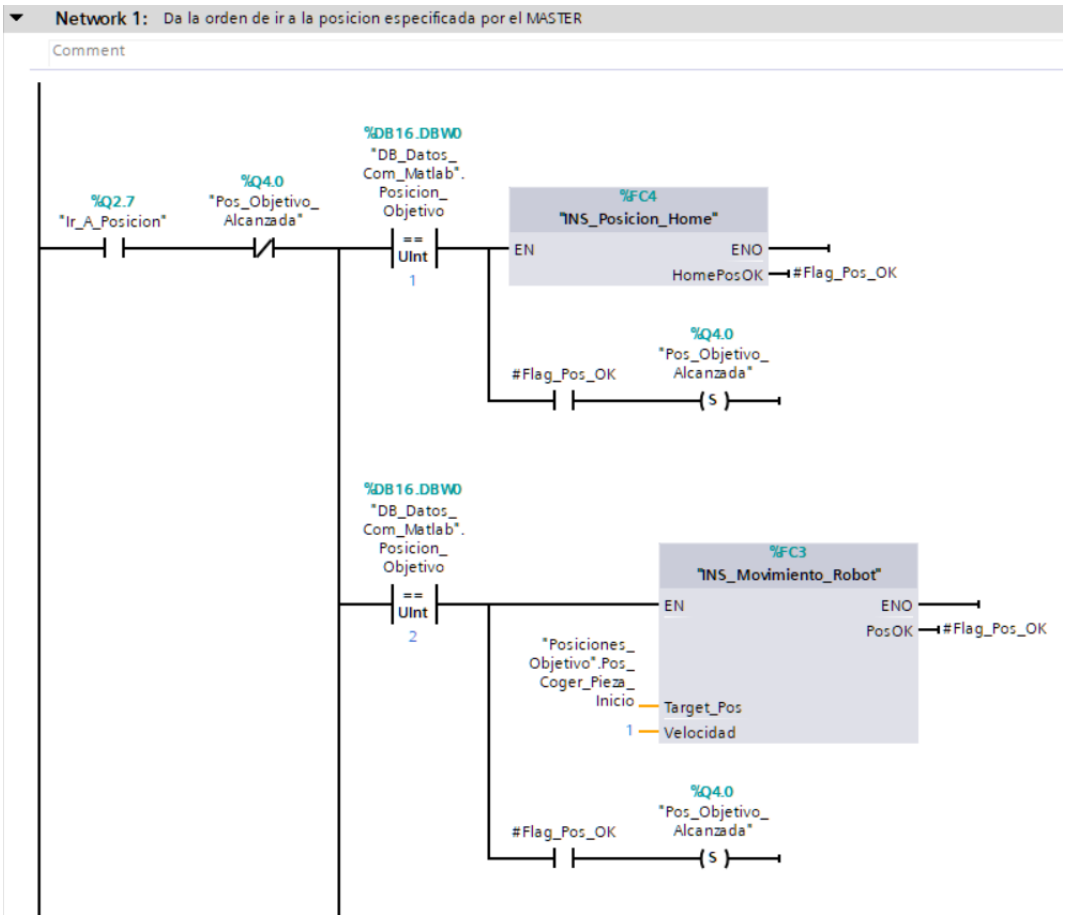




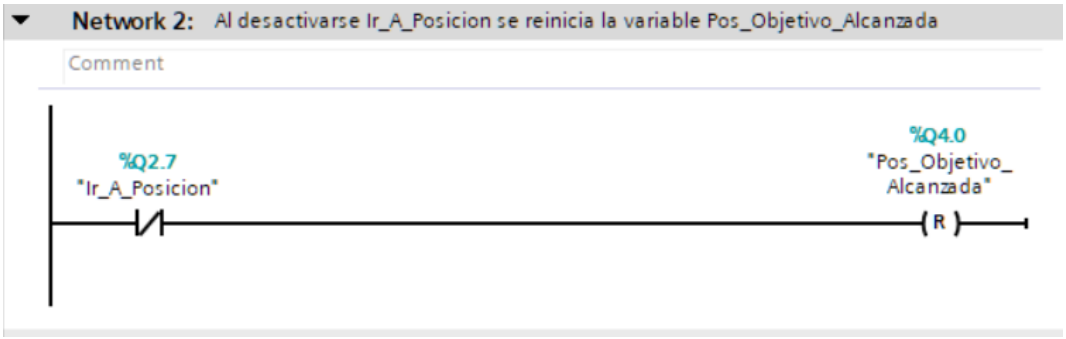
…Continúa para cada paso…



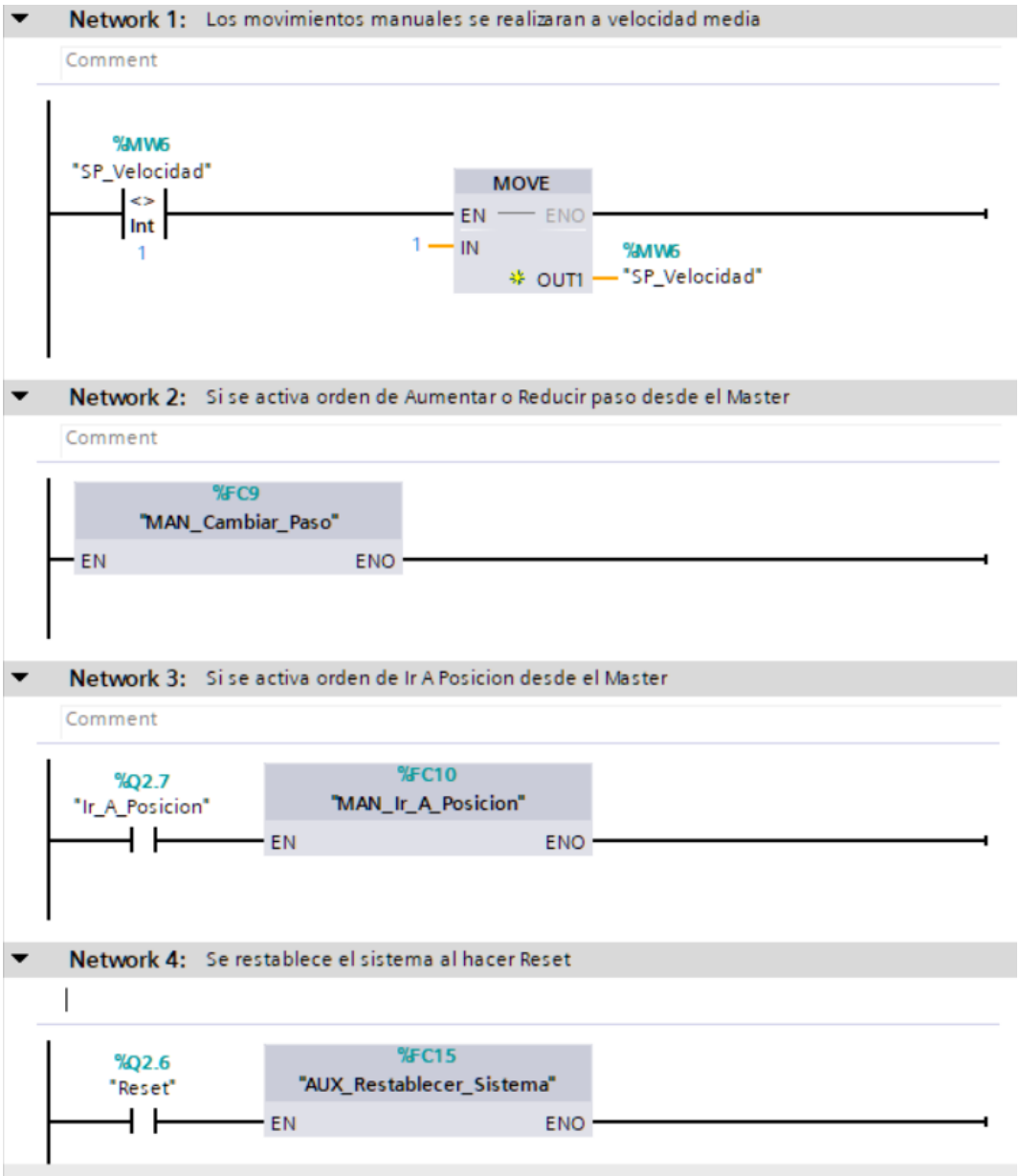
[FC10] MAN_Ir_A_Posicion



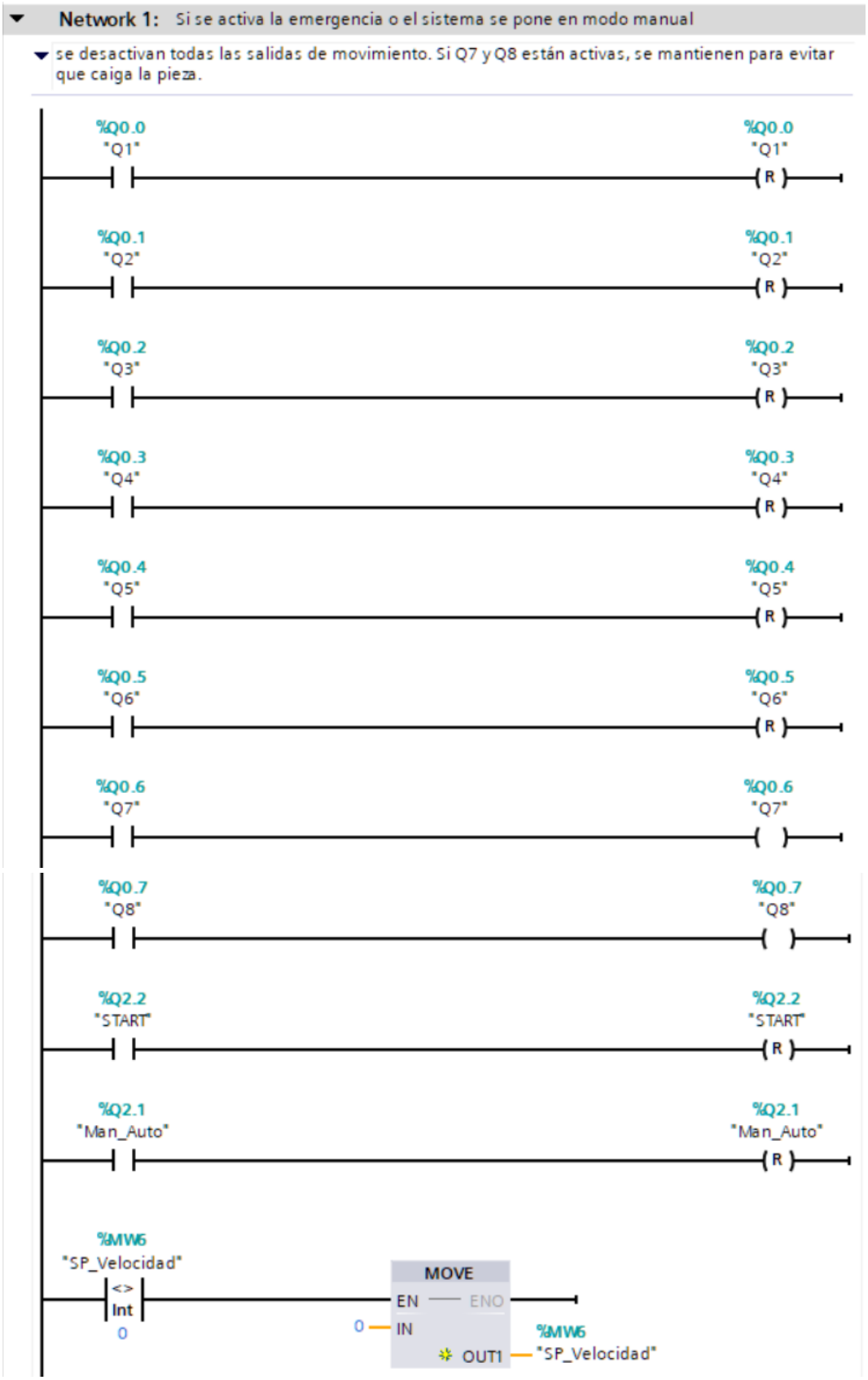
…Continúa para cada posición…



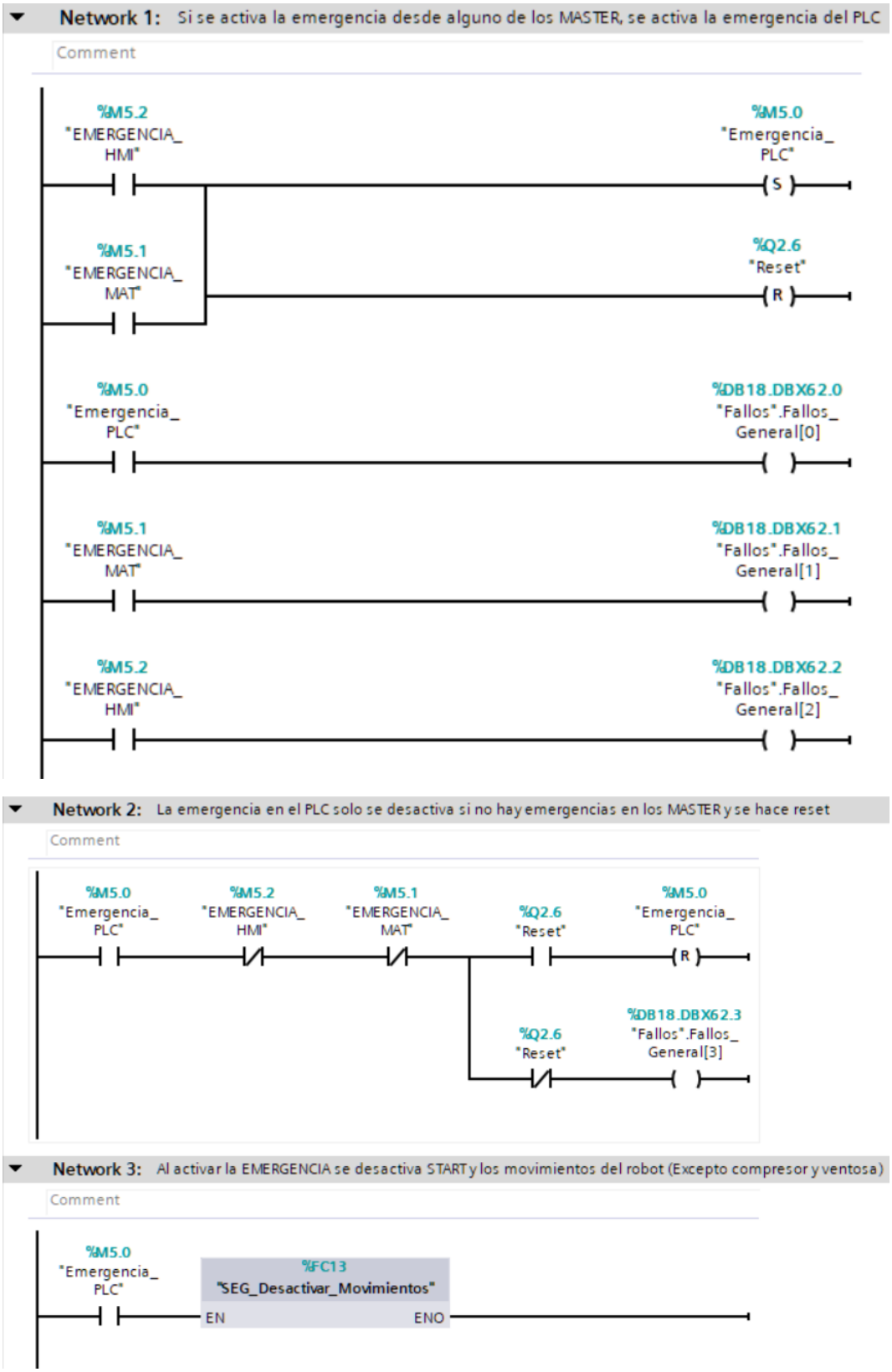
[FC12] MAN_Modo_Manual

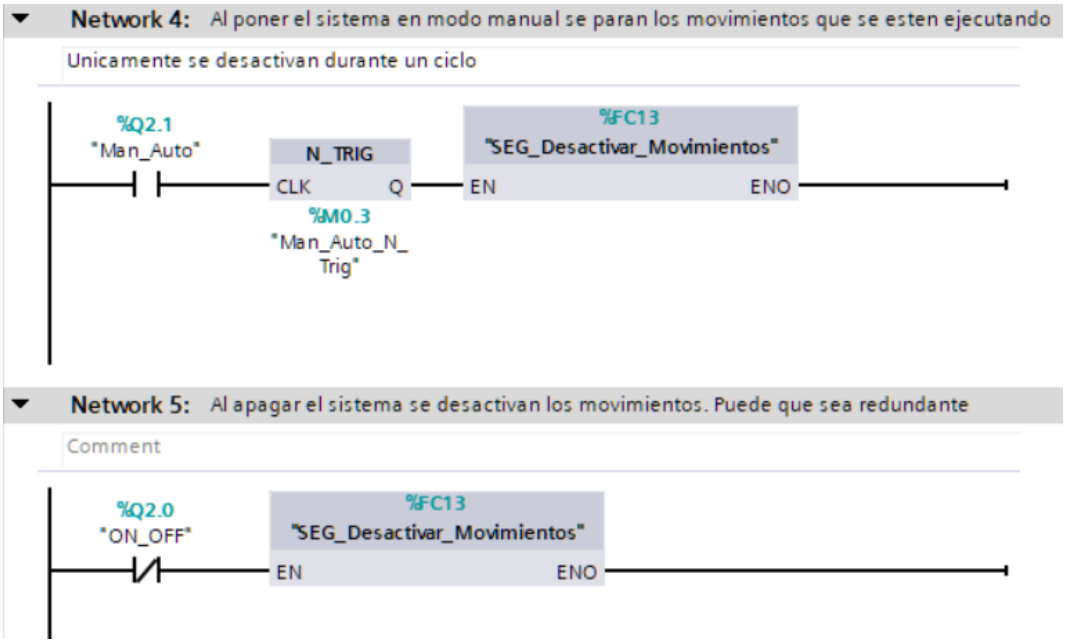


[FC13] SEG_Desactivar_Movimientos

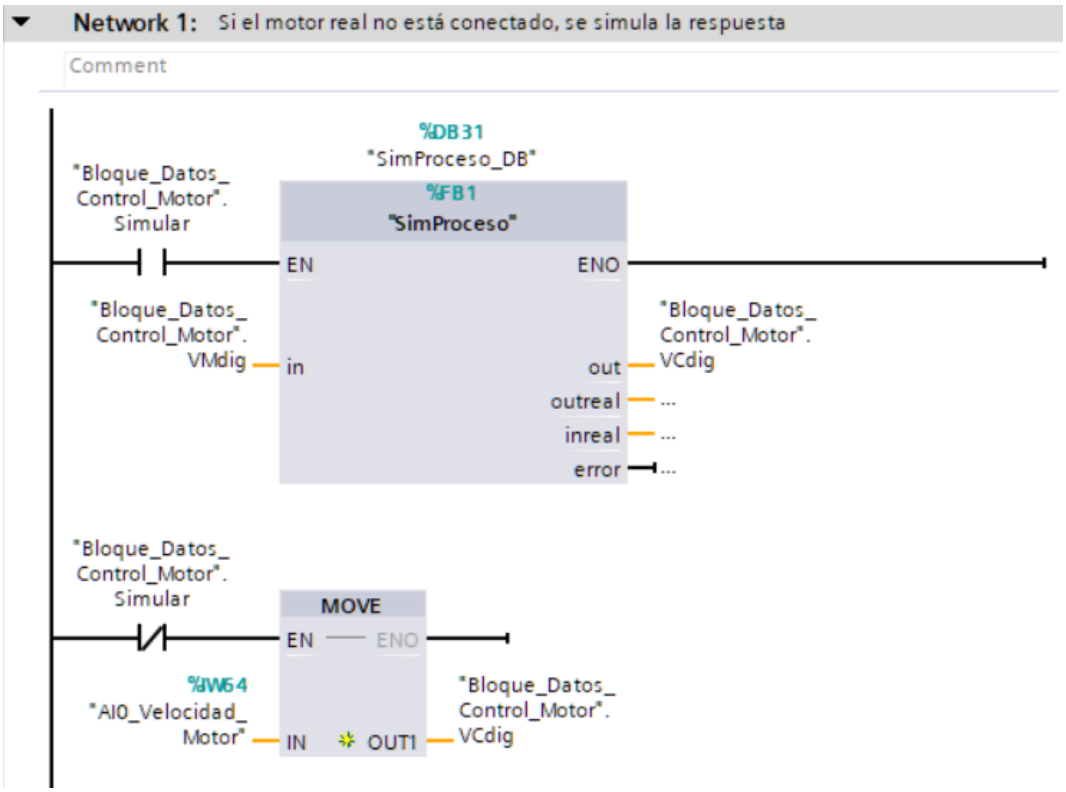


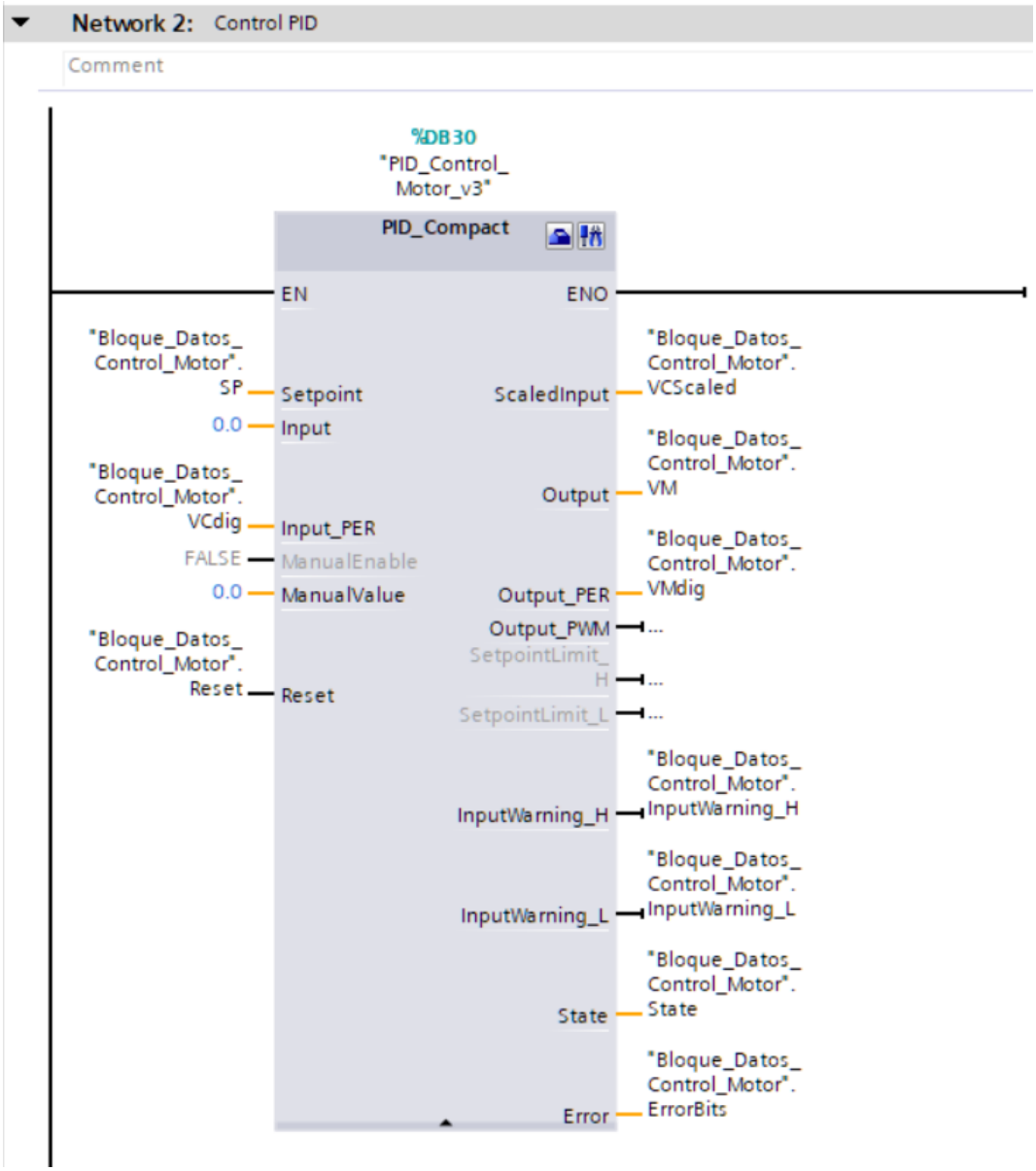
[FC14] SEG_Mecanismos_Seguridad

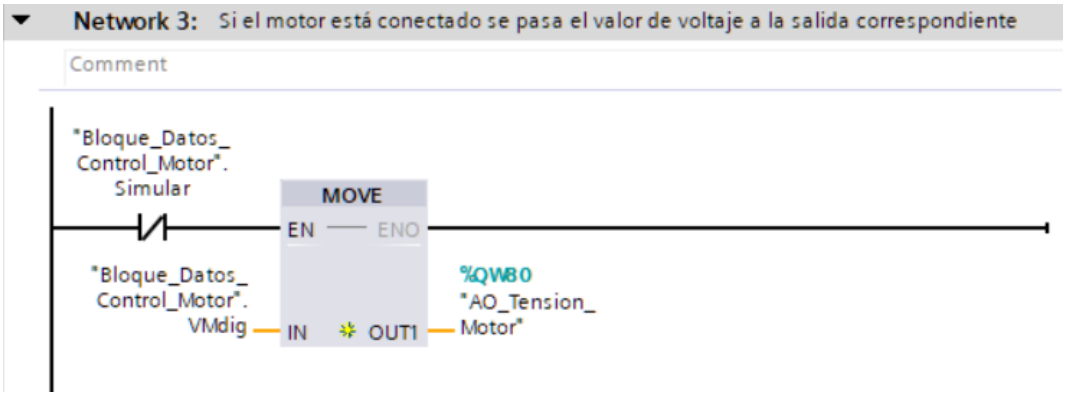




[OB30] Control_Motor




Network 2: Control PID
Comment
%DB30
"PID_Control_Motor_v3"
PID_Compact
EN
ENO
"Bloque_Datos_Control_Motor".SP
Setpoint
ScaledInput
"Bloque_Datos_Control_Motor".VCScaled
0.0
Input
"Bloque_Datos_Control_Motor".VCdig
Input_PER
Output
"Bloque_Datos_Control_Motor".VM
FALSE
ManualEnable
0.0
ManualValue
Output_PER
"Bloque_Datos_Control_Motor".VMdig
Output_PWM
"Bloque_Datos_Control_Motor".Reset
Reset
SetpointLimit_H
SetpointLimit_L
InputWarning_H
"Bloque_Datos_Control_Motor".InputWarning_H
InputWarning_L
"Bloque_Datos_Control_Motor".InputWarning_L
State
"Bloque_Datos_Control_Motor".State
Error
"Bloque_Datos_Control_Motor".ErrorBits


Network 3: Si el motor está conectado se pasa el valor de voltaje a la salida correspondiente
Comment
"Bloque_Datos_Control_Motor".Simular
MOVE
EN
ENO
"Bloque_Datos_Control_Motor".VMdig
IN
OUT1
%QW80
"AO_Tension_Motor"

Anexo 2: Programación SCADA

Estructura de control: Clase PLC

```
PLC %(i)                    % Numero de PLC
 % Qr(j)                     % Vector con valores Q leidos del PLC
 % I(j)                      % Vector con valores I leidos del PLC
 % Emergencias               % Vector con Emergencias leidas del PLC
 % Ecoder_V                  % Valor encoder vertical
 % Encoder_H                 % Valor encoder horizontal
 % Encoder_G                 % Valor encoder giro
 % Paso                      % Paso actual del PLC
 % C_Piezas                  % Piezas completadas
 % C_Piezas_M1               % Piezas completadas modelo 1
 % C_Piezas_M2               % Piezas completadas modelo 2
 % C_Piezas_Def              % Piezas defectuosas
 % C_Piezas_Def_M1           % Piezas defectuosas modelo 1
 % C_Piezas_Def_M2           % Piezas defectuosas modelo 2
 % Tiempo_Ciclo              % Tiempo en completar el ultimo proceso.
 % Vector_Tiempos            % Registro de tiempos
 % Modelo                    % Modelo en produccion
 % Pos_Objetivo_Alcanzada    % Posicion objetivo ha sido alcanzada
 % Fallos_Paso               % Fallos del paso actual en PLC
 % Fallos_General            % Fallos generales en PLC
 % Fallos_Com                % Fallos del comunicacion en PLC
 % Dispositivo               % Configuracion de comunicaciones con el PLC
 % MB_Con                    % Conexion Modbus con PLC
 % Conectado                 % Conexion exitosa con el PLC
 % Bit_Vida                  % Bit cambia de estado de manera periodica
 % Temp_Bit_Vida             % Temporizador para monitorizar bit de vida
 % Vector_Pasos              % Vector columna con el texto de cada paso
 % Vector_Fallos             % Matriz con el texto de cada fallo
```

Funciones auxiliares

```
        function Lampara_Error(app)
            % Enciende durante cinco segundos la lampara de error para avisar
            % al usuario

            if app.Error == true % Si la lampara esta encendida, la apaga y para el
temporizador


                app.ErrorLamp.Color = app.Gris;
                app.Error = false;
                stop(app.Temp_Error);

            elseif app.Error == false % Si la lampara esta apagada, la enciende y inicia el
temporizador

                app.ErrorLamp.Color = 'r';
                app.Error = true;
                start(app.Temp_Error);

            end


        end
```

```
        function Gestion_Emergencia(app)
        % Gestiona el estado de emergencia y avisa si hay alguna
        % emergencia activada.
        Emergencia_HMI = false;
        Emergencia_plc = false;
        suma_emergencias = 0;
            for i = 1:app.Total_PLCs


                suma_emergencias = suma_emergencias + sum(app.PLC(i).Emergencias) +
app.Emergencia;

            end


                if suma_emergencias == 0 % Si no hay emergencias activadas


                    app.EmergenciaLamp.Color = app.Gris; % Lampara emergencia en gris
                    app.Log.BackgroundColor = 'w'; % Log en blanco
                    app.Emergencia_PCLabel.BackgroundColor = 'g'; % Indicadores sistemas en
verde
                    app.Lamps_Emergencia_PLC(1).BackgroundColor = 'g';
                    app.Lamps_Emergencia_PLC(2).BackgroundColor = 'g';
                    app.Emergencia_HMILabel.BackgroundColor = 'g';
```

```
                else % Si hay emergencias
                    app.EmergenciaLamp.Color = 'r'; % Lampara emergencia en rojo
                    app.Log.BackgroundColor = 'r'; % Log en rojo

                    % Desactivar START
                    app.Start = false;
                    app.StartLamp.Color = app.Gris;

                    for plc = 1:app.Total_PLCs

                        if app.PLC(plc).Emergencias(1)% Si hay emergencia en el PLC

                            texto = sprintf('EMERGENCIA PLC%d ACTIVADA',plc);
                            Actualizar_Log(app,texto);

                            app.Lamps_Emergencia_PLC(plc).BackgroundColor = 'r';

                            Emergencia_plc = true;

                        elseif not(app.PLC(plc).Emergencias(1)) && Emergencia_plc == false
% Si no hay emergencia en el PLC

                            app.Lamps_Emergencia_PLC(plc).BackgroundColor = 'g';

                        end

                        if app.PLC(plc).Emergencias(3) % Si hay emergencia en el HMI

                            texto = sprintf('EMERGENCIA HMI ACTIVADA');
                            Actualizar_Log(app,texto);
                            app.Emergencia_HMILabel.BackgroundColor = 'r';
                            Emergencia_HMI = true;

                        elseif not(app.PLC(plc).Emergencias(3)) && Emergencia_HMI == false
% Si no hay emergencia en el HMI

                            app.Emergencia_HMILabel.BackgroundColor = 'g';
                        end
                    end
                end
            end
```

```
        function Conectar_PLC(app, plc, conectar, IP)
% Funcion para conectar/desconectar con el PLC designado
        if conectar
                try
                        % Crear dispositivo para conectar y conexión a dicho dispositivo
                        app.PLC(plc).Dispositivo = SiemensDevice(IPDeviceEndPoint(IP),
SiemensDeviceType.S71200);
                        app.PLC(plc).MB_Con = app.PLC(plc).Dispositivo.CreateConnection();
                        app.PLC(plc).MB_Con.Open();
```

```
                % Lectura para comprobar conexion exitosa
                app.PLC(plc).MB_Con.ReadBoolean(app.Direccion_Leer_Q, 1);

                app.PLC(plc).Conectado = true;

                % Mensaje en Log de conexion exitosa
                texto = sprintf('Conexion con PLC%d exitosa',plc);
                Actualizar_Log(app,texto);

            catch
                app.PLC(plc).Conectado = false;

                % Mensaje en Log de fallo de conexion
                texto = sprintf('Fallo de conexion con PLC%d',plc);
                Actualizar_Log(app,texto);
                Lampara_Error(app);


        end

        if app.PLC(plc).Conectado

            app.Lamparas_ON(plc).Color = 'g';

        else
            app.Lamparas_ON(plc).Color = 'r';
        end

        pause(0.1)

    else

        try
            stop(app.PLC(plc).Temp_Bit_Vida)
        catch
        end
        try
            %clear(app.PLC(i).MB_Con)
            app.PLC(plc).MB_Con.Close()
            app.PLC(plc).Conectado = false;
        catch
        end

            app.Lamparas_ON(plc).Color = 'r';


            % Mensaje en Log de PLC desconectado
            texto = sprintf('PLC%d desconectado', plc);
            Actualizar_Log(app,texto);
```

```
            end

        end
```

```
        function Bits_de_Vida(app, plc)
            %Hace parpadear la lampara de los PLC conectados para asegurar
            %que la conexion sigue activa. Si no es así se avisa al
            %usuario.
            % stop timer
            try
                stop(app.PLC(plc).Temp_Bit_Vida)
            catch
            end

            if app.PLC(plc).Bit_Vida == true

                app.Lamparas_ON(plc).Color = 'g';

            else

                app.Lamparas_ON(plc).Color = app.Gris;

            end

            start(app.PLC(plc).Temp_Bit_Vida)

        end
```

```
        function No_Bit_Vida(app, plc)
            % Si se deja de recibir el Bit de vida, la lampara del PLC se
            % vuelve azul fijo, para informar al usuario.

            app.Lamparas_ON(plc).Color = 'b';
            % Mensaje en Log de conexion perdida con PLC
            texto = sprintf('Conexion perdida con PLC%d', plc);
            Actualizar_Log(app,texto);

        end
```

```
        function  Escribir_PLC(app, objetivo, direccion, tipo,  datos) %Escribe datos en el
PLC objetivo.
            % Uso de la funcion "write" con las conexiones creadas al
            % inicio de la app


                for i = 1:app.Total_PLCs

                    if app.PLC(i).Conectado && (objetivo == 0 || objetivo == i)
                        % Si el PLC(i) esta conectado y es objetivo
                        try
                            switch tipo
                                case "bool"
```

```
                                app.PLC(i).MB_Con.WriteBoolean(direccion, datos);

                                case "int16"

                                app.PLC(i).MB_Con.WriteInt16(direccion, datos)

                            end

                        catch

                            % Mensaje en Log de fallo de escritura
                            texto = sprintf('Fallo de escritura  PLC%d, direccion %s',i,
direccion);
                            Actualizar_Log(app,texto);
                            Lampara_Error(app);

                        end
                    end
                end


            %fin fct Escribir_PLC
         end
```

```
        function Escribir_Estados(app)
            % Escribir datos de estado en las Salidas de ambos PLC

            Escribir_PLC(app, 0, app.Direccion_Escribir_Estados,"bool", [app.ON_OFF,
app.Man_Auto, app.Start] );
            Escribir_PLC(app, 0, app.Direccion_Escribir_Emergencia, "bool",
app.Emergencia);

        End
```

```
        function Escribir_Manual(app, objetivo)
            % Escribir ordenes manuales en las Salidas del PLC seleccionado

            Escribir_PLC(app, objetivo, app.Direccion_Escribir_Manual, "bool",
[app.Pieza_Defectuosa, app.Aumentar_Paso, app.Reducir_Paso, app.Reset, app.Ir_A_Posicion]);

            % Tras enviar se reinician los valores de Aumentar,reducir
            % paso y Reset

            app.Aumentar_Paso = false;
            app.Reducir_Paso = false;
            app.Reset = false;
            app.Pieza_Defectuosa = false;

        end
```

```
        function Escribir_Q(app, objetivo, Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8)
            % Escribir ordenes de control manual de movimientos en las
            % Salidas del PLC seleccionado

            Escribir_PLC(app, objetivo, app.Direccion_Escribir_Q, "bool", [Q1, Q2, Q3, Q4,
Q5, Q6, Q7, Q8])

        end
```

```
        function Escribir_Posicion(app, objetivo)
            % Escribir posicion seleccionada en la memoria del PLC seleccionado


            % Asignar numero segun posicion seleccionada
            switch app.PosicionDropDown.Value

                case "HOME"

                    posicion = 1;

                case "Coger Inicio"

                    posicion = 2;

                case "Modelo 1"

                    posicion = 3;

                case "Modelo 2"

                    posicion = 4;

                case "Final proceso"

                    posicion = 5;

                case "Espera M1"

                    posicion = 6;

                case "Espera M2"

                    posicion = 7;

                case "Descartar Pieza"
                    if app.P_Robot_Sel == 1
                        posicion = 8;
                    else
                        posicion = 5;
                    end
```

```
            end

            Escribir_PLC(app, objetivo, app.Direccion_Escribir_Posicion, "int16",
posicion);
        end
        function Escribir_Produccion(app)

            Escribir_PLC(app, 0, app.Direccion_Escribir_Produccion, "int16",
[app.Modelo_Produccion, app.Objetivo_Produccion])
            Escribir_PLC(app, 0, 'Q3.1', "bool", 1)

        end
        function Actualizar_Tab_Principal(app)
            % Actualiza la pantalla principal con la informacion leida del
            % PLC segun el Robot seleccionado

            app.Paso.Text = int2str(app.PLC(app.P_Robot_Sel).Paso);

            app.DescripcionPaso.Value =
app.PLC(app.P_Robot_Sel).Vector_Pasos(app.PLC(app.P_Robot_Sel).Paso, :);


            % Actualizar Logs Fallos

            % Fallos Paso
            j = 1;
            Texto = ["", "","",""];
            for i = 1:length(app.PLC(app.P_Robot_Sel).Fallos_Paso)


                if app.PLC(app.P_Robot_Sel).Fallos_Paso(i)

                  Texto(j,1) =
app.PLC(app.P_Robot_Sel).Vector_Fallos(app.PLC(app.P_Robot_Sel).Paso, i);
                  j = j + 1;

                end
            end
            app.FallosPasoTextArea.Value = Texto;

            % Fallos Generales y de Comunicacion
            j = 1;
            Texto = ["", "","",""];
            for i = 1:length(app.PLC(app.P_Robot_Sel).Fallos_General)


                if app.PLC(app.P_Robot_Sel).Fallos_General(i)

                  Texto(j,1) = app.PLC(app.P_Robot_Sel).Vector_Fallos(app.indice_f_gen, i);
```

```
                j = j + 1;

            end
        end
        for i = 1:length(app.PLC(app.P_Robot_Sel).Fallos_Com)


            if app.PLC(app.P_Robot_Sel).Fallos_Com(i)

              Texto(j,1) = app.PLC(app.P_Robot_Sel).Vector_Fallos(app.indice_f_com, i);
              j = j + 1;

            end
        end
        app.FallosGeneralesTextArea.Value = Texto;

    end
```

```
    function colores = Asignar_Colores(app, datos)
        % Asigna colores a los datos para actualizar las lamparas

        colores = repmat(app.Gris, 1, length(datos));
        for i = 1:length(datos)

            if datos(i) == 1

                colores(i) = "g";

            else
                colores(i) = app.Gris;

            end
        end
    end
```

```
    function Actualizar_Tab_Estados(app)
        %  Actualiza la pantalla Estado Sistema con la informacion
        %  leida del PLC

        for i = 1:app.Total_PLCs

            for j = 1:app.Longitud_Q_Lectura % Colores lamparas Qs
                try
                    Colores = Asignar_Colores(app, app.PLC(i).Qr);
                catch

                end
                app.Lamparas_Q(i,j).Color = Colores(j);

            end

            for j = 1:app.Longitud_I_Lectura % Colores lamparas Is
```

```
                Colores = Asignar_Colores(app, app.PLC(i).I);
                app.Lamparas_I(i,j).Color = Colores(j);

            end

        switch  app.PLC(i).Modelo % Mostrar modelo seleccionado en los PLC

            case 1 % Modelo 1 seleccionado

                app.Vector_Modelo(i, 1).Color = 'g';
                app.Vector_Modelo(i, 2).Color = app.Gris;

            case 2 % Modelo 2 seleccionado

                app.Vector_Modelo(i, 1).Color = app.Gris;
                app.Vector_Modelo(i, 2).Color = 'g';
        end

            % Actualizar informacion encoders
            app.Vector_Encoders(i,1).Text = int2str(app.PLC(i).Encoder_V);
            app.Vector_Encoders(i,2).Text = int2str(app.PLC(i).Encoder_H);
            app.Vector_Encoders(i,3).Text = int2str(app.PLC(i).Encoder_G);
        end
    end
```

```
    function Actualizar_Tab_CM(app)
        % Actualiza la pantalla Control Manual Movimientos con la
        % informacion leida del PLC segun el Robot seleccionado

        % Colores lamparas
        Colores = Asignar_Colores(app, app.PLC(app.CM_Robot_Sel).I);
        app.CM_I1Lamp.Color = Colores(1);
        app.CM_I2Lamp.Color = Colores(2);
        app.CM_I3Lamp.Color = Colores(3);

        Colores = Asignar_Colores(app, app.PLC(app.CM_Robot_Sel).Qr(8));
        app.CM_Q8Lamp.Color = Colores;

        % Encoders
        app.CM_LecturaEncoderV.Text = int2str(app.PLC(app.CM_Robot_Sel).Encoder_V);
        app.CM_LecturaEncoderH.Text = int2str(app.PLC(app.CM_Robot_Sel).Encoder_H);
        app.CM_LecturaEncoderG.Text = int2str(app.PLC(app.CM_Robot_Sel).Encoder_G);

    end
```

```
function Actualizar_Tab_Produccion(app)
            % Actualiza la pantallade Control de Produccion con la
            % informacion leida del PLC. Solo se llama al completar una
            % pieza
            piezas_Def = [0;0; 0];
            Promedio_Tiempo_Ciclo = [0;0];
            C_Tiempos = [0;0];

            for plc = 1:app.Total_PLCs

                if app.PLC(plc).Conectado

                    if app.PLC(plc).Vector_Tiempos == 0

                        app.PLC(plc).Vector_Tiempos = app.PLC(plc).Tiempo_Ciclo;

                    else

                        app.PLC(plc).Vector_Tiempos(length(app.PLC(plc).Vector_Tiempos) +
1) = app.PLC(plc).Tiempo_Ciclo;

                    end

                    Promedio_Tiempo_Ciclo(plc) = mean(app.PLC(plc).Vector_Tiempos, 2);
                    C_Tiempos(plc) =  app.PLC(plc).Tiempo_Ciclo;

                    piezas_Def(1) = piezas_Def(1) + app.PLC(plc).C_Piezas_Def_M1;
                    piezas_Def(2) = piezas_Def(2) + app.PLC(plc).C_Piezas_Def_M2;
                end
            end




            % ACTUALIZAR TABLA PIEZAS


            piezas =
[app.PLC(app.PLC_Datos_Produccion).C_Piezas_M1;app.PLC(1).C_Piezas_M1];
            piezas(3) = piezas(1) + piezas(2);

            piezas_Def(3) = piezas_Def(1) + piezas_Def(2);

            datos = table(app.T_columna_Modelos, piezas, piezas_Def);

            app.ProduccionPiezasTable.Data = datos;


%             ACTUALIZAR TABLA TIEMPOS DE CICLO
```

```
            datos = table(app.T_columna_Robots, C_Tiempos/1000,
Promedio_Tiempo_Ciclo/1000);

            app.ProduccionTiemposTable.Data = datos;




        end
```

```
        function  Comprobar_Pos_Alcanzada(app)
            % Comprueba si se ha alzanzado la posicion objetivo. Si es asi
            % se desactiva la orden Ir_A_Posicion y se comunica al usuario

            if app.PLC(app.P_Robot_Sel).Pos_Objetivo_Alcanzada == true

                app.Ir_A_Posicion = false;
                app.IrButton.Value = false;
                Escribir_Manual(app, app.P_Robot_Sel);

                % Mensaje en Log de posicion alcanzada
                texto = sprintf(' Robot %d posicion objetivo alcanzada' ,app.P_Robot_Sel);
                Actualizar_Log(app,texto);

            end
        end
```

```
function Leer_PLC(app)
% Uso de la funcion "read" con las conexiones creadas al
% inicio de la app. De manera periodica
Piezas_antes_lectura = [0,0];
Piezas_Def_antes_lectura = [0,0];


for i = 1:app.Total_PLCs % Para todos los PLC
        if app.PLC(i). Conectado % Si el PLC i esta conectado
        try % Leer Qs
        app.PLC(i).Qr = app.PLC(i).MB_Con.ReadBoolean(app.Direccion_Leer_Q,
        app.Longitud_Q_Lectura);
        catch
        texto = sprintf('Error lectura Qs PLC%d' ,i);
        Actualizar_Log(app,texto);
        Lampara_Error(app);


        end
        % Leer Is
        try
        app.PLC(i).I = app.PLC(i).MB_Con.ReadBoolean(app.Direccion_Leer_I,
        app.Longitud_I_Lectura);
        catch
```

```
texto = sprintf('Error lectura Is PLC%d' ,i);
Actualizar_Log(app,texto);
Lampara_Error(app);


end
% Leer Emergencias
try
app.PLC(i).Emergencias =
app.PLC(i).MB_Con.ReadBoolean(app.Direccion_Leer_Emergencia,
app.Longitud_Emergencia_Lectura);


catch
texto = sprintf('Error lectura Emergencias PLC%d' ,i);
Actualizar_Log(app,texto);
Lampara_Error(app);
end
% Leer Bits de Vida
try
estado_bit_anterior = app.PLC(i).Bit_Vida;
app.PLC(i).Bit_Vida = app.PLC(i).MB_Con.ReadBoolean(app.Direccion_Leer_Bit_Vida,
app.Longitud_Bit_Vida);
if estado_bit_anterior ~= app.PLC(i).Bit_Vida
Bits_de_Vida(app, i)
end
catch
texto = sprintf('Error lectura Bit Vida PLC%d' ,i);
Actualizar_Log(app,texto);
Lampara_Error(app);
end
if app.Ir_A_Posicion == true % Comprobar que el robot ha llegado a la posicion
seleccionada si esta en modo Ir A Posicion
try
app.PLC(i).Pos_Objetivo_Alcanzada =
app.PLC(i).MB_Con.ReadBoolean(app.Direccion_Leer_Pos_Objetivo_Alcanzada, 1);
Comprobar_Pos_Alcanzada(app);
catch
texto = sprintf('Error lectura Posicion objetivo alcanzada PLC%d' ,i);
Actualizar_Log(app,texto);
Lampara_Error(app);
end
end
% Leer Datos
try
Piezas_antes_lectura(i) = app.PLC(i).C_Piezas;
Piezas_Def_antes_lectura(i) = app.PLC(i).C_Piezas_Def;
Datos = app.PLC(i).MB_Con.ReadInt16(app.Direccion_Leer_Datos,
app.Longitud_Datos_Lectura);
app.PLC(i).Encoder_V = Datos(1,1);
app.PLC(i).Encoder_H = Datos(1,2);
```

```
app.PLC(i).Encoder_G = Datos(1,3);
app.PLC(i).Paso = Datos(1,4);
app.PLC(i).C_Piezas_M1 = Datos(1,6);
app.PLC(i).C_Piezas_M2 = Datos(1,7);
app.PLC(i).C_Piezas_Def_M1 = Datos(1,9);
app.PLC(i).C_Piezas_Def_M2 = Datos(1,10);
app.PLC(i).Tiempo_Ciclo = Datos(1,11);
app.PLC(i).C_Piezas = app.PLC(i).C_Piezas_M1 + app.PLC(i).C_Piezas_M2;
app.PLC(i).C_Piezas_Def = app.PLC(i).C_Piezas_Def_M1 + app.PLC(i).C_Piezas_Def_M2;
catch
texto = sprintf('Error lectura Datos PLC%d' ,i);
Actualizar_Log(app,texto);
Lampara_Error(app);
end
if app.Cambio_Produccion % Si hay cambio en parametros de produccion, se comprueba
la recepcion en PLC
try
app.PLC(i).Modelo = app.PLC(i).MB_Con.ReadInt16(app.Direccion_Leer_Modelo,
app.Longitud_Modelo_Lectura);
catch
texto = sprintf('Error lectura Modelo PLC%d' ,i);
Actualizar_Log(app,texto);
Lampara_Error(app);
end
end
% Leer Fallos
if app.P_Robot_Sel == i % Unicamente del robot seleccionado
try % Fallos Generales
app.PLC(i).Fallos_General =
de2bi(app.PLC(i).MB_Con.ReadByte(app.Direccion_Leer_Fallos_General,
app.Longitud_Fallos_General));
catch
texto = sprintf('Error lectura Fallos generales PLC%d' ,i);
Actualizar_Log(app,texto);
Lampara_Error(app);
end
                        try % Fallos Comunicaciones

                            app.PLC(i).Fallos_Com =
de2bi(app.PLC(i).MB_Con.ReadByte(app.Direccion_Leer_Fallos_Com,
app.Longitud_Fallos_Com));
                        catch

                            texto = sprintf('Error lectura Fallos comunicacion
PLC%d' ,i);
                            Actualizar_Log(app,texto);
                            Lampara_Error(app);

                        end
```

```
                            try % Fallos Paso actual

                                Leer_Fallos_Paso = sprintf("DB18.DBB %s",
        app.Direccion_Leer_Fallos_Paso(app.PLC(i).Paso));
                                app.PLC(i).Fallos_Paso =
        de2bi(app.PLC(i).MB_Con.ReadByte(Leer_Fallos_Paso, app.Longitud_Fallos_Paso));
                            catch

                                texto = sprintf('Error lectura Fallos Paso PLC%d' ,i);
                                Actualizar_Log(app,texto);
                                Lampara_Error(app);

                            end
                        end
                    end

            end

            % Desactivar flag cambio produccion
            if app.Cambio_Produccion

                app.Cambio_Produccion = false;

            end

            % Si se ha completado pieza o se ha descartado, se actualiza la
            % pestaña de Control de Produccion
            if sum(Piezas_antes_lectura) < (app.PLC(1).C_Piezas + app.PLC(2).C_Piezas) ||
sum(Piezas_Def_antes_lectura) < (app.PLC(1).C_Piezas_Def + app.PLC(2).C_Piezas_Def)

                Actualizar_Tab_Produccion(app)

                if sum(Piezas_Def_antes_lectura) < (app.PLC(1).C_Piezas_Def +
app.PLC(2).C_Piezas_Def)

                    texto = sprintf('Pieza Descartada');
                    Actualizar_Log(app,texto);

                end
            end
            Gestion_Emergencia(app) % Actualizar estatus emergencia

            % Actualizar información en las pantallas
            Actualizar_Tab_Principal(app);
            Actualizar_Tab_Estados(app);
            Actualizar_Tab_CM(app);
           %fin fct Leer_PLC
        end
```

```
        function Sistema_Encender_Apagar(app, estado)


            switch estado
                case 1
                    color = app.Gris_standard;
                    status = 1;

                case 0
                    color = app.Gris;
                    status = 0;
            end

            % Acciones

            %Activar/Desactivar botones
            for i = 1:length(app.Encender_Apagar_E)

                app.Encender_Apagar_E(i).Enable = status;

            end


%               %Cambiar colores
            for i = 1:length(app.Encender_Apagar_C)

                app.Encender_Apagar_C(i).BackgroundColor = color;

            end
            %fin fct Sistema_Encender_Apagar
        end
```

```
        function Sistema_Auto_Manual(app)

            for i = 1:length(app.Manual_E)

                app.Manual_E(i).Enable = not(app.Man_Auto);

            end

            for i = 1:length(app.Auto_E)

                app.Auto_E(i).Enable = app.Man_Auto;

            end

            app.ACTIVARMODOMANUALLabel.Visible = app.Man_Auto;
            %fin fct Sistema_Encender_Apagar
        end
```

```
    function Actualizar_Log(app, texto)
        % Actualiza la informacion en el Log. Se muestran los ultimos 6
        % mensajes

        app.Log_texto{app.LogIndice} = texto;

        try

            if app.LogIndice <= 6

                app.Log.Value = app.Log_texto(1:app.LogIndice);

            else

                app.Log.Value = app.Log_texto(app.LogIndice-6:app.LogIndice);

            end

            app.LogIndice = app.LogIndice + 1;

        catch
        end
    end
```

```
    function Desactivar_botones_movimiento(app)
        % Desactivar los botones de movimiento al apretar uno nuevo

        for i = 1:length(app.Vector_Movimientos)
            if app.Vector_Movimientos(i).Value
                app.Vector_Movimientos(i).Value = false;
            end
        end

    end
end
```