Universidad Zaragoza

Trabajo Fin de Grado

Automatización de un proceso de fabricación en Factory I/O controlado a través de Unity Pro

Autor/es

Jaime Calvo Baigorri

Director/es

Cristian Mahulea

Escuela de Ingeniería y Arquitectura/Tecnologías Industriales

2021

Automatización de un proceso de fabricación en Factory IO controlado a través de Unity Pro

RESUMEN

La primera parte del trabajo se trata de realizar un modelo simulado de una estación, concretamente la estación 6, de una célula de fabricación situada en el laboratorio 0.06 del edificio Ada Byron de la Escuela de Ingeniería y Arquitectura.

La segunda parte del trabajo consiste en el diseño de una nueva estación que sirva de continuación a esta y realice las funciones de almacén automatizado.

Se ha llevado a cabo un inventario de los distintos componentes, sensores y actuadores que conforman la estación 6, así como sus respectivas conexiones con el PLC que se encarga del control. Conocida en detalle la estación real, y diseñada la segunda estación, se va a emplear el software de simulación Factory IO para implementar ambos modelos. En el caso de la primera estación, se debe buscar la mayor semejanza posible con el sistema real.

Una vez realizadas las maquetas, se desarrolla el código de control. En primer lugar, al tratarse de sistemas de eventos discretos, se ha representado el esquema de control por medio de una Red de Petri. El software empleado para programar el código de control ha sido Unity Pro. Respecto al control implementado para la primera estación, se desarrolla de tal forma que sea apto para el modelo real y para el modelo simulado, es decir, que sin modificar la línea de código logre el mismo funcionamiento en la estación del laboratorio y la maqueta desarrollada en el software de simulación.

Para conseguir que el código de control sea válido para el modelo real y el simulado de la estación 6, dentro de Unity Pro, se ha realizado un mapeo de las señales de entrada y salida para establecer su dirección de memoria en función de si estamos trabajando con la estación real o la simulada.

La implementación de la segunda estación se realiza de tal forma que se sincronice el evento de dejar una base en la estación 6, con el evento de aparición de una base en la segunda estación. Para lograr esta sincronización es necesario que la segunda estación lea como variable de entrada, la variable de salida de la estación 6 (el brazo robot deja una base en la cinta).

Finalmente, se han realizado dos vídeos mostrando el resultado de estas simulaciones.

ABSTRACT

The first part of the work carried out is a simulation model corresponding to a station, specifically station 6, of a manufacturing cell located in the laboratory 0.06 of the Ada Byron building of the School of Engineering and Architecture, Zaragoza.

The second part of the work consists of the design of a new station that serves as a continuation of this one and performs the functions of an automated warehouse.

An inventory of the different components, sensors and actuators that make up the 0.06 station has been carried out, as well as their respective connections with the PLC that oversees the control. Once the real station is known in detail, and the second station is designed, the

Factory IO simulation software will be used to implement both models. In the case of the first station, the closest possible resemblance to the real system should be sought.

Once the models have been made, the control code is developed. First, since these are discrete event systems, the control scheme has been represented by means of a Petri net. The software used to program the control code was Unity Pro. Regarding the control implemented for the first station, it is developed in such a way that it is suitable for the real model and for the simulated model, i.e., without modifying the line of code, it achieves the same operation in the laboratory station and the model developed in the simulation software.

To make the control code valid for the real model and the simulated model of station 6, within Unity Pro, the input and output signals have been mapped to establish their memory address depending on whether we are working with the real or the simulated station.

The control of this station is done in such a way that the succession of events with the first station is sequential. To achieve this sequential succession of events it has been necessary to synchronize both stations, since it is necessary for the second station to read certain input variables, which correspond to output variables of the first station. The communication between two systems present in different computers through a Modbus network has been deepened to establish the synchronization.

Finally, two videos have been made showing the result of these simulations.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCION

El trabajo fin de grado que se detalla a continuación se ha realizado en colaboración con el Departamento de Informática e Ingeniería de Sistemas de la Escuela de Ingeniería y Arquitectura. Se fundamenta en los conceptos impartidos en la asignatura de Ingeniería de Control, acerca del control de **sistemas de eventos discretos (SED)**, más concretamente en las Redes de Petri. La primera parte del trabajo se centra en la simulación y control de una estación real de una célula de fabricación flexible disponible en el laboratorio L0.06.

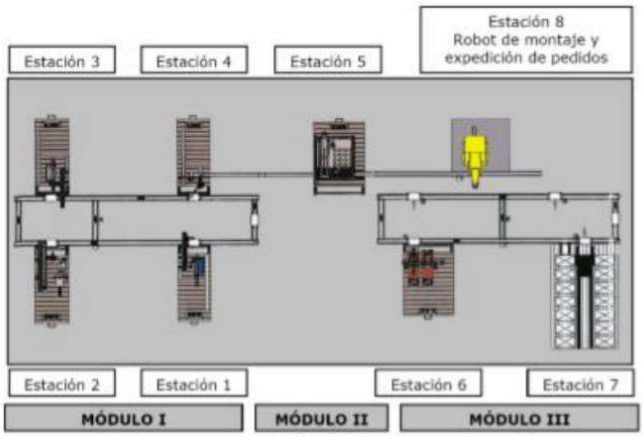


Figura 1: Esquema de la célula de fabricación flexible

La segunda parte del trabajo consiste en el diseño y control de una segunda estación que realizará las funciones de almacén y se programa de forma que la sucesión de eventos sea secuencial y coordinada con la primera estación. Cada estación se simulará en dos ordenadores diferentes, para ello será necesario estudiar la comunicación entre las distintas herramientas.

En el desarrollo de síntesis se ha empleado el software de simulación **Factory IO**. La elección de este software frente a otros es debido a que previamente en el estudio del grado ya se han realizado prácticas de asignaturas con él y, además, visualmente es sencillo y fácil de comprender.

El código de control se implementa y compila en el software **Unity Pro**. El PLC al que está conectada la estación real es de Schneider Electric, es por ello, que a la hora de elegir el software de control, se ha optado por el de la marca.

Las técnicas de implementación de una estación real a través del software Factory IO se apoyan un trabajo fin de grado previo realizado por Fernando Grima Montesa [9] que trabajó con otra estación de la célula de fabricación.

1.1. Objetivos

El objetivo principal de la primera parte de este trabajo es aumentar el grado de flexibilidad de la universidad en esta época de pandemia actual, y poder proyectarlo para tiempos en los que la situación haya mejorado. Llevar la estación real a un software de simulación accesible por los alumnos desde sus ordenadores personales, permite impartir la docencia de manera online, evitando la presencialidad de los alumnos en un laboratorio, en una única estación. Además, siendo que el objetivo de la práctica es desarrollar un programa que consiga el funcionamiento deseado sobre la estación, puede permitir que alumnos, bien porque

tengan trabajo, solape de horarios, o incluso algún imprevisto, puedan desarrollar el programa y antes de una fecha límite demostrar que funciona.

Uno de los principales problemas durante la realización de las prácticas sobre la estación real es la aglomeración de alumnos a la hora de probar sus líneas de código. Probar los códigos directamente sobre la estación real hace que el número de intentos por alumno hasta lograr el funcionamiento óptimo sea elevado, provocando largas esperas entre un estudiante y otro. La implementación de la maqueta sobre un software de simulación permite al alumno probar su código de control el número de veces que haga falta sin retrasar a ningún compañero, ya que solo implementará el control sobre la estación real cuando haya funcionado en la simulada.

Conseguir que los alumnos implementen sobre la estación real únicamente el código final, no solo supondrá un beneficio para los estudiantes sino también para la universidad. Pueden darse casos en los que el programa realizado por el alumno sea defectuoso y ponga en peligro la integridad de la estación, por tanto, reduce el riesgo de avería en la estación real.

Un segundo objetivo es, facilitar y dotar de más herramientas a los alumnos que, durante el estudio del grado, tengan que trabajar con esta estación. Se pretende facilitar la comprensión del funcionamiento y los aspectos básicos de la estación. De esta manera, se pretende motivar y fomentar el estudio del control de este tipo de estaciones, como resultado a una comprensión plena y amena de los fundamentos de los sistemas de eventos discretos.

Para lograr estos objetivos, es muy importante que los alumnos a la hora de programar el control de la estación no tengan que diferenciar si están trabajando con la estación real o simulada. La estación simulada deberá tener la misma dinámica que la real, y en aspectos en los que no sea posible por limitaciones del software, deberá ser subsanado mediante el control. Se crearán secciones de control, además de la principal, para solventar las diferencias entre estaciones, y de esta manera hacer su funcionamiento semejante.

En la realización del código de control, a la hora de definir las variables se pretende trabajar con **variables únicas** con el objetivo de simplificar la línea de código. Esto implica que las direcciones de memoria de las variables reales sean iguales a las de simulación. En caso de no ser posible, se debe realizar un **mapeo de señales** para que los alumnos, solo con indicar si trabajan en simulación o con la maqueta real, si puedan trabajar con variables únicas.

Respecto a la segunda parte del trabajo, la cual consiste en el diseño de una estación que amplíe el proceso que se inicia con la estación real, se pretende construir y controlar una nueva estación que automatice un proceso de paletización y almacenaje. Mediante el control se pretende lograr un funcionamiento secuencial entre ambos sistemas.

El objetivo principal de esta segunda parte es realizar la comunicación entre dos sistemas que se ejecutan en dos ordenadores diferentes. Para lograr una sucesión de hechos secuencial en la simulación, es necesaria la lectura de variables compartidas, es decir, la segunda estación debe leer variables de entrada/salida de la primera y viceversa. Se va a realizar un estudio sobre la conexión de varios dispositivos a una misma red Modbus con la intención de lograr esa comunicación y esa simulación coordinada entre estaciones.

La simulación de las dos estaciones deberá realizarse en dos ordenadores diferentes. De esta forma se pretende dar una solución a grandes procesos productivos en los que por tamaño, no es viable su simulación en una única escena de Factory IO, bien porque no caben o bien

porque es complicado prestar atención a todos los sucesos si todo se encuentra condensado en un mismo archivo.

1.2. Alcance

En la memoria del trabajo se detallan las distintas etapas que se han realizado en este trabajo. En primer lugar se va a comentar la **parte física de la estación**, donde primero se habla de los componentes de la estación real y seguido se nombran los objetos empleados en Factory IO para suplirlos. En este apartado se va a comentar los primeros problemas que han surgido en la realización del trabajo, ya que no todos los componentes de la estación real los podíamos encontrar en el software. Se detallará la solución propuesta a los problemas mencionados.

A continuación, se detalla todo lo relacionado con el **control**. Al igual que con la estructura, primero se detalla el control de la estación real, hablando de las entradas y salidas del sistema y del funcionamiento deseado. Tras determinar el funcionamiento óptimo se realiza la representación mediante la red de Petri de la estación. El brazo robot es el mecanismo más complejo de la estación, ya que tiene asociadas varias entradas y salidas.

Tras detallar todo lo referente a la estación real, se va a comentar los problemas que han surgido en el control de la estación simulada. Esto se debe a las diferencias entre las partes físicas de las estaciones, las cuales se habrán comentado en la primera parte del trabajo. En esta parte de la memoria se explica la solución adoptada para conseguir el objetivo final, un código de control único.

A continuación, se detalla el **mapeo de las señales** que se ha tenido que realizar por no poder emplear variables únicas. En este apartado, además de explicar cómo se ha realizado, se va a detallar el fundamento y que se debe hacer para diferenciar el trabajo sobre la estación real y sobre la simulación.

Terminada la explicación sobre el mapeo de señales, se realizará el diseño y control de la estación que sirve de ampliación a la estación real. Al igual que con la primera estación, se empezará desarrollando la estructura. El lenguaje de programación empleado para esta estación es distinto al usado anteriormente, por tanto, antes de entrar con el control de la estación se introducirá este nuevo lenguaje.

Una vez introducido el lenguaje de programación se explica cuáles son las variables que se deben leer entre estaciones para lograr un funcionamiento coordinado. También se detalla cuál ha sido la manera de programar las estaciones para reducir el tiempo de ciclo al máximo posible.

La parte referente a la estación de almacenaje y paletización termina con una explicación sobre cómo se ha conseguido la comunicación entre dos escenas, que se encuentran en dos ordenadores diferentes, pero para la simulación precisan de estar conectadas a la misma red Modbus.

Por último encontramos dos vídeos de las simulaciones realizadas en este trabajo. Estos vídeos pretenden mostrar el funcionamiento implementado en ambas estaciones, así como facilitar la comprensión apoyando la explicación en una representación visual.

La memoria del trabajo concluye con una serie de anexos. El primer anexo, habla sobre el software de simulación Factory IO y tiene como fin explicar cómo se ha llevado a cabo la creación de las escenas, hablando sobre las distintas funcionalidades que ofrece el software. El

segundo anexo, tiene el mismo objetivo que el primero, pero en este caso se detalla el segundo software empleado en este trabajo, Unity Pro. El tercer anexo será el guion de la práctica en el que se ha basado el funcionamiento deseado de la estación real. El último anexo será el código de control desarrollado en Unity Pro.

2. ESTRUCTURA DE LA ESTACIÓN

La maqueta real de la estación se sitúa en el laboratorio 0.06 del edificio Ada Byron. La estación se encarga de proporcionar las bases de dos piezas distintas a un cinta transportadora central. El transporte de estas piezas lo realiza un brazo robot.

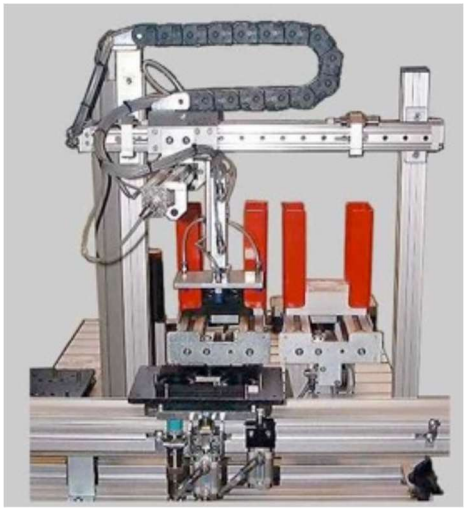


Figura 2: Estación Nº6

2.1. Partes físicas

Las partes físicas de la estación son todos los objetos que van a estar controlados por el PLC, además de aquellos que sirven como soporte de los anteriores. También consideramos como parte física a las bases y al lugar donde se almacenan.

- **2 depósitos**. Almacenan las bases, uno para cada tipo. Se tratan de alimentadores por gravedad, por lo que las placas se almacenan unas encima de otras y se van suministrando una por una.
- **2 expulsores**. Se emplean para colocar las placas en la zona donde serán recogidas por el brazo robot para ser transportadas. Estos expulsores se tratan de cilindros de simple efecto, es decir, una única variable gestiona la extensión y recogida del mismo.

Figura 3: Depósitos de gravedad y expulsores estación real

- **1 brazo robot**. Se trata de un brazo que se puede desplazar en los tres ejes XYZ y se encarga de llevar las bases a una cinta transportadora central. El brazo robot tiene el movimiento en X y en Y a través de cilindros de doble efecto, y el movimiento en Z a través de un cilindro de simple efecto.

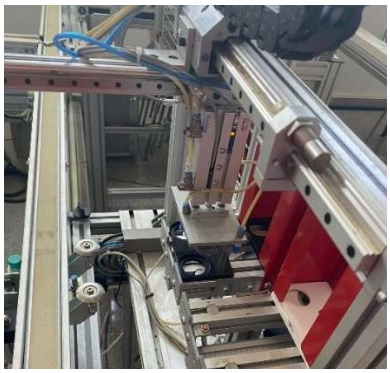


Figura 4: Brazo Robot estación real

- **1 cinta transportadora**. Es el destino final de las bases, una vez llegan son transportadas a la siguiente estación.
- **Paleta de control**. En ella encontramos los botones para indicar la marcha, paradas de emergencia, modo de funcionamiento… etc.

Figura 5: Botonera estación real

El tipo de actuador (cilindro de simple o doble efecto) que tiene cada elemento es importante de cara al control y es uno de los problemas que se ha tenido que solventar mediante el control debido a la diferencia con la estación simulada. Estas diferencias y las soluciones adoptadas se exponen más adelante en los *apartados 3 y 5.*

2.2. Sensores

Para poder controlar la parte física de la estación mediante un autómata programable se instalan una serie de sensores que actuarán como las entradas del sistema. Encontramos dos tipos de sensores:

- Incorporados en el componente. Este tipo de sensores van a formar parte de la estructura de una parte física nombra anteriormente. Van a detectar una posición concreta, de tal forma que si la estructura se halla en esa posición el sensor se pondrá a 1. Un ejemplo sería el sensor de brazo abajo, el cual detecta cuando el brazo está abajo para ponerse a 1. Además de en el brazo robot, este tipo de sensores lo encontramos en los expulsores.
- Situados en la estructura fija. Estos sensores son de tipo óptico, se encuentran instalados en la parte fija de la estación y van a detectar cuando un cuerpo se sitúa delante de ellos cortando el haz de luz. Los encontramos en la zona de recogida de las bases detectando si se ha expulsado o no una base.

Además de estos sensores, tenemos uno para cada botón de la paleta de control para detectar si se desea poner en marcha, parar o pasar de funcionamiento manual a automático, entre otros.

3. MODELADO EN FACTORY I/O

En el software de simulación Factory IO encontramos una gran variedad de elementos típicos en procesos de fabricación. En nuestro caso, se ha recorrido el menú de objetos buscando las partes físicas y los sensores descritos en el apartado 2.

3.1. Parte física

- Expulsores

Lo encontramos en el apartado de *Light Load Parts.* Lleva incorporados dos sensores, para el brazo extendido y para el brazo recogido. En la maqueta real, los expulsores tienen un único sensor que se pone a 1 cuando el brazo está recogido y a cero cuando esta extendido. Por tanto, en el software de simulación emplearemos únicamente el sensor de límite trasero.

Figura 6: Empujador Factory IO empleado como expulsor vs placas expulsoras estación real

- Brazo Robot

Se trata de la pieza más compleja de la estación, no solo porque la estructura es más aparatosa, sino porque tiene movimientos más difíciles y variados. En este caso, vamos a encontrar la pieza en el menú de *stations*.

Esta estación consta de un brazo robot y su soporte. El brazo robot tiene permitido el movimiento en XYZ y además, puede girar en torno al eje Z el elemento terminal. Lleva incorporado un sistema de vacío que permite agarrar objetos. El modo de funcionamiento puede ser digital, analógico o mixto.

Figura 7: Pick&Place Factory IO empleado como brazo robot vs brazo robot estación real

- Paleta de control

La paleta de control en el caso de la maqueta real consiste en una botonera con tres botones, marcha, rearme y seta de emergencia. También incluye dos conmutadores, ind-int que sirve para elegir qué lado expulsará la primera base, y man-aut que indica el modo de funcionamiento (manual o automático). Por último, la estación real incorpora una baliza que se enciende cuando no quedan bases.

En Factory IO encontramos estas partes de la estación en la sección de *Operators*, salvo la baliza que está en *Warning Devices*. Se emplea una caja metálica a modo de botonera.

Figura 8: Paleta de control estación simulada vs estación real

Factory IO no dispone de depositos de gravedad como los que encontramos en el laboratorio. Para suplirlos se emplea una funcionalidad del software que genera las bases automáticamente. En la estación real se llega a un punto en el que ambos depósitos se quedan sin bases. En el apartado de *control de la estación simulada* se explica como se lleva este supuesto al software de simulación.

Los distintos modos de funcionamiento de cada una de las partes, sus características y como establecer el modo de funcionamiento se detallan en el *Anexo I.*

3.2. Sensores

Además de los elementos físicos de la estación real, en el software de simulación debemos añadir los sensores descritos anteriormente. Los sensores del brazo, al no poder instalarlos sobre él como en la estación real, se colocan a lo largo de su soporte. De esta manera, se instalan:

- 4 sensores capacitivos para indicar si se ha expulsado base a ambos lados y para conocer si el brazo esta a la derecha o a la izquierda.
- 2 sensores difusos para indicar si el brazo esta delante o detrás.
- El resto de sensores los incorporan los elementos físicos de la instalación.

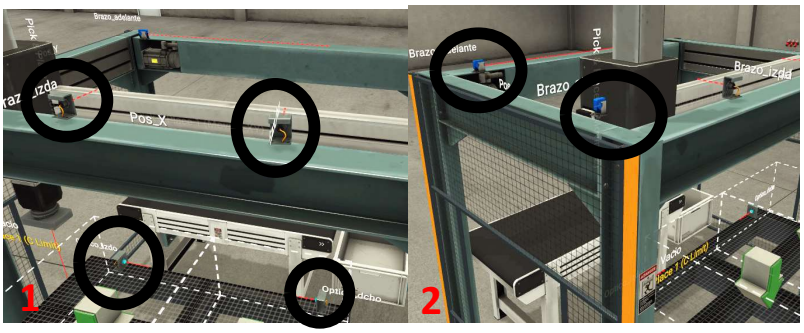


Figura 9: Sensores capacitivos(1) y sensores difusos(2) instalados

Tras haber recopilado todas las partes físicas de la estación real en el software de simulación realizamos el ensamblaje.

Figura 10: Estación simulada 1 definitiva

4. CONTROL DE LA ESTACIÓN REAL

Hay distintos formalismos de modelado para los sistemas de eventos discretos y en este trabajo se ha elegido las **Redes de Petri**. Este tipo de representación, en estaciones complejas como la que nos atañe, son más fáciles de comprender por el usuario debido a su naturaleza gráfica, teniendo a su vez un número más reducido de nodos que los autómatas finitos deterministas. La Red de Petri está definida por dos conjuntos finitos y disjuntos de lugares y transiciones que se conectan utilizando arcos. Cada lugar, representado por un círculo, corresponde a una variable de estado del sistema, mientras que las transiciones, representadas por rectángulos, hacen referencia a los sucesos que provocan un cambio de estado en el sistema.

Un lugar puede tener una o varias marcas. El marcado de un lugar corresponde al valor de la variable de estado correspondiente y es el estado (local) . Una transición está sensibilizada si todos los lugares de entrada tienen un número de marcas mayor o igual que el peso del arco que conecta el lugar con la transición. Una transición sensibilizada se puede disparar si ocurre el suceso (evento) asociado (puede ser un sensor que detecta un objeto o, en general, cualquier función lógica definida utilizando las variables de entrada).

Las Redes de Petri nos permiten modelar de una forma muy intuitiva estructuras típicas de un proceso de producción como puede ser un almacén o un recurso compartido por varios procesos, como un brazo robot. En esta estación ha sido necesario para cumplir con el funcionamiento deseado, el empleo de concurrencias. A partir de la estructura típica de una concurrencia vamos a explicar las características, anteriormente mencionadas, de este modelo de representación.

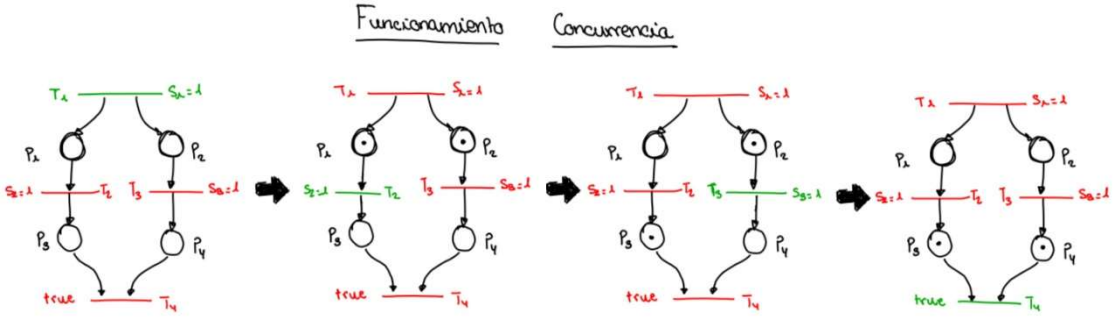


Figura 11: Esquema de funcionamiento de una concurrencia

Con el disparo de T1, se producen dos marcas, una en el lugar p1 y otra en el lugar p2. De esta manera, T2 y T3 están sensibilizadas, a la espera de que su respectivo evento ocurra para dispararse. Cuando S2=1, se dispara la transición T2, produciendo una marca en p3. Tanto el lugar p3, como el p4 modelan operaciones de espera. Se emplean para indicar que un proceso ha terminado y se encuentra esperando a que termine el otro proceso concurrente para continuar con el ciclo. Cuando S3=1 se dispara T3, produciendo una marca en p4. Al tener una marca en p3 y p4, queda sensibilizada la transición T4, y al ser su condición de disparo que la transición esté sensibilizada, se dispara directamente.

4.1. Entradas y salidas del sistema

Para poder controlar el sistema utilizando la Red de Petri, es necesario conocer las variables de entrada y salida. Las entradas (por ejemplo los sensores) se utilizan para programar

las transiciones, el cambio de valor de un sensor supone el suceso que puede provocar un cambio de estado del sistema. Por otro lado, las variables de salida corresponden a actuadores.

En el caso de las variables de entrada, todas ellas son del tipo booleano. Esto implica que el sensor tiene dos estados, 0 y 1. A continuación se detallan todas las entradas y que significa la puesta a 1 de cada sensor.

Entrada	Condición de puesta a 1
Izdo_atras	El expulsor izquierdo está recogido
Dcho_atras	El expulsor derecho está recogido
Optico_izdo	Base detectada en el lado izquierdo
Optico_dcho	Base detectada en el lado derecho
Brazo_arriba	El brazo que manipula la carga esta arriba
Brazo_abajo	El brazo que manipula la carga está abajo
Brazo_dcha	El brazo que manipula la carga está a la derecha
Brazo_izda	El brazo que manipula la carga está a la izquierda
Seta_emergencia	Seta de emergencia enclavada
Marcha	Pulsador de Marcha apretado
Man_aut	Conmutador en posición AUT
Rearme	Pulsador de Rearme apretado
Ind_int	Conmutador en posición INT
Brazo_atras	El brazo que maniupula la carga está atrás
Brazo_adelante	El brazo que manipula la carga está delante
Vacío	El vacuostato detecta vacío

Tabla 1: Entradas y su condición de puesta a 1 de la estación real

Respecto a las variables de salida, encontramos dos tipos. Por un lado, tenemos los cilindros de simple efecto. Este es el caso de los expulsores, e implica que, si la salida asociada al expulsor se pone a 1, este se extiende, y si se pone a 0 se recoge. Además del expulsor, el movimiento en Z del brazo también se realiza por medio de un cilindro de simple efecto. El segundo tipo de salida se trata de cilindros de doble efecto. Estas salidas corresponden al movimiento en X e Y del brazo robot y se deben manejar de manera impulsional. Este tipo de salida cuando se pone a 1 comienza el movimiento en una dirección, y cuando se pone a 0 el movimiento se detiene.

En la siguiente tabla se recogen las salidas del sistema y la condición de puesta a 1.

Salida	Condición de puesta a 1
Expulsar_izdo	Extender expulsor izquierdo
Expulsar_dcho	Extender expulsor derecho
Coger_placa	Las ventosas realizan vacío
Bajar_brazo	Bajar el brazo
Mover_izda	Mover el brazo a la izquierda
Mover_dcha	Mover el brazo a la derecha
Mover_adelante	Mover el brazo adelante
Mover_atras	Mover el brazo atrás
Baliza	Encender la luz de la baliza

Tabla 2: Salidas y su condición de puesta a 1 de la estación real

4.2. Funcionamiento deseado

Una vez conocidas las entradas y salidas del sistema vamos a explicar el funcionamiento de la estación para posteriormente realizar la Red de Petri y así describir de manera gráfica el control que se pretende realizar.

4.2.1. Ciclo normal

Antes de comenzar un ciclo normal, el brazo robot deberá situarse sobre el expulsor derecho con el brazo arriba. Además, los dos expulsores deberán estar recogidos. Se define así el estado de reposo del sistema.

- El ciclo comenzará cuando el usuario apriete el pulsador de Marcha. Dependiendo de la posición del conmutador ind_int se expulsará una base u otra (en posición int se expulsa primero la base derecha). El sensor óptico nos indicará que la base se encuentra en la posición de recogida.
- El brazo robot descenderá, cogerá la placa y la llevará a la cinta transportadora. La placa se deja alineada con el depósito derecho, por tanto, además de avanzar hacia adelante, deberá desplazarse a la derecha siempre que lo precise.
- Una vez haya entregado la base, el expulsor contrario al del ciclo anterior expulsará la siguiente base dando comienzo a un nuevo ciclo.

4.2.2. Fallos en el sistema

Anteriormente se ha descrito el ciclo de funcionamiento ideal. Sin embargo, durante el proceso de producción pueden suceder una serie de inconvenientes. A continuación, se explica todos los supuestos negativos y como debe actuar el sistema ante estos inconvenientes.

- **Seta de emergencia**: Si el operario pulsa la seta de emergencia, se presupone que está ocurriendo algo grave y se debe parar el funcionamiento de inmediato. Los expulsores se deben recoger y la cinta transportadora frenar. En cuanto al brazo robot, se sube, por si está ocurriendo un problema de aplastamiento, se deja de hacer vacío, por si el problema está en un atrapamiento de algo indeseado, y además, se debe frenar en el punto exacto en el que se encuentra, por si el problema es de colisión.
- **Sin bases**: Si ambos depósitos se quedan sin bases se debe activar la baliza y parar el funcionamiento hasta que sean repuestas. Una vez repuestas se pulsará el botón de rearme, y al pulsar el botón de marcha comenzará de nuevo el ciclo.
- **Pulsador de rearme**: El botón de rearme se emplea para indicar que los depósitos que se han quedado sin bases vuelven a estar llenos, y puede comenzar de nuevo el funcionamiento.

4.2.3. Red de Petri

Una vez conocidas las entradas y salidas del sistema, y el funcionamiento deseado de la estación, se procede a realizar la Red de Petri de control. Para facilitar la comprensión, se ha representado con distintos colores.

- Negro: Se corresponde con el ciclo normal de la estación.
- Rojo: Hace referencia al estado de emergencia por no tener bases en los depósitos.

- Verde: Se trata de variables auxiliares que se implementan en el control para establecer que expulsor debe trabajar en el inicio de un nuevo ciclo o para saber si hay o no placas en un depósito.

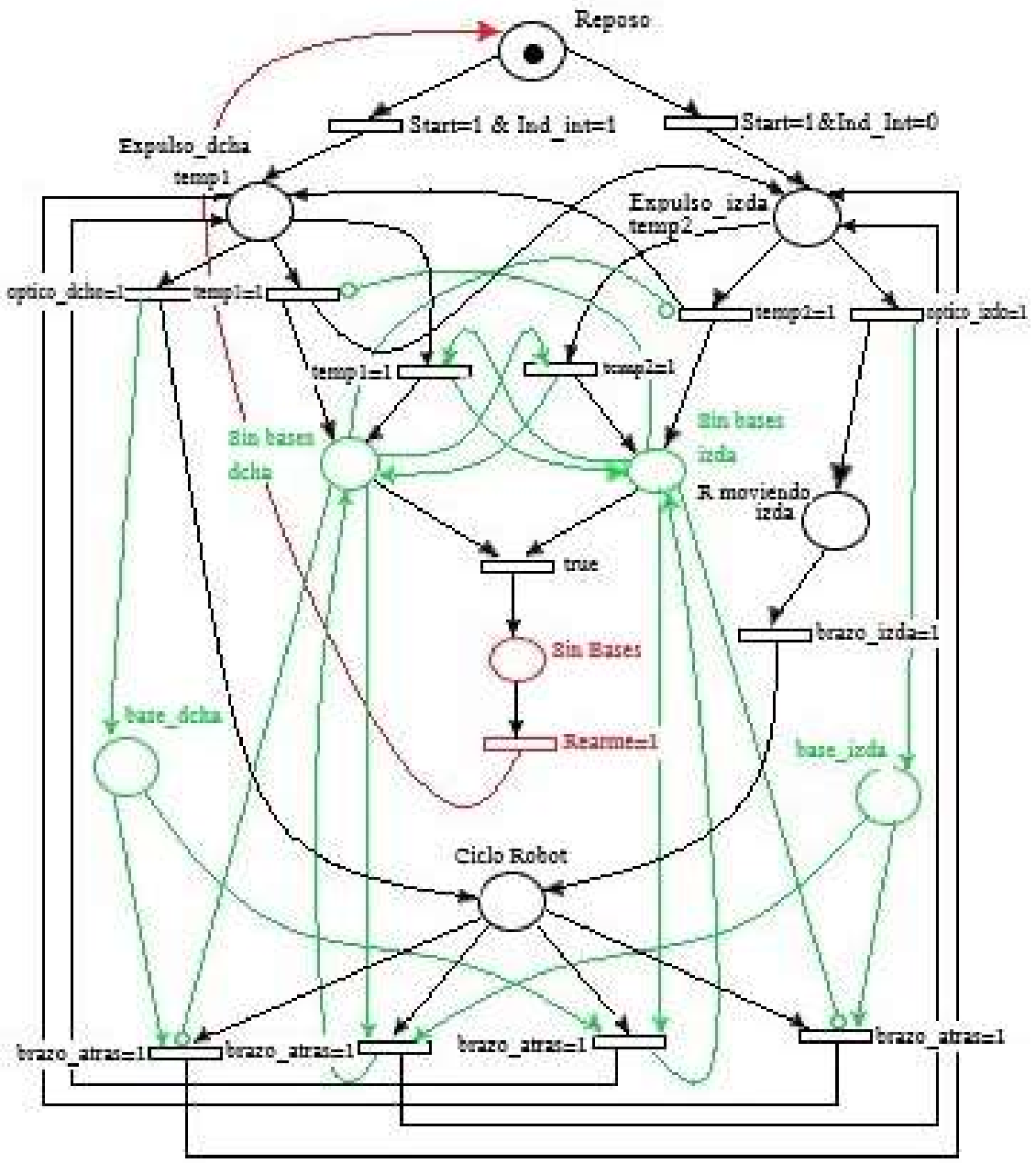


Figura 12: Red de Petri de la estación 6

A continuación, se explica cómo se ha implementado en la Red de Petri el funcionamiento del sistema ante una falta de bases en los depósitos.

Una vez se acciona el botón de marcha, además de expulsar la base correspondiente, se inicia un contador. Se establece un tiempo máximo de respuesta del sensor óptico. Si transcurrido el tiempo, el sensor no se ha puesto a 1, significa que no se ha expulsado ninguna base y que por tanto no queda ninguna base en el depósito correspondiente. A continuación se explican, las posibles respuestas del sistema en función de la disponibilidad de bases en los depósitos.

Suponemos que expulsamos del depósito de la derecha y no tenemos piezas (el contador alcanza el valor límite establecido), mientras que en el depósito de la izquierda si tenemos piezas. Una vez haya transcurrido el tiempo máximo establecido en el contador, se disparará una transición que expulsará una base de la izquierda y dejará una marca en el lugar referente a la variable auxiliar *sin bases dcha.* Esto va a provocar que el arco inhibidor que sale del lugar *sin bases dcha* no dispare nunca su transición asociada al tener un número de marcas mayor o igual que el peso asociado al arco. Cuando el depósito izquierdo se quede sin bases, se disparará la transición que deja una marca en el lugar asociado a la variable auxiliar *sin bases izda*, al estar la otra transición asociada al contador desensibilizada por el arco inhibidor. Al tener una marca en cada variable auxiliar, se va a disparar la transición que nos produce una marca en el estado de emergencia (*Sin placas*).

Estas dos variables auxiliares también las vamos a emplear para que una vez se haya detectado un depósito vacío, el sistema expulse piezas del depósito lleno en todos los ciclos hasta quedarnos sin existencias. Por ello, tras dejar la pieza el robot, se pueden disparar hasta cuatro transiciones distintas.

4.3. Control del brazo robot

El control del brazo robot se va a llevar a cabo mediante un vector de estados. El movimiento del brazo también es secuencial, y el recorrido es el mismo para cualquier ciclo. Para cada estado vamos a definir el valor de todas las variables asociadas al brazo robot. En la siguiente Red de Petri se representan todos los eventos asociados al lugar *Ciclo Robot,* que aparece en la red anterior.

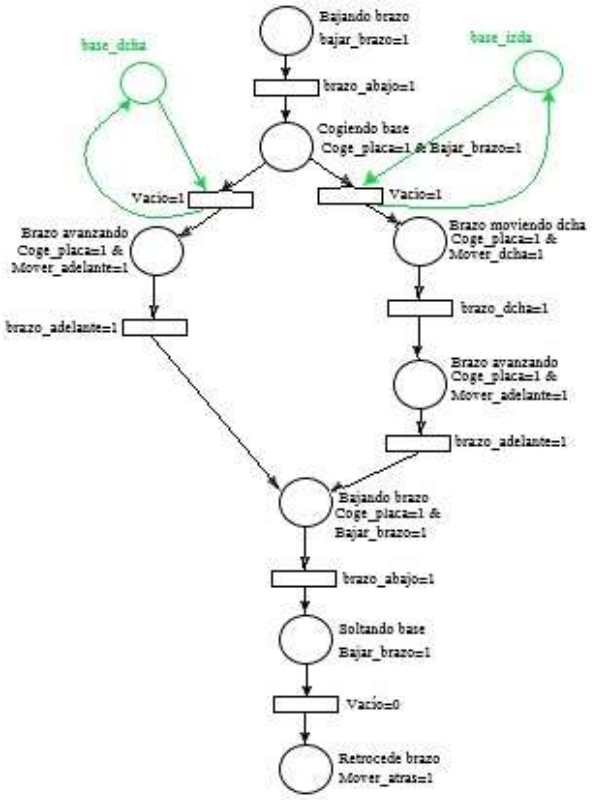


Figura 13: Red de Petri del lugar "Ciclo del brazo robot"

En la *Figura 13* podemos observar todos los estados salvo el estado de reposo, el estado de desplazamiento del brazo a la izquierda y el estado de emergencia, que aparecen que la *Figura 12.*

4.4. Control en Unity Pro

El código de control se va a realizar con el software Unity Pro de la compañía Schneider Electrics. Este software nos permite realizar la estructura de control mediante diversos métodos. En esta primera estación se va a emplear el lenguaje de programación ST, texto estructurado. Este tipo de lenguaje trabaja con expresiones, construcciones compuestas por operandos y operadores que devuelven un valor durante la ejecución. Este tipo de lenguaje consiste en sucesivas expresiones de tipo condicional, en la que cada una de ellas lleva asociada una conclusión final.

Se ha elegido este tipo de lenguaje porque es cómodo de implementar cuando se trabaja con sensores de tipo booleano. Los parámetros que caracterizan la condición serían los sensores (o entradas del sistema) y la conclusión asociada sería la salida del sistema. Asemejándolo con el lenguaje de las redes de Petri, la expresión condicional sería la transición y la conclusión el lugar/es que quedarían marcados tras el disparo de la transición.

En Unity Pro vamos a comenzar introduciendo las variables del sistema. La forma de introducir estas variables se explicará con detalle en el Anexo II. A estas variables del sistema hay que darles una dirección. Esta dirección viene determinada por la red Modbus que conecta los distintos PLCs del laboratorio. De esta manera, podemos asociar las variables del programa a los sensores y accionamientos de la estación real. Las variables de la estación se encuentran guardadas en bits de memoria de variables word. En la siguiente tabla podemos ver las direcciones de memoria de las entradas y salidas.

Entrada	Dirección de memoria	Salida	Dirección de memoria
Izdo_atras	%MW35.0	Expulsar_izdo	%MW134.0
Optico_izdo	%MW35.1	Expulsar_dcho	%MW134.1
Dcho_atras	%MW35.2	Coger_placa	%MW134.2
Optico_dcho	%MW35.3	Bajar_brazo	%MW134.3
Brazo_arriba	%MW35.4	Mover_izda	%MW134.4
Brazo_abajo	%MW35.5	Mover_dcha	%MW134.5
Brazo_dcha	%MW37.0	Mover_adelante	%MW135.0
Brazo_izda	%MW37.1	Mover_atras	%MW135.1
Seta_emergencia	%MW37.2	Baliza	%MW135.2
Marcha	%MW37.3		
Man_aut	%MW37.4		
Rearme	%MW37.5		
Ind_int	%MW39.0		
Brazo_atras	%MW39.1		
Brazo_adelante	%MW39.2		
Vacio	%MW39.3		

Tabla 3: Dirección de memoria de las entradas y salidas de la estación real

Una vez hemos definido las variables del sistema asociadas a las entradas y salidas de la estación. Debemos definir las variables auxiliares. Estas variables son aquellas, que vamos a emplear en el código de control para conseguir un correcto funcionamiento, que no llevan

asociada ninguna dirección de memoria porque no se corresponden con los sensores y los actuadores de la estación real. Las variables definidas son:

Variable	Tipo	Función
Contador1	DINT	Tiempo máximo de respuesta entre la expulsión de una base y su detección por el sensor. Determina si hay base en el depósito
Estado	ARRAY	Guardar todos los posibles estados del brazo
Placa_derecha	BOOL	Se pone a 1 cuando la base que está manejando el robot es la derecha.
Placa_izquierda	BOOL	Se pone a 1 cuando la base que está manejando el robot es la izquierda.
Sin_bases_azules	BOOL	Se pone a 1 cuando no quedan bases en el depósito derecho. Indica la disponibilidad del depósito derecho.
Sin_bases_verdes	BOOL	Se pone a 1 cuando no quedan bases en el depósito izquierdo. Indica la disponibilidad de este depósito.
Sin_placas	BOOL	Se pone a 1 cuando no hay placas en ningún depósito. Avisa de la necesidad de reponer y activación de la baliza.

Tabla 4: Variables auxiliares empleadas en el control de la estación real

Tras definir las variables auxiliares, ya tenemos definidas todas las variables necesarias para implementar el código que controlará nuestra estación real. Para ello, como se ha comentado anteriormente, vamos a emplear el lenguaje de programación ST (texto estructurado). En el explorador de proyectos, en la carpeta de programa ➔ tareas ➔ mast ➔ secciones, hacemos clic derecho e insertamos una nueva sección. En esta sección vamos a implementar la línea de código.

5. CONTROL DE LA ESTACIÓN SIMULADA

Uno de los objetivos principales de este trabajo es realizar un código de control que, sin la necesidad de ser modificado, valga tanto para la estación real como para la estación simulada. Para ello, lo primero que se ha realizado es una construcción a nivel estructural lo más similar posible a la maqueta real. Debido a que los recursos de Factory IO son limitados, alguna de las estructuras no funciona exactamente igual que en la realidad, un ejemplo es el brazo robot, que no posee un sensor incorporado en él que detecte si está arriba o abajo. Estos problemas se han solucionado creando nuevas secciones dentro de Unity Pro que reprogramasen el funcionamiento de estos sistemas. En este apartado se va a explicar los distintos problemas que han surgido y como se han subsanado. También se expondrán todas las variables auxiliares que se han añadido al programa. La conexión entre ambos softwares se explica en los Anexos I y II y las líneas de código en el Anexo IV.

5.1. Brazo Robot

5.1.1. Movimiento

En la estación real, el brazo robot tiene un funcionamiento digital, la puesta a 1 de los accionamientos activan el movimiento en la dirección deseada. En la estación de Factory IO, el funcionamiento digital del brazo funciona por cambio de flancos, cada vez que el movimiento en una dirección pasa de 0 a 1 el brazo avanza una distancia igual y constante, 0.125m. En conclusión, el funcionamiento digital de la estación no es válido.

Se establece el funcionamiento híbrido, digital y analógico, en este caso en función del voltaje inyectado, comprendido entre 0 y 10V, el brazo robot se va a mover con un movimiento lineal y uniforme a una posición exacta. Para poder trabajar con variables enteras, se define en Factory IO un factor de escala de 100. De esta manera, el valor de las entradas que llega al control está multiplicado, y el valor que le llega al PLC está dividido. Estos valores analógicos se almacenan en seis variables.

Entrada	Función	Salida	Función
Pos_X	Indica la posición en X	Mov_X	Indica el movimiento en X
Pos_Y	Indica la posición en Y	Mov_Y	Indica el movimiento en Y
Pos_Z	Indica la posición en Z	Mov_Z	Indica el movimiento en Z

Tabla 5: Variables asociadas al movimiento del brazo robot en Factory IO

Tras medir el valor en voltios de las posiciones clave a las que se tiene que desplazar el brazo en un ciclo de funcionamiento normal, se va a realizar su conversión a digital. En primer lugar se abre una nueva sección en Unity Pro. En esta nueva sección se va a asociar las variables booleanas del sistema real con el valor analógico correspondiente. En la siguiente tabla se recoge la relación entre las distintas variables.

Variable digital	Valor	Variable analógica	Valor
Mover_dcha	1	Mov_X	7.3V
Mover_izda	1	Mov_X	3.1V
Mover_atras	1	Mov_Y	0,9V
Mover_adelante	1	Mov_Y	10V
Bajar_brazo	1	Mov_Z	9V
Bajar_brazo	0	Mov_Z	0V

Tabla 6: Conversión analógica-digital de las variables asociadas al robot en Factory IO

Para frenar el robot, por ejemplo ante una **parada de emergencia**, en esta sección se establece la condición de que si simultáneamente las dos variables que dirigen el movimiento en un eje son 0 el valor de la variable Mov es igual al de la variable Pos.

5.1.2. Sensores

Además del problema relativo al movimiento, el brazo de Factory IO también presenta problemas con ciertos sensores. En concreto, el brazo no incorpora los sensores que indican si el brazo está arriba o abajo, ni el sensor que detecta vacío indicando si el objeto se ha cogido por la pinza o no.

En primer lugar vamos a resolver el problema asociado con la posición del brazo en el eje Z. Conociendo el valor en voltios del brazo cuando esta abajo y está arriba, vamos a crear una condición en la cual, si el valor de *Pos_Z* es superior o igual al valor establecido cuando la variable bajar_brazo se pone a 1, el sensor brazo_abajo se pone a 1. Si por el contrario, el valor de *Pos_Z* es inferior al valor establecido cuando la variable bajar_brazo se pone a 0, el sensor brazo_arriba se pone a 1. Para valores intermedios ambos sensores permanecen en 0. La variable *Pos_Z* se renombra a *Situación_brazo*.

El segundo problema está asociado con el agarre de las bases. El robot de simulación posee un sensor que detecta si está en contacto con otro objeto. No lo podemos emplear como sensor de vacío porque para poder agarrar el objeto una vez detectado debe pasar un tiempo, sino el agarre es defectuoso. Además, en el momento de dejar la base nunca se haría 0 porque estaría todavía en contacto. Para solucionar este problema se crean dos nuevas variables de tipo entero que van a actuar como temporizadores.

Los temporizadores se van a emplear para programar la entrada de vacío en caso de estar trabajando en modo de simulación. Al no tener una señal en el brazo que nos indique si se ha cogido o no una pieza, se van a establecer unos tiempos mínimos que aseguren el procedimiento.

Para programar la entrada de vacío se ha definido un vector de 3 estados:

- **Estado 0**: No se está efectuando ninguna acción. Se va a emplear para restablecer los contadores a 0.
- **Estado 1**: Se está atrapando una pieza. Lleva asociado un contador que cuando llega al valor establecido activa la entrada de vacío y vuelve al estado 0.
- **Estado 2**: Se está soltando una pieza. Lleva asociado un contador, distinto al del estado 1, que cuando llega al valor establecido desactiva la entrada de vacío y vuelva al estado 0.

Los sucesos que provocan un cambio de estado de 0 a 1 o a 2 se corresponden con el momento exacto en el que el robot coge o deja una pieza, y para conocer este momento nos vamos a apoyar en los sensores que nos indican la posición del brazo.

5.2. Depósitos de gravedad

En la estación real las bases se almacenan en depósitos de gravedad. Sin embargo, en Factory IO no existen este tipo de depósitos, y además, la sección de los expulsores es mayor y en caso de tener bases apiladas arrastraría más de una. Para suplir estos depósitos se instala una generación de bases automática, consistente en un cubo que siempre que se pone a 1, si no hay nada dentro de su volumen, genera una nueva base.

Este sistema de generación de bases obliga al empleo de variables que no existen en la estación real, lo que implica que no pueden ir en el código de la estación. Para llevar a cabo el control de la generación de bases se crea una **sección nueva** llamada *Simulación_de_la_planta* que se detallará en el *apartado 8.6* de esta memoria.

El sistema implantado genera bases de manera ilimitada, por tanto, se trata de depósitos que nunca se quedarían sin bases. Para simular este supuesto debemos actuar en el control. Se crean dos variables auxiliares, *bases_dcha/bases_izda,* que van a contar el número de bases generadas en cada lado. Establecemos un número máximo de bases a generar por cada depósito, de tal forma que, por medio de una nueva condición, si el valor de una de las dos variables es superior al valor establecido, no se genera base y se pone a uno la variable *sin_bases_dcha/izda* correspondiente.

Gracias a que se consigue simular el supuesto de que un depósito se quede sin bases, se podrá programar el control ante una falta de bases y comprobar en la simulación que se ha efectuado correctamente.

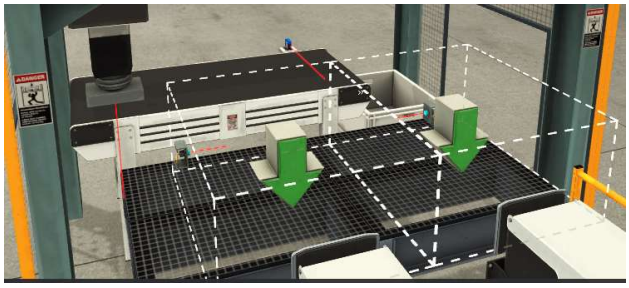


Figura 14: Depósitos de gravedad de la estación simulada

6. MAPEADO DE SEÑALES

En primera instancia, el objetivo era, además de conseguir un control único, establecer unas direcciones de memoria para las variables de simulación iguales a las de las variables reales. Al no poder llevarse a cabo, se ha realizado un mapeo de las señales. En el código de control se va a trabajar con variables únicas sin dirección de memoria. A través del mapeo de señales se les otorga un valor a estas variables en función de si estamos trabajando con la estación real o la simulada.

En primer lugar, en la tabla de variables elementales vamos a copiar de nuevo las entradas y salidas añadiéndoles el apellido real, si la variable tiene como dirección de memoria la asociada a la estación *real*, o el apellido *fact*, si tiene como dirección de memoria la asociada a la estación simulada. Definimos una última variable booleana, *simulando*, que se pondrá a 1 cuando estemos trabajando en Factory IO, y a 0 cuando trabajemos con la estación real.

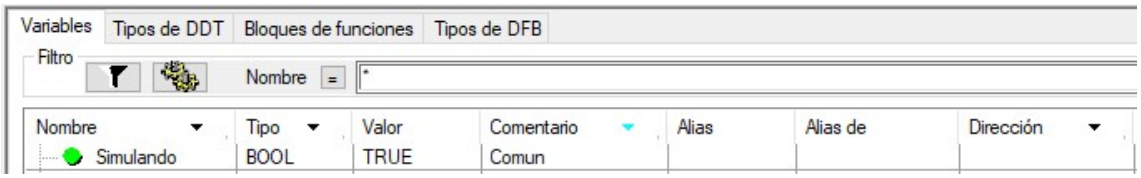


Figura 15: Variable para indicar el tipo de estación de trabajo

Se crean dos secciones nuevas, una anterior al código de control para mapear las entradas, y otra posterior para mapear las salidas. En la primera sección vamos a recoger todas las entradas del sistema, y en función del valor de simulando, les damos el valor de las variables reales o simuladas. En esta sección incluimos lo referente a los sensores de *brazo_arriba*, *brazo_abajo* y la programación de la entrada de *vacío*, añadiendo la condición de que el valor de *simulando* sea igual a 1. En la segunda sección vamos a igualar las salidas reales/simuladas a las salidas del código en función del valor de simulando. Añadimos lo referente al movimiento analógico del robot, explicado en el apartado de *control de la estación simulada*.

Con este sistema, permitimos que los alumnos puedan trabajar con variables únicas a la hora de realizar el código de control, independientemente de si están trabajando con Factory IO o con la maqueta real. Una vez hayan realizado su línea de código deberán ir a la tabla de variables elementales y modificar el valor de *simulando* en función de donde estén probando el programa.

Nombre	Tipo	Valor	Comentario	Alias	Alias de	Dirección
Marcha	EBOOL		Estacion6			
Marcha_fact	EBOOL		Estacion6			%m108
Marcha_real	BOOL		Estacion6			%mw37.3

Figura 16: Señales mapeadas en Unity Pro

7. ESTACIÓN DE PALETIZACIÓN Y ALMACENAJE

En este apartado se va a explicar todos las aspectos referentes a la segunda estación. Esta segunda estación se diseña con el objetivo de darle una continuidad al proceso iniciado en la primera estación en forma de almacenamiento. El control que se va a ejecutar sobre la nueva estación es independiente de la primera. Sin embargo, para lograr una sucesión de hechos secuencial habrá que aplicar una serie de restricciones entre ambas para que no se tengan que realizar esperas.

Con el objetivo de dar una solución a procesos productivos de gran tamaño, para los cuales no es eficiente la representación de toda la célula en una única escena de Factory IO, esta estación se construirá en una escena diferente y a la hora de ejecutar el código, la simulación se realizará con dos ordenadores distintos, debido a que Factory IO no deja abrir dos escenas simultáneamente en un mismo ordenador.

7.1. Estructura

La estación que se diseña consiste en la recepción de las bases depositadas por el brazo robot, su paletización mediante otro brazo robot y el depósito del pallet en un almacén. Para conseguir la estructura deseada nos vamos a apoyar en los siguientes elementos.

- **1 brazo robot de dos ejes**: Se va a encargar de la recoger la base y depositarla en el pallet.
- **1 mordaza**: Su función es la de atrapar la base en la posición óptima para que el brazo robot la pueda recoger.
- **1 barrera**: Se encarga de retener el pallet en la zona donde el brazo va a depositar la base.
- **1 cargadora**: Se encarga de recoger el pallet y almacenarlo.
- **1 almacén**
- **2 cintas transportadoras**: Una de ellas transportará la base hasta la mordaza y otra, será de rodillos, transportará el pallet a la cargadora.

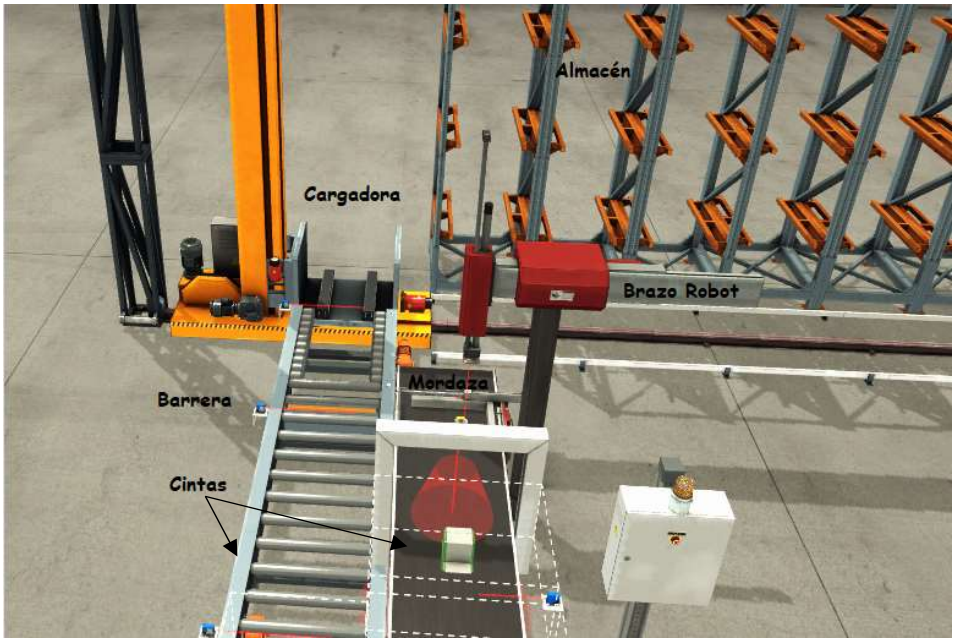


Figura 17: Estructura de la estación de paletización y almacenaje

Además de los elementos físicos, habrá que instalar unos sensores auxiliares para realizar el control. Estos sensores se detallan en la siguiente tabla:

Sensor	Tipo	Función
Palet_en_carga	Difuso	Indica que el pallet está en la zona de recepción de la base
Palet_en_cargadora	Difuso	Indica que el pallet está en la cargadora
Base_en_carga	Difuso	Indica que la base ha llegado a la mordaza
Cargadora_en_reposo	Difuso	Indica que la cargadora está en su posición de reposo
Tipo_base	Sensor de visión	Indica si tenemos una base verde o azul

Tabla 7: Sensores de la estación de paletización y almacenaje

Una vez se han seleccionado todos los elementos que conforman la estructura de la estación, se procede a realizar el control de la misma.

7.2. Control de la estación

El control de la estación se va a realizar como una estación independiente, para posteriormente realizar las conexiones entre ambas con el objetivo de que las dos estaciones se coordinen entre ellas.

En este apartado se va a explicar el funcionamiento deseado y el control realizado sobre la estación sin entrar en la conexión con la primera estación, que se explicará más adelante.

7.2.1. Funcionamiento deseado

La estación comienza con la llegada de la base a la cinta destino de la estación 1. A partir de este hito, van a suceder los siguientes eventos.

- La base va a ser transportada a la zona de carga, donde será atrapada por una mordaza y recogida por el brazo robot. En paralelo, el pallet va a ser transportado a la zona donde el brazo robot va a depositar la base.
- Una vez el pallet y la base llegan a sus respectivas zonas de carga, el brazo robot recogerá la base y la depositará sobre el pallet.
- Con la base ya sobre el pallet, va a ser transportado a la cargadora.
- Cuando el pallet este sobre la cargadora se va a depositar en el almacén. Si se trata de una base verde se depositará en la zona inferior, y si se trata de una base azul en la zona superior.
- En caso de pulsar la seta de emergencia se deberá encender la baliza y detener todos los sistemas.
- Si el almacén alcanza su máxima capacidad, se encenderá la baliza de emergencia y una vez se haya vaciado por completo, pulsando el botón de rearme se indicará que puede iniciarse un nuevo ciclo.

7.2.2. Red de Petri

Al igual que con la primera estación, se ha realizado la Red de Petri correspondiente con el control deseado del almacén.

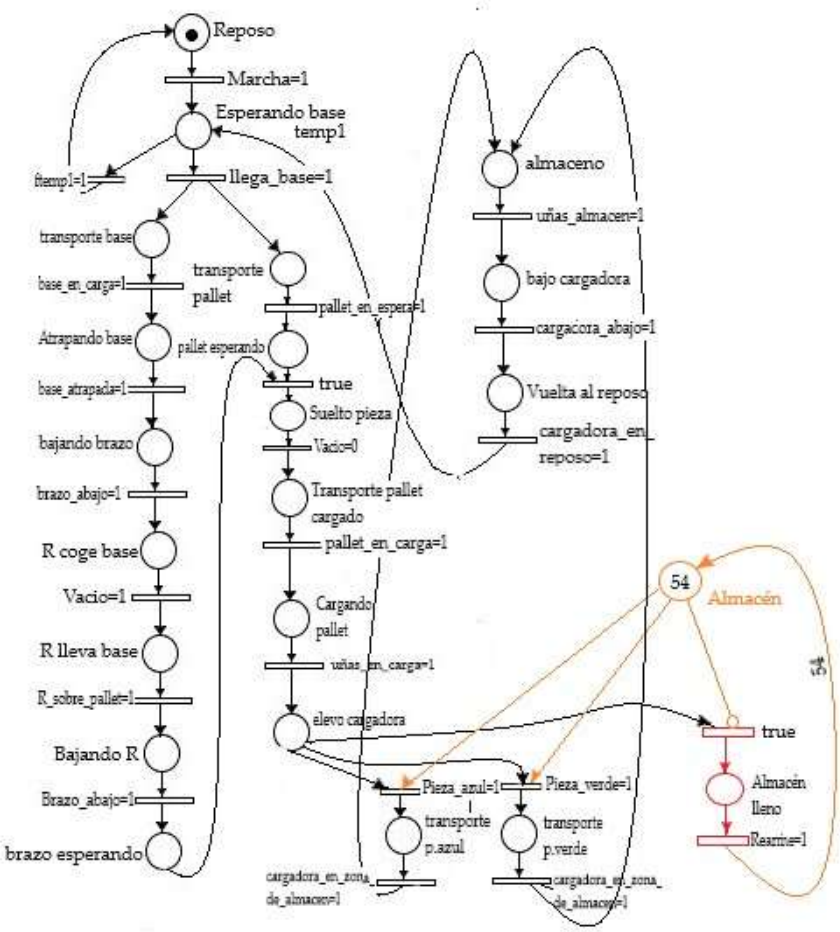


Figura 18: Red de Petri de la estación de paletización y almacenaje

En la Red de Petri podemos ver como el funcionamiento inicia con la puesta en marcha de la estación. A partir de este momento el sistema funciona de manera constante. El temporizador se instala para que en caso de que se produzca una espera con una duración por encima de lo habitual, el sistema vuelva al reposo asumiendo que la estación principal no está inyectando más bases al sistema.

7.2.3. Lenguaje SFC

El control, al igual que con la primera estación, se va a realizar en Unity Pro. En este caso, vamos a emplear un lenguaje de programación distinto al empleado anteriormente para así poder compararlos y sacarle más partido a todas las funcionalidades que ofrece el software.

El lenguaje SFC consiste en un esquema de control muy visual. Se trata de una combinación de pasos y transiciones. Los pasos son los distintos eventos que ocurren, y llevan asociados los valores para los actuadores del sistema. Por otro lado, las transiciones son los sucesos que implican un cambio de estado y van asociadas a los sensores.

Las ventajas de este lenguaje de programación frente al texto estructurado son, que por un lado es más visual y fácil de comprobar, en caso de error, dónde está fallando el sistema. Por otro lado, si conocido el funcionamiento deseado se realiza una Red de Petri a modo de esquema de control, este tipo de lenguaje se basa en trasponer esta red a ordenador. Como inconveniente, en aplicaciones complejas, es posible que las transiciones no te las de un solo

sensor o que las acciones no vayan asociadas solo al cambio de estado de un actuador, esto obliga a definir secciones en otro lenguaje de programación para asociar a transiciones y acciones.

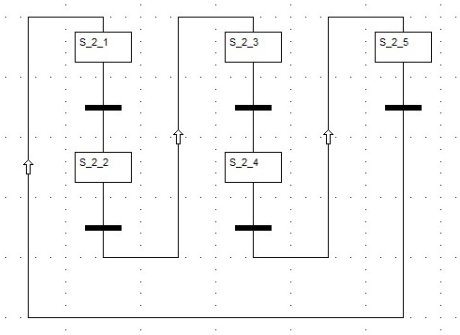


Figura 19: Modelo de código de control en lenguaje SFC

7.3. Conexión entre estaciones

Las dos estaciones diseñadas tienen una sección independiente para su control. Sin embargo, el objetivo de la segunda estación era que sirviera de ampliación a la primera, por tanto, que la cinta transportadora final fuera la inicial en la estación de almacenaje. Las dos estaciones al ser independientes no tienen el mismo tiempo de ciclo, esto implica que si se hace el control totalmente independiente, la estación con un tiempo de ciclo menor va a presionar a la lenta provocándole una acumulación de tareas que no es eficiente para el sistema.

En nuestro caso, la estación de paletización y almacenaje tiene un tiempo de ciclo mucho mayor al de la primera estación, en gran parte debido a la velocidad con la que la cargadora almacena los pallets. En este tipo de situaciones, la solución más sencilla es restringir el inicio de ciclo de la primera estación con el final de la siguiente, lo que supondría un tiempo de almacenamiento igual al tiempo de ciclo de la estación lenta.

Para reducir este tiempo de almacenamiento al mínimo se va a analizar el tarea con mayor tiempo de duración. Esta tarea es la de transporte, almacenamiento y vuelta al reposo de la cargadora, cuyo tiempo de ejecución viene impuesto por Factory IO y no se puede reducir mediante el control. El objetivo será programar el control de tal forma que, cada vez que la cargadora llega a su posición de reposo tenga un pallet cargado esperándola. Los ajustes realizados en el control son:

- Se crea una variable auxiliar que se pondrá a 1 cuando la estación de almacenaje esté lista para la recepción de una nueva base, indicando así a la primera estación que ya puede depositar la base. Esta variable se denominará estación_2.
- Se establece una segunda zona de espera, además de la espera que realiza la estación 1 (con el brazo robot sosteniendo la base sin depositarla en la cinta). Esta zona de espera se determina como la zona que garantiza que, una vez se pone en marcha de nuevo las estaciones, el pallet cargado llegue antes a la zona de carga que la cargadora a su zona de reposo.
- Conocidas las dos zonas de espera, se elige la variable de entrada que pondrá fin a las esperas. Esta variable de entrada se denomina almacenado y nos indica que el pallet se ha depositado en el almacén. Cuando esta variable se pone a 1, el robot deja la base en la cinta y la barrera desciende dejando pasar al pallet cargado hacia la zona de carga.

Al realizar la conexión entre las dos estaciones ha sido necesario introducir en la línea de código de la primera estación variables de entrada que se corresponden con salidas de la segunda estación para conseguir la sincronización entre ambas. Esto provoca que la simulación por separado de la primera estación no funcione. Para poder diferenciar entre la simulación de las dos estaciones en conjunto o la primera de manera individual, se crea una variable auxiliar llamada *DosEstaciones*. El usuario pondrá la señal a 0 si se está simulando la primera estación en solitario y a 1 si está simulando ambas de forma simultánea.

Se ha realizado un vídeo con el funcionamiento de las dos estaciones en conjunto en el cual se puede apreciar las dos zonas de espera, y como cada vez que la cargadora llega al reposo se encuentra con un nuevo pallet para almacenar. Con este ajuste se logra un tiempo de ciclo igual al de la tarea más larga. El enlace al vídeo realizado lo encontramos en el *apartado 9* de la memoria.

7.4. Escenas independientes

Los procesos productivos automatizados se controlan con métodos como el desarrollado en este trabajo. Su elevado tamaño hace que no sea eficiente tener el control de todo el proceso en una misma sección de código. En caso de querer llevar estos procesos al software de simulación Factory IO, tal y como se ha realizado en este proyecto, es posible que por tamaño, no quepa todo el proceso en una única escena. También existe la posibilidad de que se quiera centrar la atención en una estación concreta y por comodidad se busque tener las estaciones en escenas independientes y ejecutar la simulación en ordenadores distintos, uno por cada escena.

Figura 20: Esquema de funcionamiento deseado

Mediante la conexión de Unity con el PLC de simulación se crea una red Modbus. Durante la realización de este proyecto, la estación en Factory IO se conectaba a la red Modbus a través de la dirección IP local, 127.0.0.1. Para conectar una segunda escena de Factory IO al servidor Modbus, debemos establecer como dirección en el software de simulación, la IP del ordenador que está ejecutando Unity Pro. De esta forma, se logra conectar la simulación del segundo ordenador a la red Modbus que ha creado el primer ordenador, y por tanto, el PLC podrá leer las entradas que genere la simulación, y la simulación las salidas que le envíe el PLC.

Durante la realización de esta comunicación han aparecido problemas de conexión entre ordenadores, ya que el ordenador con Unity Pro denegaba la conexión al segundo ordenador. Los problemas venían provocados por la protección antivirus del ordenador, por lo que para realizar la conexión ha sido necesario desactivar el Firewall.

Al crear dos escenas distintas debemos tener en cuenta una serie de consideraciones. En primer lugar, el final de la primera estación no está conectado físicamente con el principio de

la segunda, es decir, la cinta transportadora de 1 no es la cinta transportadora de 2. Esta situación nos va a obligar a instalar dos emisores para generar los dos tipos de bases y una variable auxiliar para indicarle a la segunda estación que la primera ha dejado una base. Por último, la estación de almacenaje deberá leer de la primera estación qué base se ha depositado para saber cuál generar. En la siguiente tabla se recogen las variables auxiliares generadas.

Variable	Descripción
Llega_pieza	El brazo de E1 deja la base en la cinta
Genera_base_azul	El emisor genera una base azul en E2
Genera_base_verde	El emisor genera una base verde en E2
Base_dcha/izda	Son las utilizadas en el control de E1 pero es necesaria su lectura

Tabla 8: Variables auxiliares empleadas para el control en escenas independientes

8. SECCIONES GENERADAS

En este apartado se van a recopilar las secciones de programa que se han generado para el control de las dos estaciones que ya se han ido introduciendo a lo largo de la memoria. El proyecto cuenta con un total de 6 secciones de las cuales dos están asociadas con el control de las estaciones, otras dos con el mapeo de señales, una con los estados del robot y la última con el control de los emisores de Factory I/O. Las secciones generadas para transiciones y pasos en el lenguaje SFC no se van a entrar a valorar.

Cabe destacar que el orden de estas secciones sigue el orden de lectura de un autómata programable, el cual lee las entradas que le llegan de la simulación, ejecuta el código de control y establece el valor de las salidas que envía a la simulación. Por tanto, en el caso de la estación 1 donde se ha realizado un mapeo de entradas y salidas, el orden es *Mapeo de entradas > Control de la estación 1/Estados del brazo robot > Mapeo de salidas.*

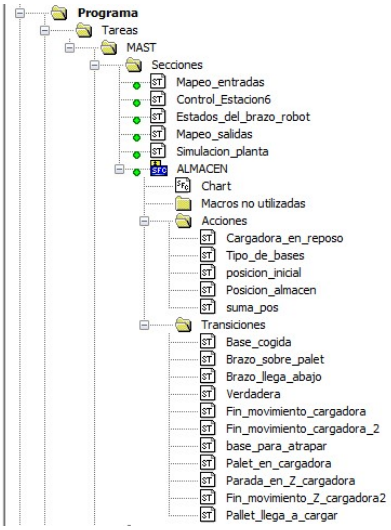


Figura 21: Secciones generadas en el control

Los códigos de control elaborados para cada una de las secciones se recogen en el Anexo IV.

8.1. Control Estación 6

Esta sección es la que los alumnos deberán realizar durante el desarrollo de la práctica. En ella se emplean variables únicas, es decir, son independientes del trabajo con la estación real o el trabajo con la simulación. El alumno únicamente deberá establecer el valor de la variable simulando en función de donde esté trabajando.

La redacción del control en esta sección se ciñe a lograr el funcionamiento deseado que se pide en el guion de la práctica. Las correcciones necesarias para solventar las diferencias entre la estación simulada y la estación real se realizan en otras secciones.

8.2. Estados del brazo robot

La función de esta sección es facilitar la comprensión del código de control de la estación 1. Como se ha comentado anteriormente, el brazo robot se ha controlado mediante un vector de estados. En función del estado, el valor de las variables características del robot cambia.

En la redacción del código de control se establecen los estados del robot y los sucesos que desencadenan los cambios de estado. Para que el control sea completo es preciso definir el

valor de las variables en cada estado. Esta definición se puede realizar en la misma sección en la que se realiza el control de la estación, pero se ha optado por llevar la descripción de los estados a una nueva sección para de esta manera reducir la extensión del programa principal y facilitar la comprensión tanto del programa como de los estados.

8.3. Mapeo de entradas

En el apartado 6 se ha explicado la necesidad de realizar un mapeo de señales para poder trabajar con variables únicas en el código de control principal. En primer lugar se deben mapear las entradas, que hacen referencia a los sensores de la estación.

En esta sección se lleva a cabo el procedimiento explicado en el *apartado 6* de esta memoria. Además, se van a solventar los problemas referentes a las entradas, provocados por la diferencia de sensores entre estaciones. Estos sensores son los que nos indican si el brazo está arriba o abajo y el sensor de vacío. La forma de resolver este problema se explica en el *apartado 5.1.2.*

8.4. Mapeo de salidas

De la misma manera que se han mapeado las entradas, debemos mapear las salidas, las cuales hacen referencia a los actuadores de la estación.

En esta sección también se van a resolver los problemas referentes a la diferencia de actuadores entre estaciones. El problema en este caso es que los actuadores del brazo robot en la estación real son digitales y en la estación simulada son analógicos. La forma en la que se ha resuelto esta problemática esta explicada en el *apartado 5.1.1.*

8.5. Control almacén

Esta sección se corresponde con el control del almacén. La sección se compone a su vez de otras subsecciones que se emplean para definir acciones de paso y variables asociadas a transiciones más complejas.

En esta sección podemos encontrar las variables que comparten entre estaciones como condicionante de alguna transición, pero no encontramos la parte correspondiente a la generación de piezas por parte de los emisores. Esto se realiza de esta manera, para que en caso de que la estación fuera real y le precediera la primera estación diseñada, el control fuera perfectamente válido.

En esta sección de control se establece la programación de la variable DosEstaciones, en la que si el valor de esta es 0, no se inicia el ciclo del almacén, y por tanto, no condiciona el funcionamiento de la primera estación.

8.6. Simulación de planta

Las dos secciones correspondientes al control de estaciones emplean únicamente expresiones válidas para estaciones reales. Sin embargo, ambas estaciones de simulación emplean emisores, para los cuales hay que introducir una orden de generar pieza para la aparición una nueva base o pallet en el sistema. Para poder realizar códigos de control principales que empleen instrucciones reales y a la vez poder llevar a cabo la simulación en el software Factory I/O, se crea esta sección.

En esta sección se van a dar las órdenes de generación de piezas en las distintas estaciones en el momento en el que corresponda, simulando que se encuentran en un depósito (en la estación 1) o que llegan de la cinta de la estación 1 (almacén). Además, se va a aprovechar para solventar el problema debido a la no presencia de depósitos de gravedad en Factory I/O. La resolución a este problema se detalla en el *apartado 5.2.*

9. VALIDACIÓN DE LOS RESULTADOS OBTENIDOS

En este apartado se van a adjuntar una serie de vídeos que muestren como el funcionamiento de las estaciones simuladas, tras aplicar el código de control, cumplen con el funcionamiento deseado, expuesto a lo largo de esta memoria.

En primer lugar, se adjunta un vídeo con el funcionamiento de la estación 1. Esta estación es la que los alumnos deberán programar durante la realización de las prácticas de la asignatura.

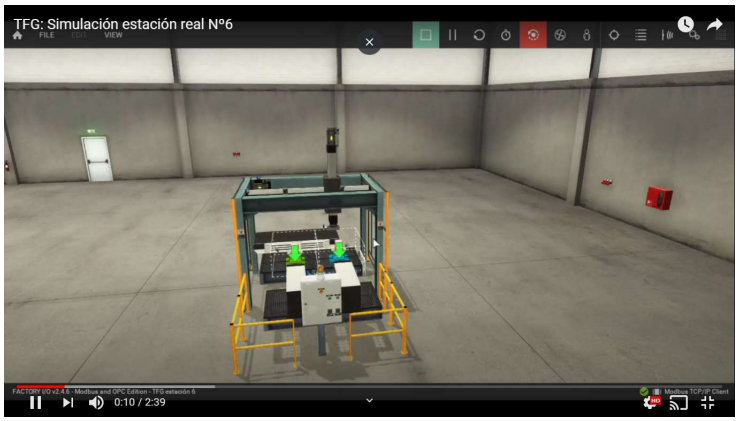


Figura 22: Vídeo de la simulación de la estación Nº6.
Enlace: https://www.youtube.com/watch?v=q5CTgkxdBzE

El segundo vídeo adjunto, se corresponde con el funcionamiento de las dos estaciones (estación 1-almacén) simultáneamente. Con este vídeo se quiere demostrar como el control realizado sobre ambas hace que ambas estaciones trabajen de forma coordinada sin que la estación más veloz apriete a la más lenta. El vídeo se encuentra dividido en dos ya que cada mitad corresponde con un ordenador diferente.

Figura 23: Video de la simulación conjunta de dos escenas de Factory IO.
Enlace: https://www.youtube.com/watch?v=uwkgePS8Sp0

En caso de problemas con la visualización, ambos vídeos se encuentran en el repositorio[1] junto con los archivos de los distintos programas desarrollados en este trabajo fin de grado.

[1] Enlace al repositorio: https://drive.google.com/drive/folders/18GP0gsyuDLjF7-UdwhRug5hBuYYeRm-0?usp=sharing

10. CONCLUSIONES

El trabajo fin de grado se ha desarrollado con el fin de cumplir una serie de objetivos principales: reducir el tiempo de realización de las prácticas y aumentar la seguridad del laboratorio reduciendo el número de pruebas que realiza el alumno sobre la estación real, facilitar la comprensión de los sistemas de eventos discretos y desarrollar una simulación coordinada de dos sistemas que se encuentran en ordenadores diferentes aportando una solución a aquellas células que, por necesidad o comodidad, requieren más de una escena de simulación. Una vez finalizado el proyecto se va a valorar los resultados obtenidos y el grado de cumplimiento de estos objetivos.

En el software de simulación Factory I/O se ha llevado a cabo la construcción de una estación ficticia cuyas partes físicas tienen un alto grado de coincidencia con los elementos de la estación real presente en el laboratorio L0.06 del edificio Ada Byron.

Las limitaciones del software de simulación en cuanto a variedad de componentes se han subsanado empleando elementos que fueran similares a los de la estación real. Apoyándonos en el control hemos conseguido que estos elementos, en un principio distintos, se comporten de la misma manera que en la estación real. Un ejemplo de este tipo de ajuste lo encontramos en los actuadores del brazo robot o en los depósitos de gravedad.

En el software de programación Unity se ha elaborado un código de control único para ambas estaciones. Previo a la implementación del código, se han elaborado las Redes de Petri necesarias para explicar el funcionamiento deseado de la estación en base al fundamento teórico de las mismas. Una vez elaboradas, se implementa el código. Para lograr que este fuera único se han creado secciones de apoyo que controlasen los elementos cuya actuación fuera distinta a los de la estación real. El control que se programa ha conseguido simular con éxito el supuesto de que un depósito se quede sin bases, a pesar de no disponer en el software de simulación de depósitos de gravedad.

El control único implica el empleo de variables únicas, es decir, no diferenciar entre variables de la simulación y variables reales. Durante la realización del proyecto se ha visto como el software de simulación no permitía almacenar las variables de la forma en que se habían almacenado en la estación real, con bits de una variable WORD. Se ha elaborado un mapeo de señales que ha permitido el empleo de variables únicas ya que es el propio programa el que, en función de una variable auxiliar que indica con que estación se está trabajando, establece a las variables la dirección de memoria correspondiente.

La segunda estación elaborada se comporta como un almacén automatizado y permite dar continuidad y un final de proceso al depósito de las bases en una cinta transportadora, que realiza la primera estación. El control que se ha diseñado para la misma cumple con el funcionamiento deseado y su comunicación con la estación precedente permite reducir los tiempos de espera.

Una vez valorados los resultados obtenidos, podemos concluir que el trabajo realizado y explicado en la presente memoria logra la implementación en un software de simulación de una estación real, que permitirá al alumno probar sus códigos primero en el ordenador, evitando así posibles daños en la estación por códigos defectuosos y la aglomeración de alumnos en la estación real para probar sus controles. Una vez obtenido el código de control que implementa el funcionamiento óptimo de la simulación, el alumno podrá cargarlo en el PLC de la estación real sin necesidad de variar el código. Además, no deberá preocuparse por las diferencias entre

elementos ya que las secciones de apoyo permiten que la realización de un código para el funcionamiento deseado expuesto en el guion controle ambas estaciones de manera óptima.

El diseño de la segunda estación cumple con los requerimientos de almacén, y el código implementado, gracias a una comunicación eficaz entre ambas estaciones, consigue reducir los tiempos de espera, haciendo el tiempo entre pallet almacenado mínimo.

Mediante la implementación de las estaciones en ordenadores independientes se consigue dar una solución óptima a aquellos procesos, que no se pueden realizar en una única escena, a partir de un procedimiento sencillo y eficaz.

El estudio realizado para la conexión de dos escenas independientes de Factory IO, se podría realizar para la conexión de dos proyectos Unity, de forma que, el control de cada estación se encontrara en proyectos independientes, logrando un control distribuido entre estaciones. De esta manera se conseguiría, no solo disponer de las estaciones en escenas independientes, sino también de los códigos de control en archivos distintos.

El sector del control y la automatización de procesos productivos se está desarrollando estos últimos años de forma exponencial. Esto implica que el uso de tecnologías que hoy en día son eficientes, el día de mañana puedan quedar obsoletas. Se podría llevar a cabo un trabajo fin de grado que estudiará las tecnologías empleadas en la célula de fabricación del laboratorio y planteara mejoras que aumentaran el rendimiento de la misma, aprovechando la descripción realizada sobre la estación 6 en esta memoria.

Por último, este trabajo fin de grado podría servir como punto de partida para alguien que se proponga llevar una célula de fabricación real, compuesta por varias estaciones, al software de simulación, y a partir de ahí, al disponer de recursos compartidos entre estaciones, realizar un estudio sobre la optimización del tiempo de ciclo de la célula.

11. BIBLIOGRAFÍA

[1] Manual Unity Pro. Universidad de León. [Internet]. Disponible en: http://lra.unileon.es/sites/lra.unileon.es/files/Documents/plc/Unity_Pro/Manuales_Unity/Manual_Unity.pdf

[2] Manual de referencia de Unity Pro. Universidad de León. [Internet]. Disponible en: http://lra.unileon.es/sites/lra.unileon.es/files/Documents/plc/Unity_Pro/Manuales_Unity/Unity_Manual%20de%20Referencia.pdf

[3] Manual de Factory IO. Factory IO, Real Games. [Internet]. Disponible en: https://docs.factoryio.com/

[4] Guía rápida Unizar. Centro Politécnico Superior de Zaragoza. [Internet]. Disponible en: http://automata.cps.unizar.es/post/documentos/grafcetunity.pdf

[5] Trabajo Fin de Grado "*Diseño de un sistema de control distribuido usando Factory IO y Codesys V3*". José Marín Sánchez. [Internet]. Disponible en: https://idus.us.es/bitstream/handle/11441/83982/TFG-1778-MARIN.pdf?sequence=1&isAllowed=y

[6] Tutorial Práctico Unity Pro 3.0-Modicon M340. Universidad de León. [Internet]. Disponible en: http://lra.unileon.es/sites/lra.unileon.es/files/Documents/plc/Unity_Pro/Manuales_Unity/Turorial_Unity.pdf

[7] Factory IO Simulación 3D de fábrica. Universidad de Anáhuac. [Internet]. Disponible en: https://www.anahuac.mx/mexico/noticias/Factory-IO-Simulacion-3D-de-fabrica

[8] Guion Práctica 4 "*Control de una célula de fabricación flexible*". ingeniería de Control, Grado en Tecnologías Industriales. Disponible en el ADD para alumnos matriculados en la asignatura del grado.

[9] Trabajo Fin de Grado *"Desarrollo de una planta virtual en Factory IO y control mediante PLC".* Fernando Grima Montesa [Internet] Disponible en: https://zaguan.unizar.es/record/85222/files/TAZ-TFG-2019-2900.pdf?version=1

12. ANEXOS

12.1. ANEXO I: Guía básica Factory I/O

12.1.1. Acerca de Factory IO

Factory IO es un software de simulación 3D, creado por Real Games, el cual nos permite construir y controlar procesos industriales en tiempo real. Las características que convierten a Factory IO en un software atractivo son sus 20 escenarios inspirados en aplicaciones industriales frecuentes y una librería con más de 80 componentes industriales que permiten crear escenas personalizadas al gusto del usuario. Los requerimientos mínimos para su uso son: Windows Vista o superior, Intel Core 2 Duo a 2GHz, 1GB de RAM y 500MB de disco duro.

12.1.2. Creación de una escena

En este apartado se va a describir el procedimiento que se ha seguido para la construcción de la parte física de las distintas estaciones desarrolladas en este trabajo.

En primer lugar, debemos crear una nueva escena. Tras ejecutar el programa, en la primera interfaz clicaremos en *New* y nos aparecerá una escena vacía de Factory IO. Esta escena se irá amueblando conforme vayamos introduciendo componentes.

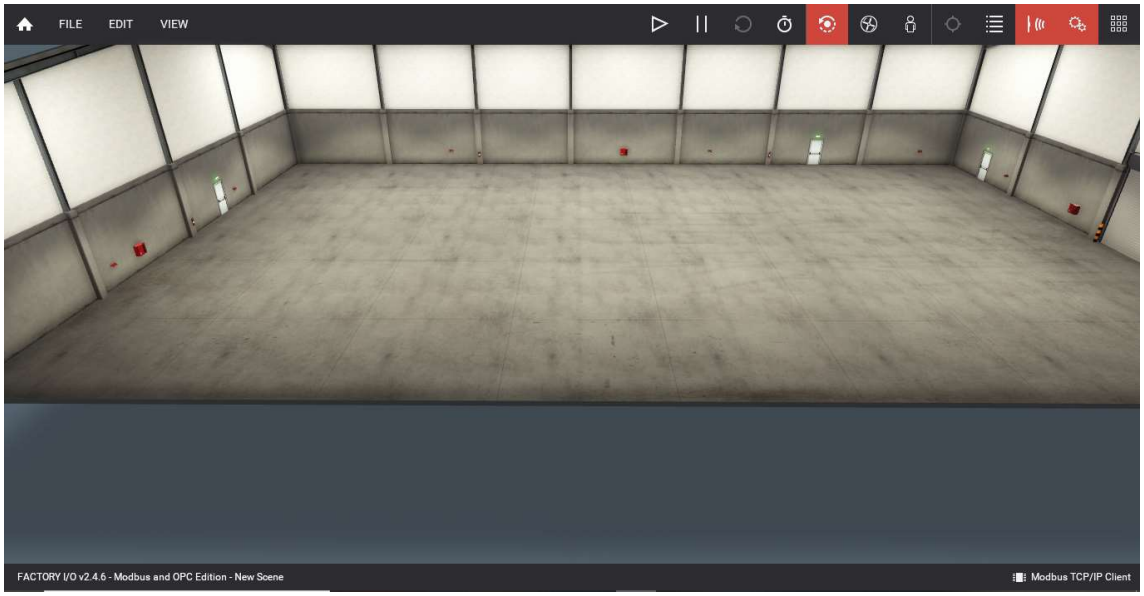


Figura 24: Escena vacía Factory IO

Para comenzar a introducir los distintos elementos que componen nuestra estación, debemos seleccionar el botón de la esquina superior derecha. Este botón abrirá la biblioteca de elementos que el software ofrece.

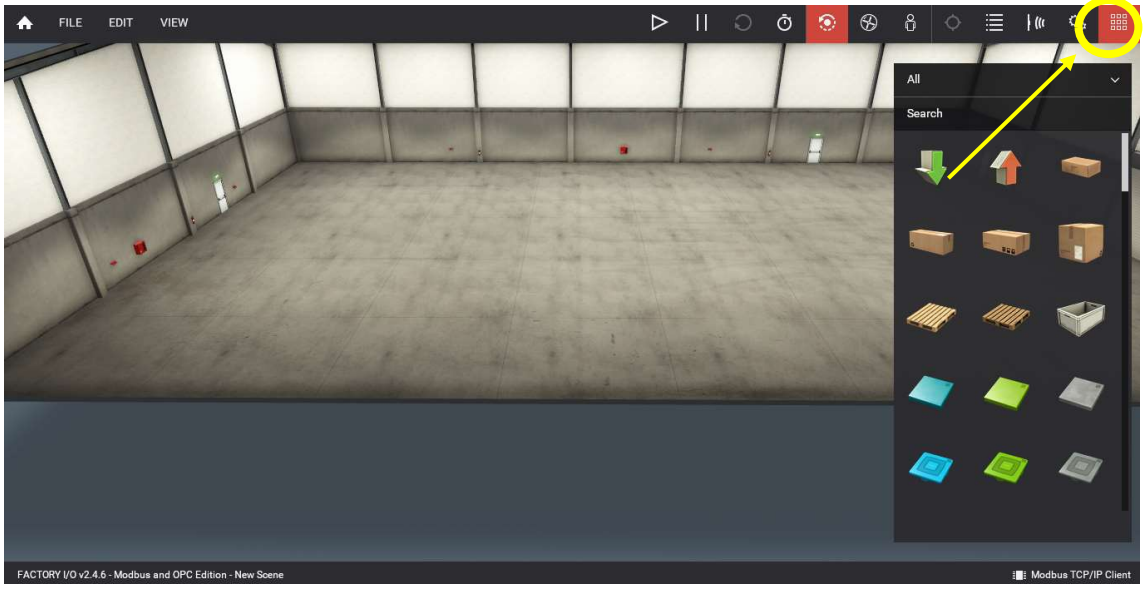


Figura 25: Cómo abrir la biblioteca de componentes en Factory IO

La biblioteca de Factory I/O divide los componentes en 8 grupos. Además, con el botón de *search,* podemos buscar el componente que buscamos si conocemos su nombre. Los grupos en los que se encuentran divididos los componentes son:

- **Artículos**: En este grupo encontramos las materias primas dentro de un proceso de producción, aquellos elementos que pueden ser manipulados por personal o maquinaria y son transportados a las distintas estaciones de un proceso productivo.
- **Piezas de carga pesada**: Elementos que sirven para el transporte de las materias primas más pesadas, como son las cajas o los pallets.
- **Piezas de carga ligera**: Elementos empleados para transportar piezas de poco peso. A diferencia de las piezas de carga pesada, este tipo de piezas nos permiten ejecutar tareas de forma rápida ya que trabajan a mayor velocidad.
- **Sensores**: En este grupo se encuentran los distintos sensores que ofrece Factory I/O que se pueden emplear para detectar la presencia de piezas, medir distancias o detectar el tipo de pieza.
- **Operadores**: Indicadores luminosos y botones propios de una botonera para controlar la puesta en marcha, la parada de emergencia, el reseteo… etc.
- **Estaciones**: Este grupo está formado por elementos típicos dentro de un proceso de producción cuya complejidad es mayor que la del resto de componentes debido a la gran cantidad de sensores que posee incorporados y de tareas distintas que puede realizar.
- **Dispositivos de advertencia**: Alarmas sonoras y balizas que indican algún tipo de fallo.
- **Pasarelas**: Piezas utilizadas para construir pasillos para trabajadores o aislar perímetros.

Además de todas estas piezas, seleccionando el grupo *all,* encontramos el emisor y el agente de mudanzas que sirven para generar o eliminar artículos.

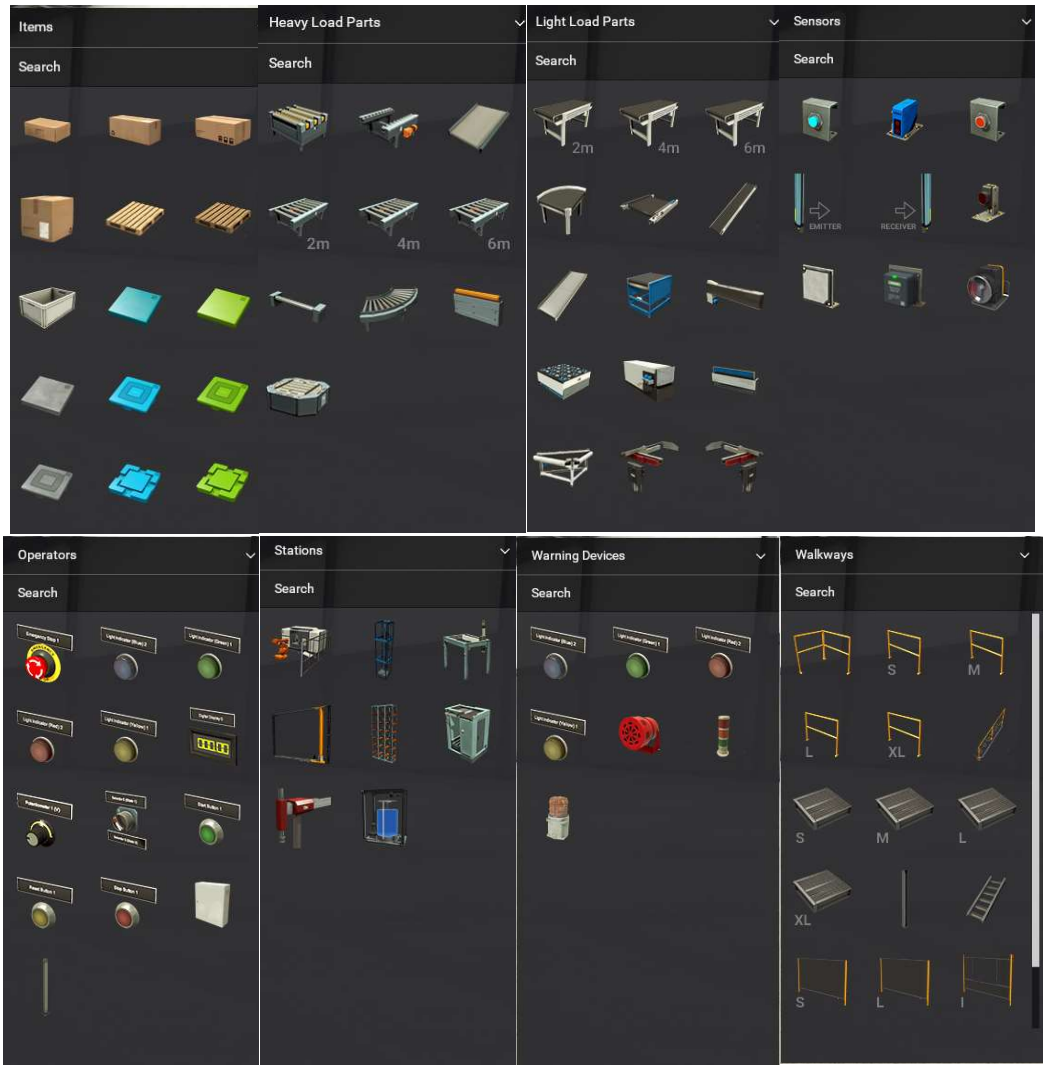


Figura 26: Grupos de componentes en Factory IO

Para comenzar a construir la estación arrastraremos la pieza a la escena vacía. Una vez se haya colocado la pieza, haciendo click derecho podemos ver las distintas opciones de disposición en el espacio que nos ofrece Factory I/O.

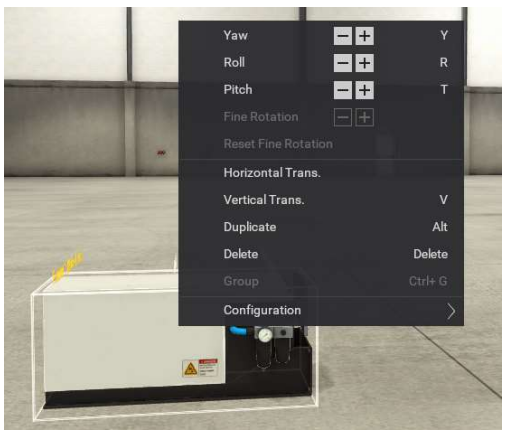


Figura 27: Opciones de manejo y configuración de objetos en Factory IO

En primer lugar, podemos rotar la pieza en torno a los tres ejes cartesianos en pasos de 90º. Además de la rotación, se permite la traslación tanto horizontal como vertical. La pieza se

puede duplicar y eliminar. Por último, la opción de configuración nos permite seleccionar el modo de funcionamiento de la pieza.

Los modos de funcionamiento dependen de la pieza en cuestión, a continuación se va a detallar los modos de funcionamiento que se han seleccionado en los distintos componentes empleados en este trabajo.

1. Expulsores

Los expulsores permiten una configuración monoestable o biestable. La diferencia entre estas dos es que el funcionamiento como monoestable tiene un único actuador que controla el movimiento hacia atrás y hacia delante, mientras que en biestable posee un actuador para cada movimiento. Además, permite el funcionamiento en digital, donde los actuadores son variables booleanas que en caso de establecerse a 1 realizan el recorrido máximo permitido, o en analógico, donde la salida del actuador varía entre 0 y 10V en función de la distancia recorrida y también permite establecer la velocidad del movimiento.

El funcionamiento seleccionado es monoestable, debido a que el actuador de la estación real es un cilindro de simple efecto, y digital, por ser las entradas y salidas de las estación real de tipo booleano. Este funcionamiento se caracteriza por:

Variable	E/S del controlador	Tipo	Descripción (puesta a 1)
Empujador #	Entrada	Booleana	Avanza
Empujador # (límite frontal)	Salida	Booleana	Empujador extendido
Empujador # (límite trasero)	Salida	Booleana	Empujador recogido

Tabla 9: Entradas y salidas del expulsor de Factory IO

2. Cintas de transporte

Tanto las cintas transportadoras como las cintas de rodillos pueden funcionar en modo digital, o bien con una salida booleana para poner en marcha la cinta, o bien con dos salidas booleanas para hacer girar la cinta en cualquiera de los dos sentidos. En el funcionamiento analógico, la cinta solo gira en un sentido y a través de una salida flotante se establece la velocidad.

El funcionamiento seleccionado es digital con una única salida booleana ya que el transporte de las distintas piezas tiene un único sentido.

Variable	E/S del controlador	Tipo	Descripción (puesta a 1)
Transportador de rodillos #	Entrada	Booleana	Transportador en marcha

Tabla 10: Entradas y salidas de la cinta de transporte de Factory IO

3. Estación escoger y colocar de 3 ejes

El brazo robot tiene 3 modos de funcionamiento. El primero de ellos es el funcionamiento digital, donde todas sus salidas son booleanas. El segundo es el modo analógico, donde las variables referentes al movimiento del robot y la posición que ocupa en el espacio son variables flotantes entre 0 y 10V. El último modo de funcionamiento es una mezcla entre digital y analógico donde, a las variables del modo analógico se añaden dos nuevas variables booleanas que nos indican si el robot se está moviendo en el plano XY y en la dirección Z.

Como se ha comentado en la memoria del trabajo, no se puede emplear el funcionamiento digital por el tipo de movimiento que implica en el brazo, por ello se selecciona el movimiento digital y analógico. Estas son sus características:

Variable	E/S del controlador	Tipo	Descripción
Ajuste de Pick&Place # X	Entrada	Flotante	Posición en X (0-10V)
Ajuste de Pick&Place # Y	Entrada	Flotante	Posición en Y (0-10V)
Ajuste de Pick&Place # Z	Entrada	Flotante	Posición en Z (0-10V)
Elegir y colocar # C	Entrada	Booleana	Gira la pinza
Pick&Place # (Agarrar)	Entrada	Booleana	Activa ventosas
Pick&Place # Posición X	Salida	Flotante	Posición actual en X
Pick&Place # Posición Y	Salida	Flotante	Posición actual en Y
Pick&Place # Posición Z	Salida	Flotante	Posición actual en Z
Pick&Place # (Moving-Z)	Salida	Booleana	Moviéndose en Z
Pick&Place # (Moving XY)	Salida	Booleana	Moviéndose en plano XY
Pick&Place # (Caja detectada)	Salida	Booleana	Detecta un artículo
Pick&Place # (Limite C)	Salida	Booleana	Pinza en límite angular

Tabla 11: Entradas y salidas de la estación del brazo robot de 3 ejes de Factory IO

4. Estación escoger y colocar de 2 ejes

Los modos de funcionamiento de la estación son análogos a los de la estación de 3 ejes. La diferencia a la hora de seleccionar el modo es que no tenemos la restricción de que esta estación pertenezca a una estación real, lo que implica que, por comodidad, se ha seleccionado el funcionamiento digital. Este funcionamiento digital se caracteriza por:

Variable	E/S del controlador	Tipo	Descripción (puesta a 1)
Pick&Place # Z	Entrada	Booleana	Muévete en Z
Pick&Place # X	Entrada	Booleana	Muévete en X
Pick&Place # Girar CW	Entrada	Booleana	Rota sentido horario
Pick&Place # Girar izquierda	Entrada	Booleana	Rota sentido antihorario
Pick&Place # Pinza CW	Entrada	Booleana	Gira pinza horariamente
Pick&Place # Pinza CCW	Entrada	Booleana	Gira pinza antihoraria
Pick&Place # Agarre	Entrada	Booleana	Activa ventosa
Pick&Place # Moving X	Salida	Booleana	Moviéndose en X
Pick&Place # Moving-Z	Salida	Booleana	Moviéndose en Z
Pick&Place # giratorio	Salida	Booleana	Girando el brazo
Pick&Place # Pinza giratoria	Salida	Booleana	Girando pinza
Pick&Place # Detectado	Salida	Booleana	Detectando un artículo

Tabla 12: Entradas y salidas del brazo robot de dos ejes de Factory IO

5. Grúa cargadora

La cargadora tiene 4 modos de funcionamiento. La cargadora se desplaza a lo larga de X y Z y tiene guardadas en su memoria 54 posiciones distintas en el espacio. En el funcionamiento digital introducimos la posición destino mediante el número en formato digital. Por otro lado, en el funcionamiento analógico, se introduce la posición destino a través de dos salidas flotantes, una para el eje X y otra para el eje Z. Al igual que con el brazo robot, tenemos un tercer modo de funcionamiento mixto (digital-analógico) que posee las mismas variables que el funcionamiento analógico añadiendo dos variables booleanas para indicar si la cargadora se esta moviendo en alguno de los dos ejes. Por último, tenemos el funcionamiento numérico, que es igual al digital salvo que la variable que indica la posición destino es un número entero.

Por facilidad de cara a la programación final, se escoge el funcionamiento numérico.

Variable	E/S del controlador	Tipo	Descripción
Posición destino	Entrada	Entero	Muévase a la celda deseada
Núm. De grúa (izquierda)	Entrada	Booleana	Mueva las uñas a la izquierda
Núm. De grúa (derecha)	Entrada	Booleana	Mueva las uñas a la derecha
Grúa # Elevación	Entrada	Booleana	Eleva cargadora
Grúa # Moving X	Salida	Booleana	Moviéndose en X
Grúa # Moving Z	Salida	Booleana	Moviéndose en Z
Grúa # Límite izquierdo	Salida	Booleana	Uñas en límite izquierdo
Grúa # Límite medio	Salida	Booleana	Uñas en el centro
Grúa # Límite derecho	Salida	Booleana	Uñas en límite derecho

Tabla 13: Entradas y salidas de la grúa cargadora de Factory IO

6. Emisor

El emisor nos permite generar los artículos que ofrece en su biblioteca Factory I/O a través de órdenes de control. En caso de seleccionarse más de un artículo, se generarán de forma aleatoria. Además, se puede elegir el tiempo que transcurre entre cada emisión estableciendo unos valores mínimos y máximos. En caso de que estos valores estén a cero, se emitirá siempre que no haya ninguna pieza dentro del volumen del emisor. Por último, también se puede introducir el número máximo de bases a emitir, siendo el máximo 500.

Respecto al funcionamiento, tenemos el modo definido por el usuario, en el que seleccionamos el artículo a emitir y a través del control indicamos cuando emitir, y el modo definido por el control, donde además de cuándo emitir se indica que emitir a través del número identificativo.

Se ha seleccionado el modo definido por el usuario, ya que solo necesitamos que genere un tipo de artículo por emisor y de esta forma el control se simplifica.

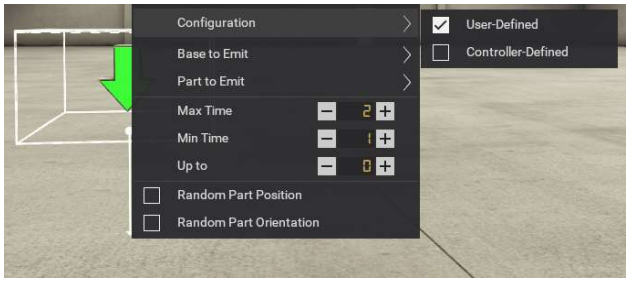


Figura 28: Opciones de configuración de la pieza emisora

Una vez se han dispuesto los elementos que componen las distintas estaciones y se ha seleccionado el modo de funcionamiento, para facilitar la comprensión y el control posterior vamos a cambiarle el nombre a los distintos sensores y actuadores.

En primer lugar, debemos mostrar en pantalla las etiquetas tanto de los sensores como de los actuadores. Para ello tenemos dos opciones:

- Desde la pestaña *View > Sensor tags y Actuator tags*
- Pinchando en los iconos que aparecen en la esquina superior derecha

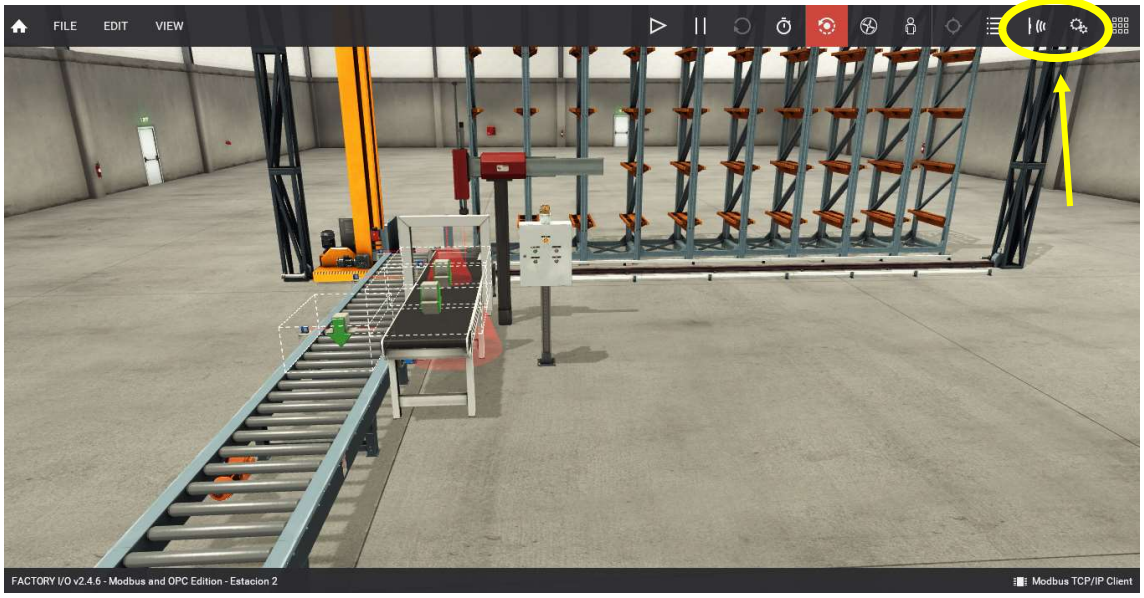


Figura 29: Modo de activación de la representación de sensores y actuadores en Factory IO

Nos aparecerá encima de cada sensor/actuador el nombre predeterminado que Factory I/O da a cada componente. Haciendo doble click sobre él se abrirán las opciones de cara al control que nos ofrece el sensor/actuador, las cuales se detallarán más adelante, y al lado el nombre, el cual podremos editar a nuestro gusto.

Figura 30: Menú de sensor en Factory IO

Tras la selección de los componentes que conforman la estación, la selección del modo de funcionamiento de cada uno y la identificación de todas las variables de entrada y salida por un nombre más intuitivo, estamos en disposición de comenzar con la simulación de la escena.

12.1.3. Simulación de una escena

La simulación de una escena se puede realizar a través de dos formas. La primera es la realización de una simulación forzado en el que es el usuario el que fuerza los valores de los distintos actuadores. La segunda simulación es a través de un código de control. Este código de control se realiza en un programa externo y se carga en el PLC de Factory I/O. Además dentro de las simulaciones, Factory I/O permite inyectar fallos en los componentes para así poder simular fallos que se pueden producir en una estación real.

En primer lugar, se va a explicar cómo realizar un control forzado de la estación. Para ello deberemos pinchar en *View > Dock all tags.*

Aparecerán a nuestra izquierda una lista con todos los sensores y actuadores presentes en la escena. Los sensores irán acompañados de una circunferencia con un círculo en su interior oscuro (indicando que el sensor está a cero) y los actuadores de un círculo verde con una I de color blanco (indicando que el actuador no está forzado por el usuario).

Para comenzar con la simulación forzada pincharemos en el botón *play.* Forzaremos el valor de los actuadores pinchando en el círculo verde. Veremos como este cambia a color azul y en función de si es claro u oscuro estará a 1 o a 0.

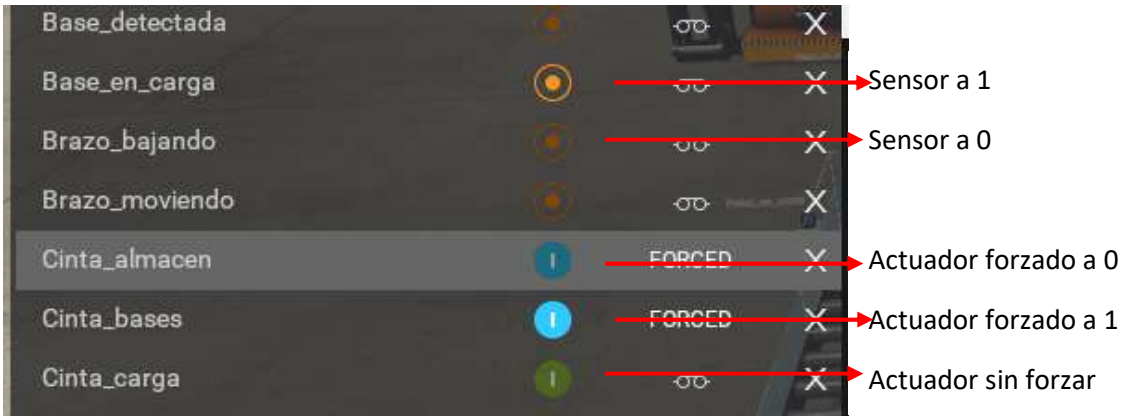


Figura 31: Posibles estados de sensores y actuadores en una simulación forzada de Factory IO

Este tipo de simulación nos permite ir modificando los valores de los actuadores para forzar el funcionamiento deseado y de esta manera poder comprobar que los sensores instalados toman los valores correctos para el funcionamiento óptimo de la estación.

En segundo lugar, se va a explicar como se realiza una simulación con un código de control. Este código se redacta en un programa aparte, en nuestro caso Unity Pro. Para poder establecer la conexión entre las variables del programa y las de Factory I/O debemos darle la misma dirección de memoria, por tanto, comenzaremos introduciendo como establecer la dirección de memoria de las variables en el software Factory I/O.

En la parte inferior derecha se encuentra el PLC de la estación, en el cual se introducirán las variables de entrada y salida. Si clicamos en él y entramos en configuración podemos establecer el número de entradas y salidas del sistema.

Figura 32: Icono del PLC de Factory IO

La configuración del PLC incluye los siguientes apartados:

Autoconectar	Intenta conectarse periódicamente al PLC
Anfitrión	Dirección IP del servidor
Puerto	Número de puerto TCP
ID esclavo	ID esclavo
Leer digital	Desde dónde leer las entradas digitales (Entradas o bobinas)
Leer registro	Desde donde leer los registros (de entrada o de retención)
Escala	Los valores del sensor flotante se multiplican por este valor y los del actuador se dividen para de esta forma poder trabajar con enteros
Entradas digitales	Desplazamiento de dirección y número de bobinas (máx. 256)
Salidas digitales	Desplazamiento de dirección y número de entradas digitales a usar (máx. 256)
Registrar entradas	Desplazamiento de dirección y número de registros de retención a usar (máx. 64)

Registrar salidas	Desplazamiento de dirección y número de registros de entrada que se utilizarán (máx. 64)

Tabla 14: Características del PLC Modbus TCP/IP Client

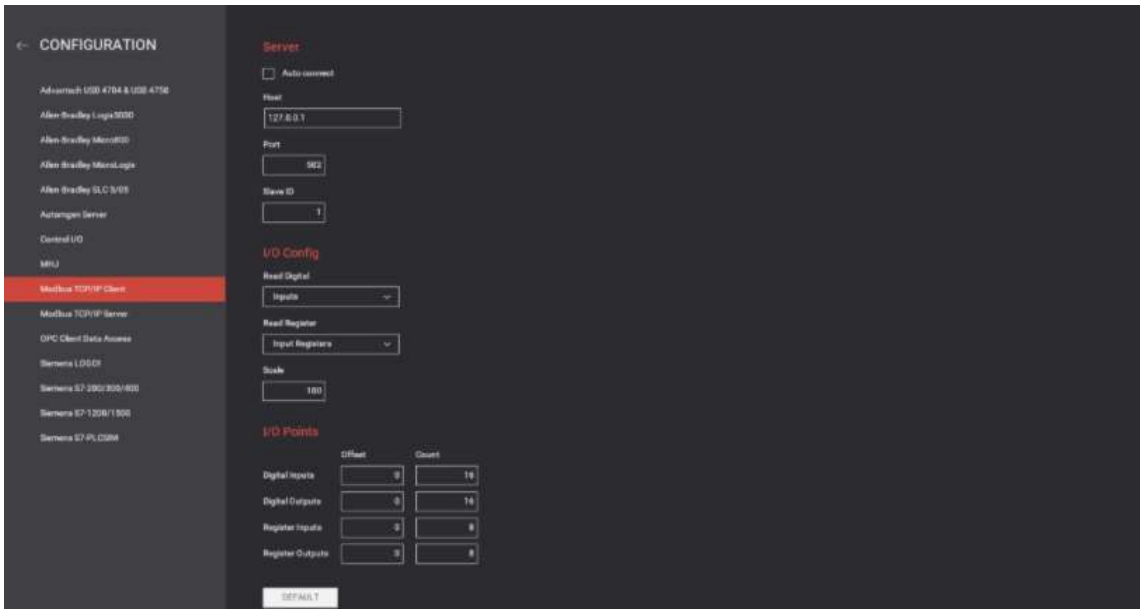


Figura 33: Menú configuración del PLC en Factory IO

Una vez se ha configurado el PLC, salimos del menú de configuración y vamos a conectar las entradas y salidas a los distintos pines habilitados, que se corresponderán con la dirección de memoria de cada una de las variables. Por último, estableceremos la conexión con el software de programación pinchando en *connect* y dándole al *play* iniciaremos la simulación.

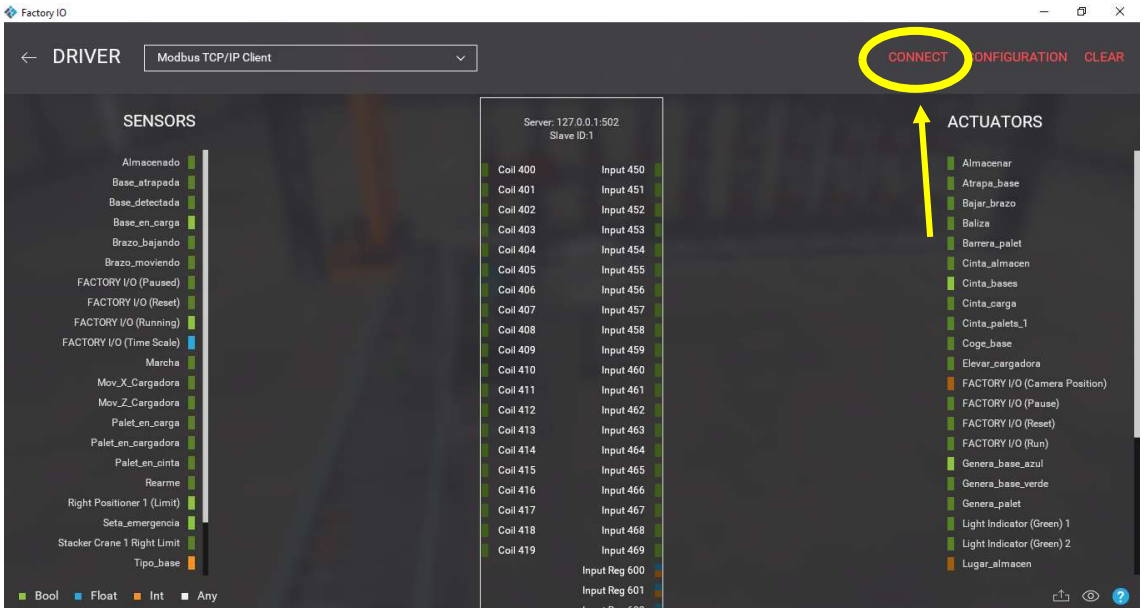


Figura 34: Conexión del PLC de Factory IO a la red Modbus

12.2. ANEXO II: Guía básica Unity Pro

Unity Pro es un software de programación, puesta a punto y explotación para los autómatas Modicon, M340, Premium, Quantum y coprocesadores Atrium desarrollado por la empresa Schneider Electric.

El objetivo de este anexo es familiarizarnos con el software, explicando el procedimiento para crear un proyecto como el que se ha realizado en este trabajo fin de grado.

Comenzaremos con la **creación de un nuevo proyecto**, para ello en Fichero > Nuevo, seleccionaremos el tipo de autómata a emplear. Debido a que el uso va a ser el control de una estación simulada y comunicación a través de una red Modbus, seleccionamos el BMX P34 2000 y pinchamos en aceptar.

Aparecerá a nuestra izquierda un menú con las distintas opciones del proyecto. Comenzaremos con la **introducción de las variables del sistema**. Para ello pinchamos en *Variables e Instancias FB > Variables elementales*. Se abrirá una nueva pestaña dónde podremos introducir el nombre de la variable, el tipo de variable y su dirección de memoria (esta dirección de memoria deberá coincidir con la otorgada a la variable en el PLC del software de simulación). Además se le podrá añadir un comentario a la variable a modo de explicación y también se podrá forzar el valor de la misma.

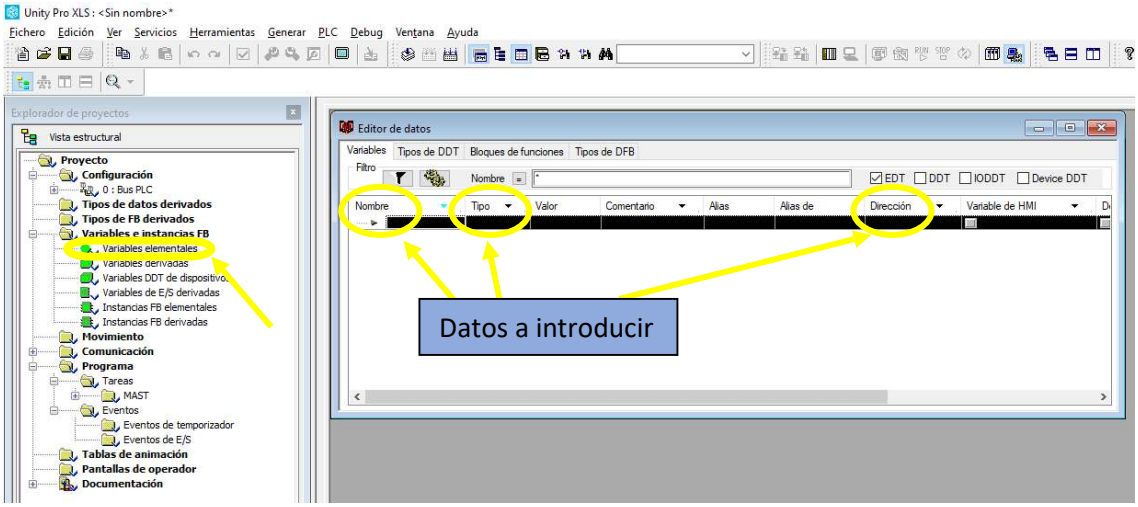


Figura 35: Establecer las variables elementales en Unity Pro

Una vez introducidas las variables necesarias para el correcto control de la estación, se procede a **redactar el código de control**. El software de programación ofrece 6 tipos diferentes de lenguaje de programación. En este caso se va a explicar el procedimiento para redactar los 2 tipos de lenguaje empleados.

Para poder escribir el código debemos crear una nueva sección. Para ello pincharemos en *Programa > MAST*, click derecho en la carpeta de secciones y seleccionamos *Nueva sección*. Se abrirá una nueva pestaña en la que introduciremos el nombre y el lenguaje de programación.

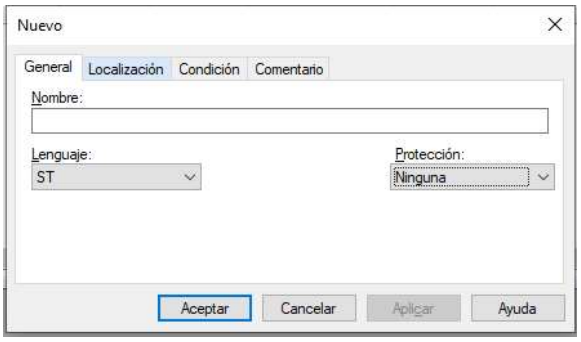


Figura 36: Nueva sección Unity Pro

En primer lugar, se va a explicar el **lenguaje ST**. Este tipo de lenguaje consiste en la sucesión de expresiones condicionales. En estas expresiones las condiciones hacen referencia a las entradas del sistema y las conclusiones a la salida. Un ejemplo de este tipo de estructura es el siguiente:

```
end_if;
if (brazo_adelante=1 and brazo_arriba=1 and estado[0]=4) or (brazo_adelante=1 and brazo_arriba=1 and est
        estado[0]:=5;
end_if;

(* Vamos a poner una condicion para que el programa solo genere 3 bases azules, para de esta forma
simular que nos hemos quedado sin placas a la derecha *)
if brazo_abajo=1 and estado[0]=5 and Placa_derecha=1 and bases_dcha>2 then
        estado[0]:=6;
end_if;
if brazo_abajo=1 and estado[0]=5 and Placa_derecha=1 and bases_dcha<=2 then
        estado[0]:=6;
        Bases_dcha:=bases_dcha+1;
end_if;
(*Vamos a hacer lo mismo con las placas de la izquierda, pero en este caso generara una mas para que hay
diferencia entre ambos lados y podamos programar que el robot deje todas las placas antes de avisar que
hace falta reponer*)
if brazo_abajo=1 and estado[0]=5 and Placa_izquierda=1 and bases_izda>4 then
        estado[0]:=6;
end_if;
if brazo_abajo=1 and estado[0]=5 and Placa_izquierda=1 and bases_izda<=4 then
        estado[0]:=6;
        Bases_izda:=bases_izda+1;
end_if;
if (vacio=0 and Placa_derecha=1 and Sin_bases_verdes=0 and estado[0]=6) or (contador2>50 and Placa_derec
        estado[0]:=7;
        expulsar_izdo:=1;
        Placa_derecha:=0;
        Placa_izquierda:=1;
end_if;
```

Figura 37: Ejemplo de código en lenguaje ST

En el caso del **lenguaje SFC**, se trata de un tipo de lenguaje que consiste en una secuencia de etapas y transiciones. Cada etapa implica una acción y el paso de una etapa a otra se rige por las transiciones.

La acción que lleva asociada una etapa se puede corresponder con una salida del sistema directamente o con una función de varias salidas del sistema. A su vez, una transición puede llevar asociado el valor de un sensor de la estación o una función más compleja de los valores de varios sensores.

Para la realización del código, empezaremos disponiendo los distintos **pasos, transiciones y la conexión entre ellos**. Otro elemento que se puede emplear es el salto, que nos permite pasar a un paso sin necesidad de conectarlos para así evitar el exceso de cruces entre las conexiones.

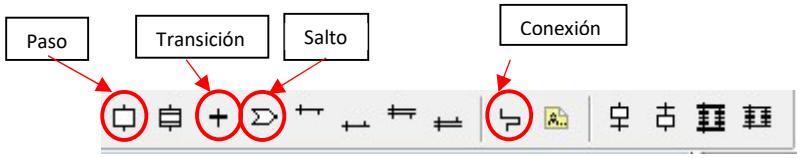


Figura 38: Componentes del lenguaje SFC

Una vez hemos colocado todos los pasos y transiciones sobre la sección vamos a asociarles las acciones y variables pertinentes. Comenzamos haciendo doble clic sobre los pasos donde se va a abrir la siguiente ventana:

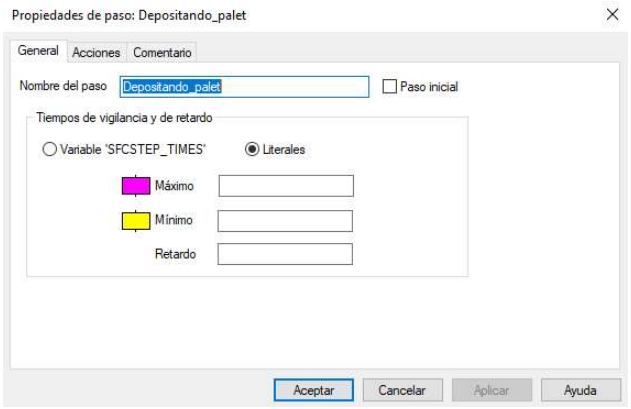


Figura 39: Menú de paso, lenguaje SFC

En el apartado general se deberá indicar el nombre del paso y se podrá establecer:

- Tiempo máximo: Tiempo máximo que puede estar activo un paso, en caso de excederse se emitirá un mensaje de error. Si no se indica ningún tiempo, este no se vigilará.
- Tiempo mínimo: Tiempo mínimo que debe estar activo un paso, en caso de no alcanzarse se emitirá un mensaje de error. Si no se indica ningún tiempo, no se vigilará.
- Retardo: Tiempo mínimo que el paso debe estar activo.

En la pestaña de **acciones** será donde seleccionemos las acciones asociadas al paso. En primer lugar, debemos seleccionar si la acción se trata directamente de una variable o si por el contrario se trata de una sección. En caso de ser una sección, se deberá editar y programar en otro de los lenguajes de programación que ofrece Factory I/O. Además, a cada acción se le deberá establecer un descriptor que defina su control. Los tipos de descriptores son:

N/Ninguno	Si se activa el paso la acción se pone a 1, si se desactiva la acción se pone a 0
S	La acción queda activa aún cuando el paso termina. Se pone a 0 cuando se usa el descriptor *R*.
R	La acción activada por el descriptor *S* se pone a 0.
L	La acción se mantendrá activa durante el tiempo establecido, aunque el paso no haya terminado. Si el paso se desactiva, la acción se pone a 0
D	Se establece un tiempo de retardo para la activación de la acción. Una vez transcurrido el paso, la acción se pone a 0.
P	Si el paso se activa, la acción se pone a 1 y permanece durante un ciclo del programa.
DS	La acción queda activa hasta que se reestablece con el descriptor *R*. La diferencia con *S* es que la activación de la acción lleva un tiempo de retardo
P1	Si el paso se activa (flanco 0->1) la acción se activa y permanece durante un ciclo del programa.
P0	Si el paso se desactiva (flanco 1->0) la acción se activa y permanece durante un ciclo del programa.

Tabla 15: Opciones de acción de paso, Unity Pro

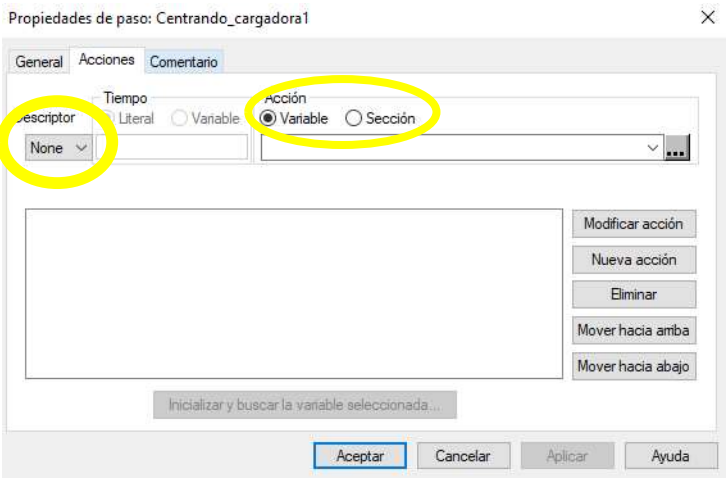


Figura 40: Menú de acción, lenguaje SFC

Una vez definidas las acciones asociadas a los pasos de nuestro programa, se van a definir las variables asociadas a las **transiciones**. Estas variables van a suponer la condición de cambio de un paso a otro. Al igual que con las acciones, pueden estar asociadas directamente a una variable o a una sección. Estos son varios ejemplos de secciones:

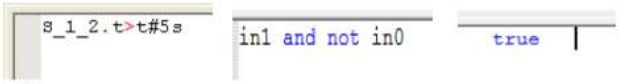


Figura 41: Ejemplo de secciones de transición, lenguaje SFC

La condición de cambio de paso se puede fijar a un valor de entrada de 0 pinchando en el recuadro de condición de transición invertida. De esta forma, cuando el valor de la variable asociada sea igual a 0 se disparará la transición.

Una vez hemos terminado con la redacción de los dos programas que van a controlar las estaciones, y de las secciones de apoyo realizadas para lograr el funcionamiento deseado, se detalla como **compilar el programa y la conexión con el PLC**.

En la pestaña *Generar*, clicamos en *Analizar Proyecto* para comprobar que no hay fallos en la redacción.

Si el resultado del análisis es de 0 errores, estamos en disposición de generar el proyecto. Vamos a *Generar Proyecto* en la pestaña de *Generar.*

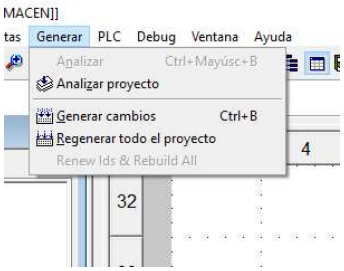


Figura 42: Generar proyecto en Unity Pro

Con el proyecto generado, lo conectamos al PLC. En la pestaña *PLC > Establecer dirección*, se va a indicar la dirección del software de simulación, en nuestro caso 127.0.0.1, y en medios TCPIP.

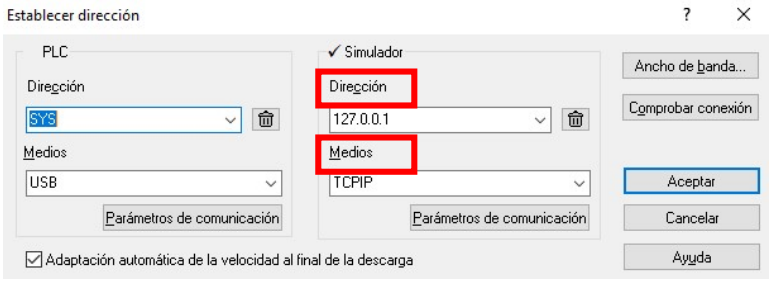


Figura 43: Establecer dirección en Unity Pro

En la pestaña *PLC > Conectar*, establecemos la conexión con Factory I/O. Una vez conectados en *PLC > Transferir proyecto a PLC*, cargamos las secciones de programa en el PLC de la simulación.

Por último en *PLC > Ejecutar* iniciamos la ejecución del programa en la simulación.

12.3. ANEXO III: Guion Práctica 4

Práctica 4: Control de una célula de fabricación flexible.

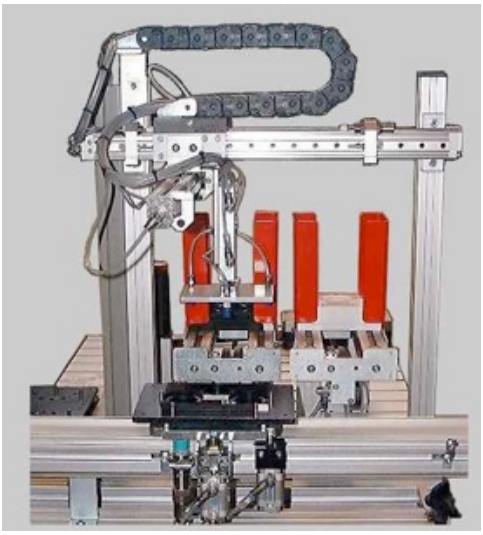
IMPORTANTE: Antes de la sesión en el laboratorio, ***es imprescindible haber realizado el estudio previo.***

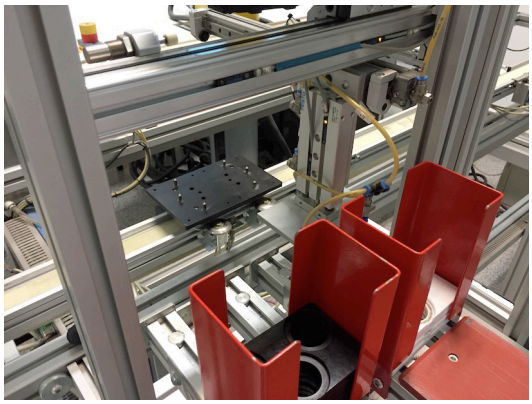
Objetivo: realizar la automatización completa de la estación 6 de la célula de fabricación, utilizando el lenguaje estructurado (ST) y el lenguaje *sequential function chart* (SFC, comúnmente denominado Grafcet).

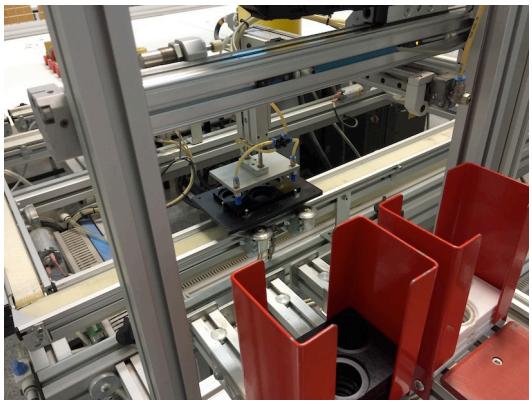
Funcionamiento de la estación 6

La estación 6 suministra las placas base sobre las que se montan los pedidos que luego se almacenarían en la estación 7. Existen dos tipos diferentes de placas, cuya única diferencia es el color: negro y blanco. Por medio de una serie de cilindros neumáticos y unas ventosas de vacío tendremos la posibilidad de trasladar las piezas de un punto a otro hasta colocarlas sobre el palet.

La estación posee dos alimentadores por gravedad sobre los cuales introduciremos cada uno de los dos tipos de base posibles, uno en cada uno de ellos. La situación de cada uno de los tipos de base debe ser la mostrada en la figura adjunta. También hay que colocar las placas correctamente orientadas: si se meten de forma errónea las ventosas del brazo que debe manipularlas coincidirán con los huecos de la placa y no se podrán levantar (al perderse el efecto de succión sobre la placa). Así que es necesario mirar la disposición de las ventosas en el brazo manipulador antes de introducir placas en los almacenes.







Funcionamiento básico

El inicio del proceso ha de lanzarse cuando el operario pulse el botón Marcha de la botonera de la estación. Eso permite sacar las piezas de una en una, teniendo el tiempo suficiente para retirar manualmente la que se acaba de dejar, y –lo que es más importante para la seguridad del operario– hacerlo con la máquina parada. El funcionamiento automático básico es el siguiente:

1) sacar una placa del alimentador por gravedad, para lo cual debe extenderse un cilindro de simple efecto existente en su base. El color de las placas negras/blancas que se sacan depende del conmutador `IND-INT` de la botonera (salidas `Expulsar_izdo`/ `Expulsar_drcho`).
2) con el cilindro completamente extendido (no existe sensor para detectar dicha extensión): si existe placa será detectada por un sensor óptico (y ya puede recogerse el cilindro que lo sacó); si no existe placa habrá que avisar al operador de que debe recargar el alimentador que se ha quedado vacío (lo que puede hacerse mediante una baliza existente en la propia estación)
3) posicionar el brazo sobre la placa recién extraída, bajarlo, hacer vacío para sujetar la placa: cuando el sensor detecte que se ha producido el vacío, significará que la placa esta sujeta, y podremos subir el brazo
4) llevar la placa hacia el carro, bajarla, y desactivar la succión; cuando el sensor de vacío nos indique que se ha dejado de succionar, subir el brazo y llevarlo donde corresponda
5) si un alimentador se queda sin piezas, hay que avisar al operador utilizando la baliza existente en la estación.

Algunas consideraciones:

1) Antes de cualquier ciclo de funcionamiento, la estación se tiene que posicionar de manera automática en el estado de reposo que corresponde a:
    a. Alimentadores atrás
    b. Brazo encima del alimentador de la derecha (placas blancas): arriba, atrás y a la derecha (visto desde la botonera)
2) Algunos movimientos se pueden ejecutar en paralelo (por ejemplo mover el cabezal a la derecha y después adelante) optimizando los tiempos de producción. En la medida de lo posible, paraleliza los movimientos para optimizar los tiempos.
3) Durante el funcionamiento normal, hay que considerar las siguientes situaciones imprevistas:
    a. Si existe placa al fondo (ya sacada por algún motivo, pieza detectada por el óptico), hay que desplazar el cabezal y coger la pieza sacada y continuar con el proceso normal.
    b. Si no quedan placas en el depósito (no se activa el óptico en un tiempo razonable) hay que encender la baliza (de manera intermitente, usando una temporización diferente al estado de emergencia), y tras recargar y pulsar `RESET` se continua con el proceso normal.

Para activar la luz de forma intermitente se puede utilizar el bit de sistema %S6 (un temporizador interno regula el cambio en el estado de este bit; buscad en la ayuda de Unity el funcionamiento de dicho bit).

Descripción de las entradas y salidas

A continuación se tabulan las entradas y salidas de este automatismo. Puesto que se utiliza una periferia descentralizada (basada en bus CAN), las entradas y las salidas están *mapeadas* sobre bits de determinadas palabras de memoria (por lo que tendrás que tener especial cuidado cuando accedas a ellas en modo escritura). Los símbolos dados son a modo de ejemplo (puedes usar otros con los que te sientas más cómodo, si quieres). Nótese que el símbolo Reset no puede usarse, al coincidir con una palabra reservada del lenguaje de programación, y en su lugar se ha usado Rearme. Tampoco existen acentos en los símbolos, evidentemente.

ENTRADAS		
Objeto	**Símbolo**	**Condición de puesta a 1**
%MW35.0	Izdo_atras	El expulsor izquierdo está atrás.
%MW35.1	Optico_izdo	Hay placa sobre el sensor óptico izquierdo.
%MW35.2	Dcho_atrás	El expulsor derecho está atrás.
%MW35.3	Optico_dcho	Hay placa sobre el sensor óptico derecho.
%MW35.4	Brazo_arriba	El brazo que manipula la placa está arriba.
%MW35.5	Brazo_abajo	El brazo que manipula la placa está abajo.
%MW37.0	Brazo_dcha	El brazo que manipula la placa está a la derecha.
%MW37.1	Brazo_izda	El brazo que manipula la placa está a la izquierda.
%MW37.2	Seta_emergencia	Seta de emergencia enclavada.
%MW37.3	Marcha	Pulsador MARCHA apretado.
%MW37.4	Man_aut	En posición AUT.
%MW37.5	Rearme	Pulsador RESET apretado.
%MW39.0	Ind_int	En posición INT.
%MW39.1	Brazo_atras	El brazo que manipula la placa está a la atrás.
%MW39.2	Brazo_adelante	El brazo que manipula la placa está a la adelante.
%MW39.3	Vacio	El vacuostato detecta vacío.

Respecto a las salidas, las %MW134.0, .1 y .3 corresponden a cilindros neumáticos de **simple efecto**. Cuando la salida %MW134.2 vale 0, se deja de generar vacío. Las salidas %MW134.4 a %MW135.1 pueden (deben) manejarse de forma impulsional, ya que corresponden a **cilindros de doble efecto**.

SALIDAS		
Objeto	**Símbolo**	**Acción ejecutada si su valor es 1**
%MW134.0	Expulsar_izdo	Extender expulsor izquierdo.
%MW134.1	Expulsar_dcho	Extender expulsor derecho.
%MW134.2	Coger_placa	Generar vacío en las ventosas.
%MW134.3	Bajar_brazo	Bajar el brazo.
%MW134.4	Mover_izda	Mover el brazo hacia la izquierda.
%MW134.5	Mover_dcha	Mover el brazo hacia la derecha.
%MW135.0	Mover_adelante	Mover el brazo hacia la adelante.
%MW135.1	Mover_atras	Mover el brazo hacia la atrás.
%MW135.2	Baliza	Encender la luz de la baliza luminosa.

Estudio previo

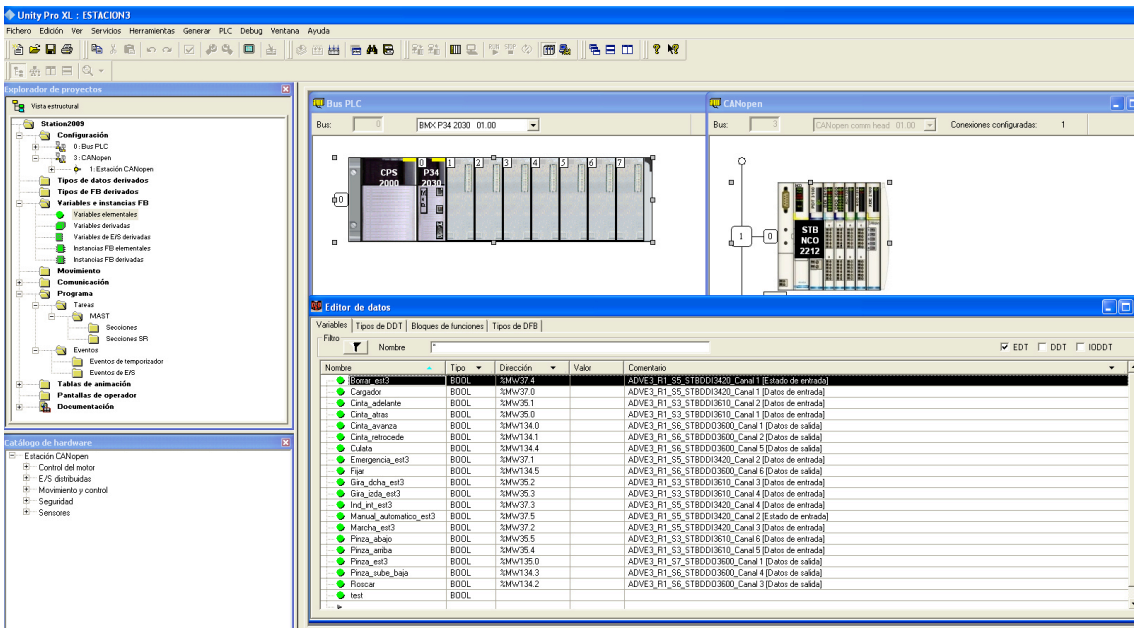
Es necesario que el grupo de prácticas acuda a la sesión con el problema estudiado y una solución consistente en el modelo del funcionamiento automático de la estación 6 y su control usando una red de Petri interpretada.

Sesión de laboratorio

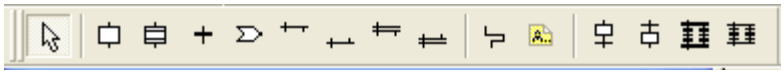
Implementa el funcionamiento del automatismo descrito antes.

Se realizará la implementación de la red de Petri elaborada en el estudio previo y se verificará el correcto funcionamiento del programa. Para realizar la implementación se utilizará el fichero correspondiente disponible en el repositorio en moodle de la asignatura: estacion6.stu.

Los ficheros con extensión stu son ficheros de programación en Unity. Una vez abierto el fichero aparecerá una ventana donde se visualiza el explorador de proyectos a la izquierda. El explorador de proyectos está también disponible en el menú Herramientas. Los ficheros proporcionados para realizar la sesión vienen ya con la configuración hardware correcta, así como con las variables de memoria, ya introducidas, asociadas a la comunicación con la máquina. Estas variables están declaradas en Variables e instancias FB/Variables elementales del Explorador de proyectos.



Para realizar la programación se abrirán nuevas secciones con el botón derecho del ratón en Programa/Tareas/MAST/Secciones del Explorador de proyectos dándoles el nombre que se desee. Para programar la sección principal en SFC se utilizará la siguiente paleta:



Mediante ella se podrá seleccionar, insertar etapas, macro-etapas, transiciones, saltos, ramificaciones y conjunciones alternativas, ramificaciones y conjunciones paralelas, conexiones, comentarios, secuencias etapa-transición o transición-etapa, secuencias paralelas o secuencias alternativas.

La sección deberá tener una etapa inicial que se puede definir marcando la casilla *Paso inicial* de la pestaña General del paso. Para introducir un tiempo de retraso de una etapa, se puede utilizar la variable Nombre_del_paso.t en las transiciones. La variable Nombre_del_paso.x se puede utilizar para comprobar si la etapa está activa o no.

Para insertar una variable predefinida pulsar el botón derecho del ratón y después Selección de datos seguido del símbolo  para que se despliegue el listado donde elegir. Se puede crear una variable no existente pulsando con el botón derecho del ratón sobre el nombre escrito de la variable en el código, seguido de *Crear variable*.

Existen dos tipos de variables lógicas:

BOOL: Podrá valer False (=0), o True (=1).

EBOOL: Podrá valer False (=0), o True (=1) pero también incluye información relativa a la gestión de los flancos ascendentes o descendentes y el forzado.

Las transiciones pueden definirse como variable o como sección. Para crear una sección de transición se le dará un nombre y se elegirá a continuación el lenguaje en que desarrollarla. Las secciones de transición son fácilmente localizables en la carpeta Transiciones. Para utilizar una variable negada como transición se marcará la casilla “Condición de transición invertida”.

Las acciones en los pasos pueden introducirse como variables o como sección. Una vez elegida la variable, o escrito el nombre de la sección se deberá pulsar “Nueva acción” para guardarla. En el caso de las secciones se hará doble clic sobre el nombre para elegir el lenguaje en que desarrollarla. Las secciones de acción son fácilmente localizables en la carpeta Acciones. Cada paso puede contener varias variables y secciones. El descriptor a utilizar para la acción es:

- N (None): Si el paso está activo la acción se establece en 1. Si el paso no está activo la acción se establece en 0.

Mediante el menú Ver/Visualización expandida se pueden visualizar tanto los pasos como las acciones asociadas a éstos.

Análisis, transferencia y ejecución del código

Si se desea comprobar el correcto comportamiento del código mediante simulación se elegirá el menú PLC/Modalidad de simulación. Si por el contrario se desea ejecutar el código en un autómata real se elegirá PLC/Modalidad estándar. En este último caso se deberá establecer la dirección física del autómata mediante PLC/Establecer dirección. Se introducirá la dirección del autómata de la célula y se seleccionará como medio TCPIP (estos parámetros están ya configurados en los ficheros suministrados para la realización de la práctica).

Una vez elegida la modalidad se analizará el proyecto mediante el menú Generar/Analizar proyecto y se depurarán los posibles errores. Una vez depurados los errores aparecerá el mensaje "Analizado" en color amarillo en la parte inferior de la ventana. A continuación se ejecutará el menú Generar/Regenerar todo el proyecto con objeto de tener el proyecto generado. Si el proyecto se ha generado correctamente ha de aparecer el mensaje "Generado" en color verde en la parte inferior de la ventana. Si el proyecto ya se había generado con anterioridad pero se han realizado cambios no es necesario regenerar todo el proyecto. Es suficiente con ejecutar Generar/Generar cambios.

El proyecto generado se ha de transferir al PLC (real o simulado). Para ello primero habrá que conectarse con el autómata mediante PLC/Conectar. Si el proyecto no coincide con el que hay en el autómata aparecerá indicado mediante el mensaje "Diferente" en color rojo en la parte inferior de la ventana. Para que coincidan se seleccionará PLC/Transferir proyecto a PLC, se elegirá si se desea ejecutar el PLC después de la transferencia, y se pulsará transferir en el cuadro de diálogo. Una vez transferido el proyecto el mensaje "Diferente" habrá sido reemplazado por el mensaje "Igual" en color verde. Si el programa no está en ejecución aparecerá el mensaje "Stop" en amarillo en la parte inferior. Para ejecutarlo se seleccionará PLC/Ejecutar y el mensaje de "Stop" dará paso al de "Run" en color verde. Mediante PLC/Ejecutar el programa se ejecutará desde el comienzo si se acaba de transferir o, si ya se había ejecutado previamente, desde donde se había pasado a "Stop" mediante PLC/Detener. Si se desea ejecutar el programa desde el inicio se deberá llevar el programa a "Stop" mediante PLC/Detener, luego pulsar PLC/Inicializar, y finalmente PLC/Ejecutar.

12.4. ANEXO IV: Secciones de control

MAST

Propiedades específicas

Configuración	Cíclica
Configuración del periodo de tareas	0
Configuración del tiempo de watchdog	250

Mapeo_entradas : [MAST]

```
   1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|
```

```
 1  if simulando=0 then
 2          izdo_atras:=izdo_atras_real;
 3          optico_izdo:=optico_izdo_real;
 4          dcho_atras:=dcho_atras_real;
 5          optico_dcho:=optico_dcho_real;
 6          brazo_abajo:=brazo_abajo_real;
 7          brazo_adelante:=brazo_adelante_real;
 8          brazo_arriba:=brazo_arriba_real;
 9          brazo_atras:=brazo_atras_real;
10          brazo_dcha:=brazo_dcha_real;
11          brazo_izda:=brazo_izda_real;
12          ind_int:=ind_int_real;
13          man_aut:=man_aut_real;
14          marcha:=marcha_real;
15          rearme:=rearme_real;
16          seta_emergencia:=seta_emergencia_real;
17          vacio:=vacio_real;
18  else
19          izdo_atras:=izdo_atras_fact;
20          optico_izdo:=optico_izdo_fact;
21          dcho_atras:=dcho_atras_fact;
22          optico_dcho:=optico_dcho_fact;
23          brazo_adelante:=brazo_adelante_fact;
24          brazo_atras:=brazo_atras_fact;
25          brazo_dcha:=brazo_dcha_fact;
26          brazo_izda:=brazo_izda_fact;
27          ind_int:=ind_int_fact;
28          man_aut:=man_aut_fact;
29          marcha:=marcha_fact;
30          rearme:=rearme_fact;
31          seta_emergencia:=seta_emergencia_fact;
32  end_if;
33
34  (*Vamos a diseñar los sensores de brazo arriba y brazo abajo, que no existen en Factory IO*)
35
36  if Simulando=1 and Situacion_brazo>870 then
37          brazo_abajo:=1;
38  else
39          brazo_abajo:=0;
40  end_if;
41
42  if Simulando=1 and Situacion_brazo=0 then
43          brazo_arriba:=1;
44  else
45          brazo_arriba:=0;
46  end_if;
47
48  (*Vamos a programar el sensor de vacio para Factory IO*)
49  if Simulando=1 and brazo_atras=1 and RE(brazo_abajo)=1 and estado[1]=0 then
50          estado[1]:=1;
51  end_if;
52  if Simulando=1 and estado[1]=1 and cont1>50 then
53          estado[1]:=0;
54          vacio:=1;
55  end_if;
56
57  if Simulando=1 and estado[1]=0 and RE(brazo_abajo)=1 and brazo_adelante=1 then
58          estado[1]:=2;
59  end_if;
60  if Simulando=1 and estado[1]=2 and contador2>50 then
61          estado[1]:=0;
62          vacio:=0;
63  end_if;
64
65  (*A continuación se detalla el valor de los contadores para cada estado*)
66  if estado[1]=0 then
67          cont1:=0;
68          contador2:=0;
69  end_if;
70  if estado[1]=1 then
71          cont1:=cont1+1;
72  end_if;
73  if estado[1]=2 then
74          contador2:=contador2+1;
75  end_if;
```

Control_Estacion6 : [MAST]

```
   1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|
  1  if %s13 then
  2          expulsar_dcho:=0;
  3          expulsar_izdo:=0;
  4          estado[0]:=0;
  5          Placa_izquierda:=0;
  6          Placa_derecha:=0;
  7          Sin_placas:=0;
  8  end_if;
  9  if estado[0]=0 and brazo_dcha=1 and Sin_Placas=1 then
 10          estado[0]:=10;
 11  end_if;
 12  if estado[0]=10 and Sin_placas=1 and Rearme=1 then
 13          Sin_placas:=0;
 14          primer_ciclo:=1;
 15          Sin_bases_verdes:=0;
 16          Sin_bases_azules:=0;
 17          Placa_izquierda:=0;
 18          Placa_derecha:=0;
 19          Baliza:=0;
 20          estado[0]:=1;
 21          bases_dcha:=1;
 22          bases_izda:=1;
 23  end_if;
 24  if brazo_dcha=1 and estado[0]=0 and Sin_placas=0 then
 25          estado[0]:=1;
 26          bases_dcha:=1;
 27          bases_izda:=1;
 28  end_if;
 29  if brazo_atras=1 and brazo_dcha=1 and brazo_arriba=1 and marcha=1 and estado[0]=1 and ind_int=1 then
 30          Expulsar_dcho:=1;
 31          Placa_derecha:=1;
 32  end_if;
 33  if brazo_atras=1 and brazo_dcha=1 and brazo_arriba=1 and marcha=1 and estado[0]=1 and ind_int=0 then
 34          Expulsar_izdo:=1;
 35          Placa_izquierda:=1;
 36  end_if;
 37  if optico_dcho=1 and estado[0]=1 then
 38          estado[0]:=2;
 39          Expulsar_dcho:=0;
 40  end_if;
 41  if optico_izdo=1 and estado[0]=1 then
 42          estado[0]:=8;
 43          Expulsar_izdo:=0;
 44  end_if;
 45  if brazo_izda=1 and estado[0]=8 then
 46          estado[0]:=2;
 47  end_if;
 48  if brazo_abajo=1 and estado[0]=2 then
 49          estado[0]:=3;
 50  end_if;
 51  if vacio=1 and estado[0]=3 and Placa_izquierda=1 then
 52          estado[0]:=9;
 53  end_if;
 54  if vacio=1 and estado[0]=3 and Placa_derecha=1 then
 55          estado[0]:=4;
 56  end_if;
 57  if (brazo_adelante=1 and brazo_arriba=1 and estado[0]=4 and estacion2=0) or (brazo_adelante=1 and brazo_arriba
57>>=1 and estado[0]=9 and estacion2=0) then
 58          estado[0]:=5;
 59          no_hay:=1;
 60  end_if;
 61
 62  if brazo_abajo=1 and estado[0]=5 and Placa_derecha=1 then
 63          estado[0]:=6;
 64  end_if;
 65
 66  if brazo_abajo=1 and estado[0]=5 and Placa_izquierda=1 then
 67          estado[0]:=6;
 68  end_if;
 69
 70  (*En este caso, al no tener en Factory IO un sensor que inique si se han acitvado las ventosas
 71  creamos una condición doble, una para la estación real por medio de la señal de las ventosas
 72  y otra para la simulada por medio de un contador*)
 73
 74  if (vacio=0 and Placa_derecha=1 and Sin_bases_verdes=0 and estado[0]=6) then
 75          estado[0]:=7;
 76          expulsar_izdo:=1;
 77          Placa_derecha:=0;
 78          Placa_izquierda:=1;
 79  end_if;
 80  if (vacio=0 and Placa_derecha=1 and Sin_bases_verdes=1 and estado[0]=6) then
 81          estado[0]:=7;
 82          expulsar_dcho:=1;
 83  end_if;
 84  if (vacio=0 and Placa_izquierda=1 and Sin_bases_azules=0 and estado[0]=6) then
 85          estado[0]:=7;
 86          expulsar_dcho:=1;
```

Control_Estacion6

```
   1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|

 87          Placa_izquierda:=0;
 88          Placa_derecha:=1;
 89  end_if;
 90  if (vacio=0 and Placa_izquierda=1 and Sin_bases_azules=1 and estado[0]=6) then
 91          estado[0]:=7;
 92          expulsar_izdo:=1;
 93  end_if;
 94  if estado[0]=7 and brazo_atras=1 and optico_dcho=1 then
 95          estado[0]:=2;
 96          expulsar_dcho:=0;
 97  end_if;
 98  if estado[0]=7 and brazo_atras=1 and optico_izdo=1 then
 99          estado[0]:=8;
100          expulsar_izdo:=0;
101  end_if;
102  (*Programamos que la estación siga funcionando hasta que ambos lados se queden sin placas,
103  en caso de fallar en un lado, se sacaran todas las placas del otro.*)
104
105  if contador1>100 and Placa_derecha=1 and Sin_bases_verdes=0 and estado[0]=7 then
106          Sin_bases_azules:=1;
107          expulsar_dcho:=0;
108          expulsar_izdo:=1;
109          Placa_derecha:=0;
110          Placa_izquierda:=1;
111          contador1:=0;
112  end_if;
113  if contador1>100 and Placa_derecha=1 and Sin_bases_verdes=1 and estado[0]=7 then
114          Sin_bases_azules:=1;
115          Sin_placas:=1;
116          estado[0]:=0;
117          baliza:=1;
118          expulsar_izdo:=0;
119          expulsar_dcho:=0;
120  end_if;
121  if contador1>100 and Placa_izquierda=1 and Sin_bases_azules=0 and estado[0]=7 then
122          Sin_bases_verdes:=1;
123          expulsar_dcho:=1;
124          Placa_derecha:=1;
125          Placa_izquierda:=0;
126          contador1:=0;
127  end_if;
128  if contador1>100 and Placa_izquierda=1 and Sin_bases_azules=1 and estado[0]=7 then
129          Sin_bases_verdes:=1;
130          Sin_placas:=1;
131          estado[0]:=0;
132          baliza:=1;
133          expulsar_izdo:=0;
134          expulsar_dcho:=0;
135  end_if;
136
137
138  (*Se va a programar el funcionamiento de la seta de emergencia, se debe tener en cuenta que por defecto
139  el valor de la seta en Factory IO es igual a 1*)
140
141  if Seta_emergencia=0 then
142          estado[0]:=10;
143          expulsar_dcho:=0;
144          expulsar_izdo:=0;
145          emergencia:=1;
146          baliza:=1;
147  end_if;
148  if Seta_emergencia=1 and emergencia=1 then
149          estado[0]:=0;
150          emergencia:=0;
151          baliza:=0;
152          Sin_bases_verdes:=0;
153          Sin_bases_azules:=0;
154          Placa_derecha:=0;
155          Placa_izquierda:=0;
156  end_if;
```

Estados_del_brazo_robot : [MAST]

```
   1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|
```

```
 1  if estado[0]=0 then
 2          Bajar_brazo:=0;
 3          Mover_derecha:=1;
 4          Mover_izda:=0;
 5          Mover_atras:=1;
 6          Mover_adelante:=0;
 7          Coger_placa:=0;
 8  end_if;
 9  if estado[0]=1 then
10          Bajar_brazo:=0;
11          Mover_derecha:=0;
12          Mover_izda:=0;
13          Mover_atras:=0;
14          Mover_adelante:=0;
15          Coger_placa:=0;
16  end_if;
17  if estado[0]=2 then
18          Bajar_brazo:=1;
19          Mover_derecha:=0;
20          Mover_izda:=0;
21          Mover_atras:=0;
22          Mover_adelante:=0;
23          Coger_placa:=0;
24          contador1:=0;
25  end_if;
26  if estado[0]=3 then
27          Bajar_brazo:=1;
28          Mover_derecha:=0;
29          Mover_izda:=0;
30          Mover_atras:=0;
31          Mover_adelante:=0;
32          Coger_placa:=1;
33  end_if;
34  if estado[0]=4 then
35          Bajar_brazo:=0;
36          Mover_derecha:=0;
37          Mover_izda:=0;
38          Mover_atras:=0;
39          Mover_adelante:=1;
40          Coger_placa:=1;
41  end_if;
42  if estado[0]=5 then
43          Bajar_brazo:=1;
44          Mover_derecha:=0;
45          Mover_izda:=0;
46          Mover_atras:=0;
47          Mover_adelante:=0;
48          Coger_placa:=1;
49  end_if;
50  if estado[0]=6 then
51          Bajar_brazo:=1;
52          Mover_derecha:=0;
53          Mover_izda:=0;
54          Mover_atras:=0;
55          Mover_adelante:=0;
56          Coger_placa:=0;
57  end_if;
58  if estado[0]=7 then
59          Bajar_brazo:=0;
60          Mover_derecha:=0;
61          Mover_izda:=0;
62          Mover_atras:=1;
63          Mover_adelante:=0;
64          Coger_placa:=0;
65          contador1:=contador1+1;
66  end_if;
67  if estado[0]=8 then
68          Bajar_brazo:=0;
69          Mover_derecha:=0;
70          Mover_izda:=1;
71          Mover_atras:=0;
72          Mover_adelante:=0;
73          Coger_placa:=0;
74          contador1:=0;
75  end_if;
76  if estado[0]=9 then
77          Bajar_brazo:=0;
78          Mover_derecha:=1;
79          Mover_izda:=0;
80          Mover_atras:=0;
81          Mover_adelante:=1;
82          Coger_placa:=1;
83  end_if;
84  if estado[0]=10 then
85          Bajar_brazo:=0;
86          Mover_derecha:=0;
87          Mover_izda:=0;
```

Estados_del_brazo_robot

```
   1|       10|       20|       30|       40|       50|       60|       70|       80|       90|       100|       110|
```

```
88          Mover_atras:=0;
89          Mover_adelante:=0;
90          Coger_placa:=0;
91   end_if;
```

Mapeo_salidas : [MAST]

```
   1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|
```

```
 1  (*Vamos a darle un valor a las salidas del programa en función de si estamos trabajando
 2  con la maqueta real o con la estación de simulación*)
 3
 4  if Simulando=0 then
 5          Expulsar_izdo:=Expulsar_izdo_real;
 6          Expulsar_dcho:=Expulsar_dcho_real;
 7          Coger_placa:=Coger_placa_real;
 8          Bajar_brazo:=bajar_brazo_real;
 9          Mover_izda:=Mover_izda_real;
10          Mover_derecha:=Mover_derecha_real;
11          Mover_adelante:=Mover_adelante_real;
12          Mover_atras:=Mover_atras_real;
13          Baliza:=Baliza_real;
14  else
15          Expulsar_izdo_fact:=Expulsar_izdo;
16          Expulsar_dcho_fact:=Expulsar_dcho;
17          Coger_placa_fact:=Coger_placa;
18          Baliza_fact:=Baliza;
19  end_if;
20
21  (*Vamos a convertir las salidas digitales del programa en salidas analógicas necesarias para el robot*)
22
23  if mover_derecha=1 and simulando=1 then
24          Mov_X:=730;
25  end_if;
26  if mover_izda=1 and simulando=1 then
27          Mov_X:=310;
28  end_if;
29  if mover_derecha=0 and mover_izda=0 and simulando=1 then
30          Mov_X:=Pos_X;
31  end_if;
32  if mover_atras=0 and mover_adelante=0 and simulando=1 then
33          Mov_Y:=Pos_Y;
34  end_if;
35  if mover_atras=1 and simulando=1 then
36          Mov_Y:=90;
37  end_if;
38  if mover_adelante=1 and simulando=1 then
39          Mov_Y:=1000;
40  end_if;
41  if bajar_brazo=0 and simulando=1 then
42          Mov_Z:=0;
43  end_if;
44  if bajar_brazo=1 and simulando=1 then
45          Mov_Z:=900;
46  end_if;
```

Simulacion_planta : [MAST]

```
   1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|
 1  if Simulando=1 and estado[0]=1 and Sin_placas=1 and Rearme=1 then
 2          Genera_base_verde:=1;
 3          Genera_base_azul:=1;
 4  end_if;
 5  if brazo_dcha=1 and estado[0]=1 and Sin_placas=0 and Simulando=1 then
 6          Genera_base_verde:=1;
 7          Genera_base_azul:=1;
 8  end_if;
 9  if brazo_abajo=1 and estado[0]=3 and Simulando=1 then
10          Genera_base_verde:=0;
11          Genera_base_azul:=0;
12  end_if;
13  if estado[0]=5 then
14          suma_base:=1;
15  end_if;
16  if brazo_abajo=1 and suma_base=1 and estado[0]=6 and Placa_derecha=1 and bases_dcha<=2 and Simulando=1 then
17          Genera_base_azul:=1;
18          bases_dcha:=bases_dcha+1;
19          suma_base:=0;
20  end_if;
21  if brazo_abajo=1 and suma_base=1 and estado[0]=6 and Placa_izquierda=1 and bases_izda<=4 and Simulando=1 then
22          Genera_base_verde:=1;
23          bases_izda:=bases_izda+1;
24          suma_base:=0;
25  end_if;
26
27  (*A continuación vamos a controlar la generación de bases en la segunda estacion*)
28
29  if RE(llega_pieza)=1 and Placa_derecha=1 and no_hay=1 then
30          Genera_base_azul_al:=1;
31          Genera_palet:=1;
32          no_hay:=0;
33  end_if;
34  if Pieza_a=1 then
35          genera_base_azul_al:=0;
36  end_if;
37  if RE(llega_pieza)=1 and Placa_izquierda=1 and no_hay=1 then
38          Genera_base_verde_al:=1;
39          Genera_palet:=1;
40          no_hay:=0;
41  end_if;
42  if pieza_v=1 then
43          Genera_base_verde_al:=0;
44  end_if;
45  if palet_en_cinta=1 then
46          genera_palet:=0;
47  end_if;
48
49  (*Se programa la variable que nos indica si se está ejecutando una única estación o las dos*)
50
51  if DosEstaciones=0 then
52          estacion2:=0;
53  end_if;
54
```

ALMACEN : [MAST]

Comentario

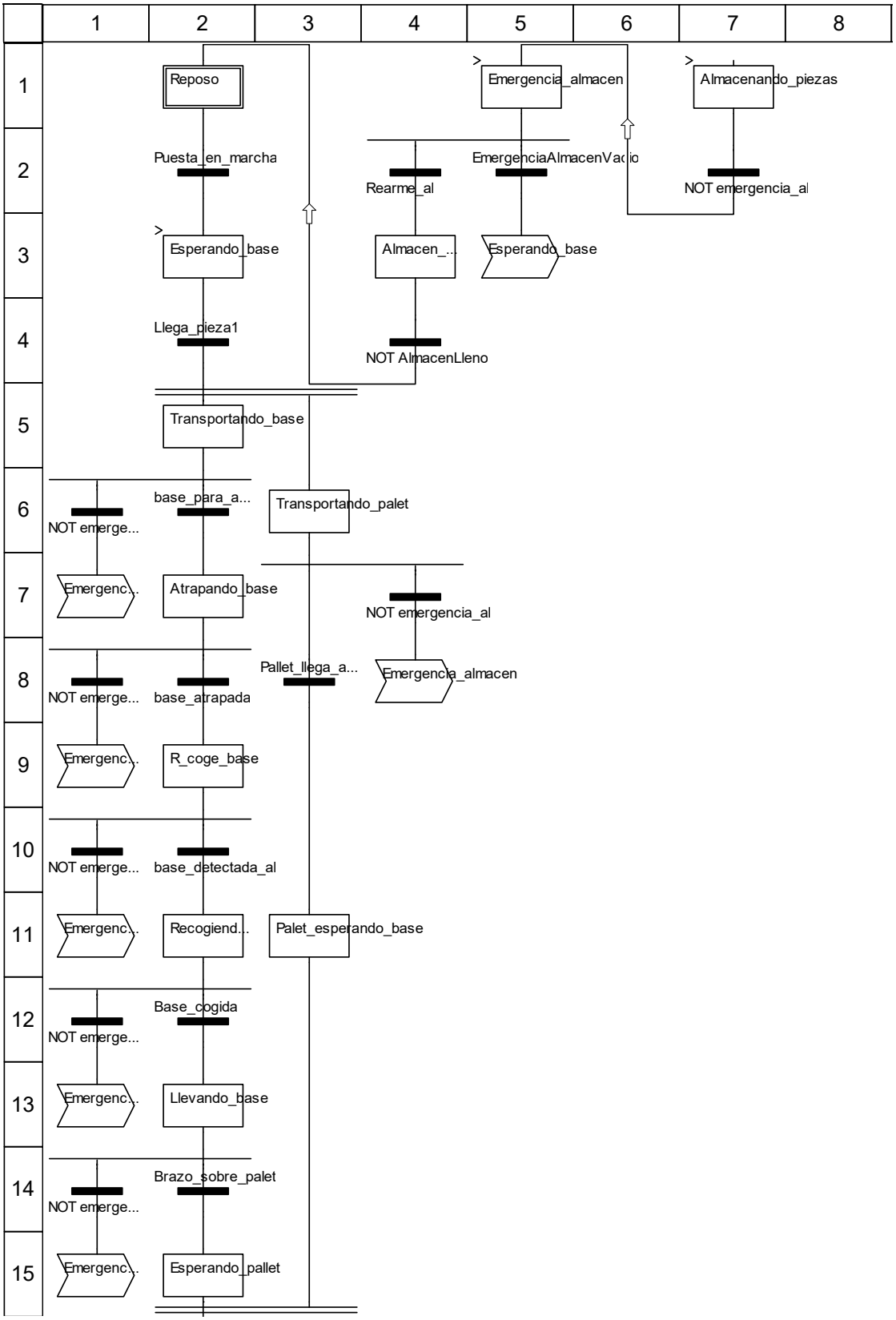
Propiedades comunes

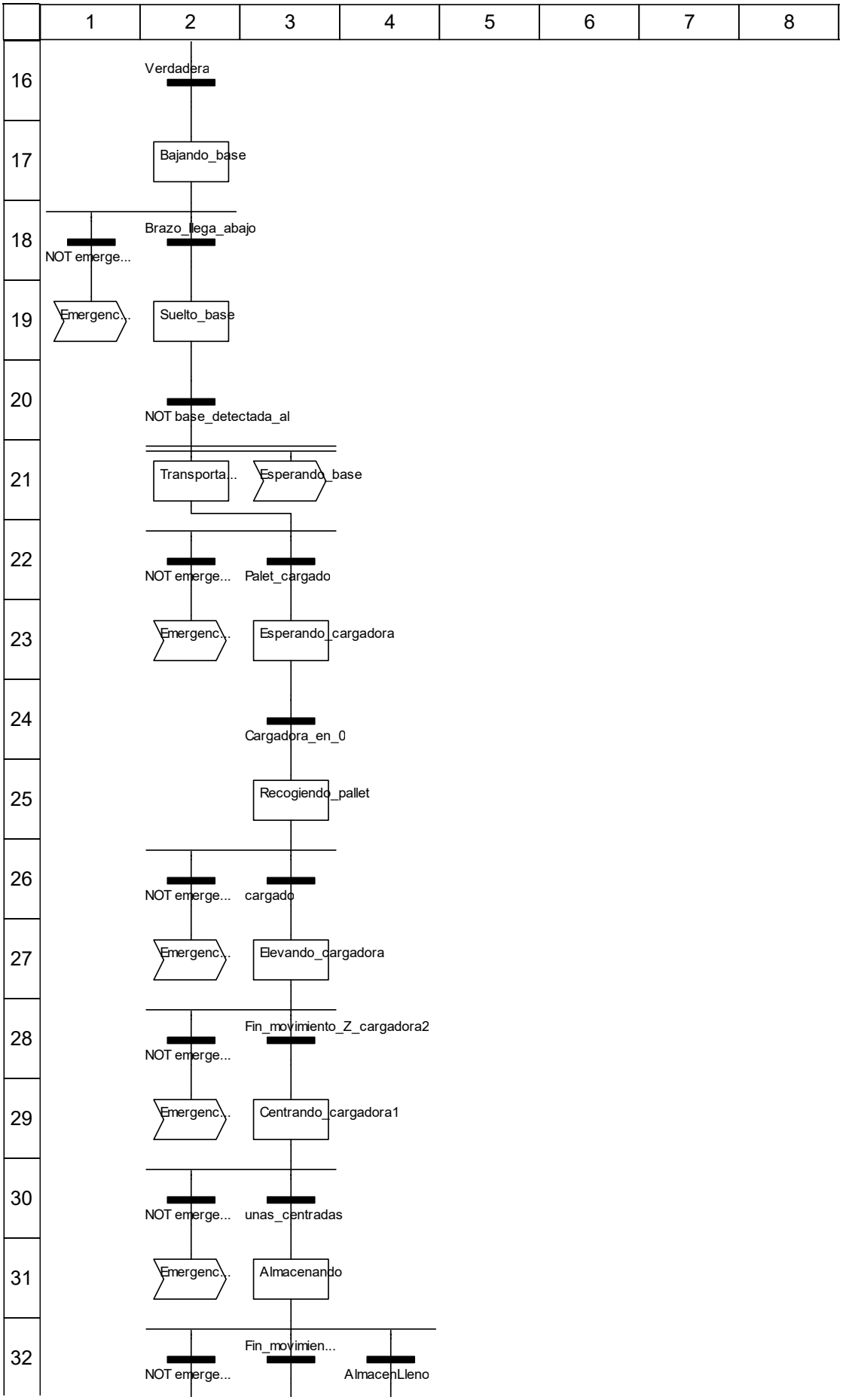
Módulo funcional	
Nombre de la condición	

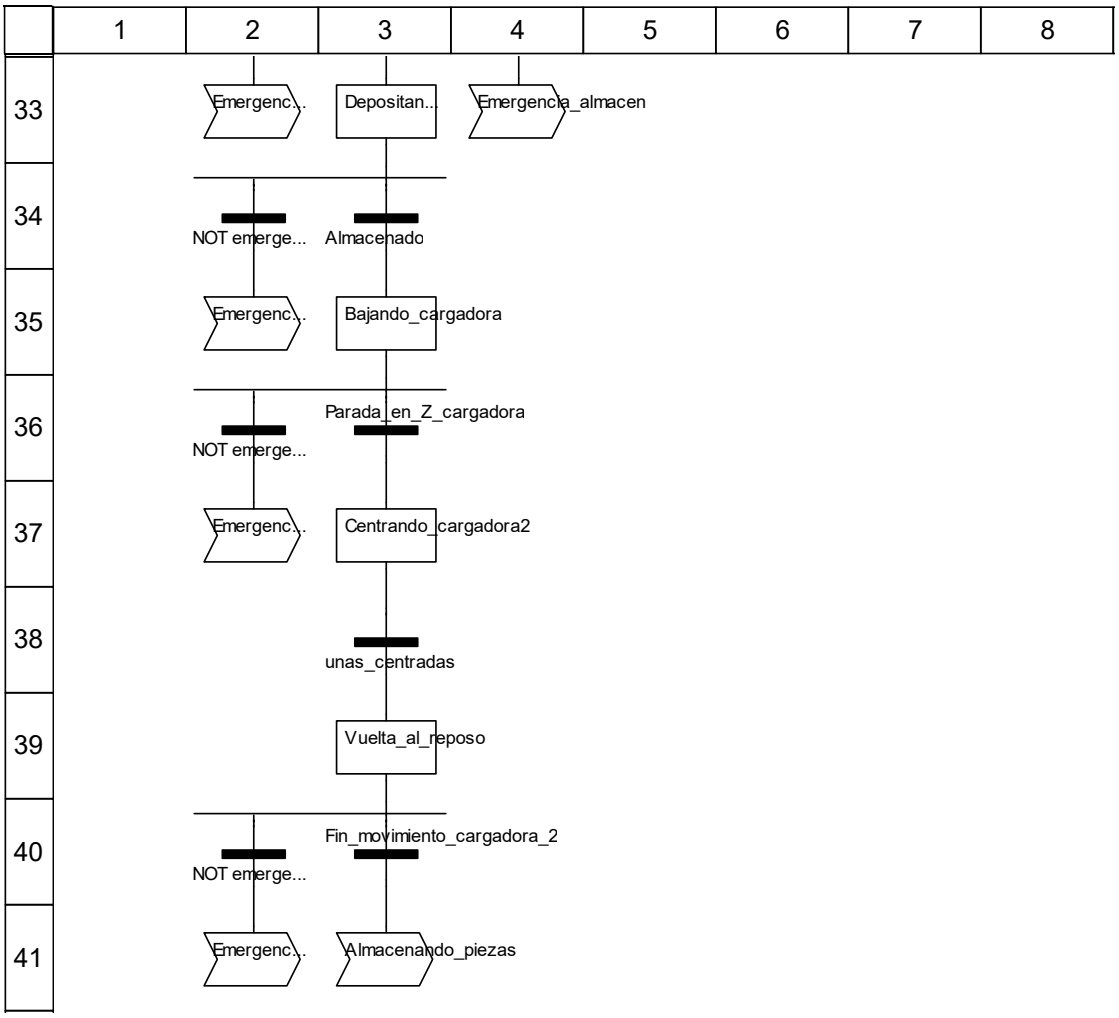
Propiedades específicas

Control de operador	No
Número de área	5

Chart : [MAST - ALMACEN]




1
2
3
4
5
6
7
8
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
Verdadera
Bajando_base
Brazo_llega_abajo
NOT emerge...
Emergenc...
Suelto_base
NOT base_detectada_al
Transporta...
Esperando_base
NOT emerge...
Palet_cargado
Emergenc...
Esperando_cargadora
Cargadora_en_0
Recogiendo_pallet
NOT emerge...
cargado
Emergenc...
Elevando_cargadora
Fin_movimiento_Z_cargadora2
NOT emerge...
Emergenc...
Centrando_cargadora1
NOT emerge...
unas_centradas
Emergenc...
Almacenando
Fin_movimien...
NOT emerge...
AlmacenLleno



Descripción de objeto

Pasos:

Almacen_vaciado		(4, 3)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: R	Tiempo:	Variable: AlmacenLleno

Almacenando		(3, 31)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Elevar_cargadora
Descriptor: N	Tiempo:	Sección: ST :: Posicion_almacen

Almacenando_piezas	(7, 1)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	

Atrapando_base	(2, 7)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	
Acciones:	

Descriptor: N	Tiempo:	Variable: atrapa_base

Bajando_base		(2, 17)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: baja_brazo_al
Descriptor: None	Tiempo:	Variable: Mover_brazo_al
Descriptor: None	Tiempo:	Variable: Coge_placa_al

Bajando_cargadora		(3, 35)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: almacenar

Centrando_cargadora1		(3, 29)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		

Centrando_cargadora2		(3, 37)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		

Depositando_palet		(3, 33)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: almacenar
Descriptor: None	Tiempo:	Variable: Elevar_cargadora

Elevando_cargadora		(3, 27)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Elevar_cargadora

Emergencia_almacen		(5, 1)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Baliza_al
Descriptor: S	Tiempo:	Variable: emergenciaAL

Esperando_base		(2, 3)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: R	Tiempo:	Variable: estacion2
Descriptor: R	Tiempo:	Variable: emergenciaAL

Esperando_cargadora		(3, 23)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		

Esperando_pallet		(2, 15)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Mover_brazo_al
Descriptor: None	Tiempo:	Variable: Coge_placa_al

Llevando_base		(2, 13)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Mover_brazo_al
Descriptor: None	Tiempo:	Variable: Coge_placa_al

Palet_esperando_base		(3, 11)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: D	Tiempo: t#3s	Variable: barrera

R_coge_base		(2, 9)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: N	Tiempo:	Variable: atrapa_base
Descriptor: N	Tiempo:	Variable: baja_brazo_al

Recogiendo_base		(2, 11)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: N	Tiempo:	Variable: baja_brazo_al
Descriptor: N	Tiempo:	Variable: Coge_placa_al

Recogiendo_pallet		(3, 25)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: cargando

Reposo (paso inicial)		(2, 1)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Sección: ST :: posicion_inicial
Descriptor: S	Tiempo:	Variable: primer_ciclo

Suelto_base		(2, 19)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		

Transportando_almacen		(2, 21)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: cinta_carga

Descriptor: None	Tiempo:	Variable: cinta_suministro
Descriptor: None	Tiempo:	Sección: ST :: suma_pos
Descriptor: R	Tiempo:	Variable: primer_ciclo

Transportando_base			(2, 5)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: N	Tiempo:	Variable: cinta_bases	
Descriptor: N	Tiempo:	Sección: ST :: Tipo_de_bases	
Descriptor: S	Tiempo:	Variable: estacion2	

Transportando_palet			(3, 6)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: D	Tiempo: t#3s	Variable: barrera	
Descriptor: N	Tiempo:	Variable: cinta_suministro	

Vuelta_al_reposo			(3, 39)
Tiempo de supervisión mín./máx.:			Tiempo de retardo:
Comentario:			
Acciones:			
Descriptor: None	Tiempo:	Sección: ST :: Cargadora_en_reposo	

Transiciones:

Nombre	Tipo de condición	Posición	Comentario
AlmacenLleno	Variable	(4, 32)	
Almacenado	Variable	(3, 34)	
ST :: Base_cogida	Sección	(2, 12)	
ST :: Brazo_llega_abajo	Sección	(2, 18)	
ST :: Brazo_sobre_palet	Sección	(2, 14)	
Cargadora_en_0	Variable	(3, 24)	
ST :: EmergenciaAlmacenVacio	Sección	(5, 2)	
ST :: Fin_movimiento_Z_cargadora2	Sección	(3, 28)	
ST :: Fin_movimiento_cargadora	Sección	(3, 32)	
ST :: Fin_movimiento_cargadora_2	Sección	(3, 40)	
ST :: Llega_pieza1	Sección	(2, 4)	
NOT AlmacenLleno	Variable	(4, 4)	
NOT base_detectada_al	Variable	(2, 20)	
NOT emergencia_al	Variable	(1, 6)	
NOT emergencia_al	Variable	(1, 8)	
NOT emergencia_al	Variable	(1, 10)	
NOT emergencia_al	Variable	(1, 12)	
NOT emergencia_al	Variable	(1, 14)	
NOT emergencia_al	Variable	(1, 18)	
NOT emergencia_al	Variable	(2, 22)	
NOT emergencia_al	Variable	(2, 26)	
NOT emergencia_al	Variable	(2, 28)	
NOT emergencia_al	Variable	(2, 30)	
NOT emergencia_al	Variable	(2, 32)	
NOT emergencia_al	Variable	(2, 34)	
NOT emergencia_al	Variable	(2, 36)	

NOT emergencia_al	Variable	(2, 40)	
NOT emergencia_al	Variable	(4, 7)	
NOT emergencia_al	Variable	(7, 2)	
Palet_cargado	Variable	(3, 22)	
ST :: Pallet_llega_a_cargar	Sección	(3, 8)	
ST :: Parada_en_Z_cargadora	Sección	(3, 36)	
ST :: Puesta_en_marcha	Sección	(2, 2)	
Rearme_al	Variable	(4, 2)	
ST :: Verdadera	Sección	(2, 16)	
base_atrapada	Variable	(2, 8)	
base_detectada_al	Variable	(2, 10)	
ST :: base_para_atrapar	Sección	(2, 6)	
cargado	Variable	(3, 26)	
unas_centradas	Variable	(3, 30)	
unas_centradas	Variable	(3, 38)	

Saltos:

Nombre	Posición	Comentario
Almacenando_piezas	(3, 41)	
Emergencia_almacen	(1, 7)	
Emergencia_almacen	(1, 9)	
Emergencia_almacen	(1, 11)	
Emergencia_almacen	(1, 13)	
Emergencia_almacen	(1, 15)	
Emergencia_almacen	(1, 19)	
Emergencia_almacen	(2, 23)	
Emergencia_almacen	(2, 27)	
Emergencia_almacen	(2, 29)	
Emergencia_almacen	(2, 31)	
Emergencia_almacen	(2, 33)	
Emergencia_almacen	(2, 35)	
Emergencia_almacen	(2, 37)	
Emergencia_almacen	(2, 41)	
Emergencia_almacen	(4, 8)	
Emergencia_almacen	(4, 33)	
Esperando_base	(3, 21)	
Esperando_base	(5, 3)	

Cargadora_en_reposo <Acción> : [MAST - ALMACEN]

```
  1|      10|      20|      30|      40|      50|      60|      70|      80|      90|      100|      110|
1  lugar_almacen:=55;
```

Tipo_de_bases <Acción> : [MAST - ALMACEN]

```
  1|      10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|
1  if tipo_base=3 then
2          base_azul:=1;
3          base_verde:=0;
4  end_if;
5  if tipo_base=6 then
6          base_verde:=1;
7          base_azul:=0;
8  end_if;
```

posicion_inicial <Acción> : [MAST - ALMACEN]

```
  1|      10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|
1  if emergenciaAL=0 then
2          pos_verde:=0;
3          pos_azul:=55;
4  end_if;
5  if emergenciaAL=1 then
6          pos_verde:=pos_verde;
7          pos_azul:=pos_azul;
8  end_if;
9
```

Posicion_almacen <Acción> : [MAST - ALMACEN]

```
1|      10|      20|      30|      40|      50|      60|      70|      80|      90|      100|      110|
```

```
1  if pos_verde <> pos_azul then
2          if base_verde=1 and suma_posicion=1 then
3                  lugar_almacen:=pos_verde+1;
4                  pos_verde:=lugar_almacen;
5                  suma_posicion:=0;
6          end_if;
7          if base_azul=1 and suma_posicion=1 then
8          lugar_almacen:=pos_azul-1;
9          pos_azul:=lugar_almacen;
10         suma_posicion:=0;
11         end_if;
12 else
13         AlmacenLleno:=1;
14 end_if;
```

suma_pos <Acción> : [MAST - ALMACEN]

```
  1|       10|       20|       30|       40|       50|       60|       70|       80|       90|       100|       110|
1  suma_posicion:=1;
```

Base_cogida <Transición> : [MAST - ALMACEN]

```
  1|      10|      20|      30|      40|      50|      60|      70|      80|      90|      100|      110|
1  Recogiendo_base.t>t#0.5s
```

Brazo_sobre_palet <Transición> : [MAST - ALMACEN]

```
  1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|

1   FE(brazo_moviendo)=1
2
```

Brazo_llega_abajo <Transición> : [MAST - ALMACEN]

```
  1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|

1   FE(brazo_bajando_al)=1
2
```

Verdadera <Transición> : [MAST - ALMACEN]

```
  1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|
1   (true and FE(almacenado)=1) or (true and primer_ciclo=1)
```

Fin_movimiento_cargadora <Transición> : [MAST - ALMACEN]

```
  1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|

1   (FE(Mov_X_cargadora)=1 and Mov_Z_cargadora=0 and AlmacenLleno=0)or (Mov_X_cargadora=0 and FE(Mov_Z_cargadora)=
1>>1 and AlmacenLleno=0)
```

Fin_movimiento_cargadora_2 <Transición> : [MAST - ALMACEN]

```
  1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|
1   (FE(Mov_X_cargadora)=1 and Mov_Z_cargadora=0)or (Mov_X_cargadora=0 and FE(Mov_Z_cargadora)=1)
```

base_para_atrapar <Transición> : [MAST - ALMACEN]

```
  1|      10|      20|      30|      40|      50|      60|      70|      80|      90|      100|      110|
1  FE(base_en_carga)=1
```

Palet_en_cargadora <Transición> : [MAST - ALMACEN]

```
  1|      10|      20|      30|      40|      50|      60|      70|      80|      90|     100|     110|
1  FE(palet_cargado)=1
```

Parada_en_Z_cargadora <Transición> : [MAST - ALMACEN]

```
  1|      10|      20|      30|      40|      50|      60|      70|      80|      90|     100|     110|
1  FE(Mov_Z_cargadora)=1
```

Fin_movimiento_Z_cargadora2 <Transición> : [MAST - ALMACEN]

```
  1|      10|      20|      30|      40|      50|      60|      70|      80|      90|      100|      110|
1  FE(Mov_Z_cargadora)=1
```

Pallet_llega_a_cargar <Transición> : [MAST - ALMACEN]

```
  1|      10|      20|      30|      40|      50|      60|      70|      80|      90|     100|     110|
1  RE(palet_en_espera)=1
```

Llega_pieza1 <Transición> : [MAST - ALMACEN]

```
  1|      10|      20|      30|      40|      50|      60|      70|      80|      90|      100|      110|
1  RE(llega_pieza)=1 and DosEstaciones=1
```

Cargadora_reposando <Transición> : [MAST - ALMACEN]

```
  1|      10|      20|      30|      40|      50|      60|      70|      80|      90|     100|     110|
1  FE(cargadora_en_0)=1
```

EmergenciaAlmacenVacio <Transición> : [MAST - ALMACEN]

```
  1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|
1  emergencia_AL=1 and AlmacenLleno=0
```

Puesta_en_marcha <Transición> : [MAST - ALMACEN]

```
  1|      10|      20|      30|      40|      50|      60|      70|      80|      90|      100|      110|
1  marcha=1 or marcha_al=1
```