Universidad Zaragoza
1542

Trabajo Fin de Grado

Desarrollo de una planta virtual en Factory I/O y control mediante PLC

Autor

Fernando Grima Montesa

Directores

José Ramón Asensio Diago

Cristian Florentín Mahulea

Departamento de Informática e Ingeniería de Sistemas

Escuela de Ingeniería y Arquitectura

2019

Índice General:

Índice de Figuras

Resumen

En los últimos años han ido apareciendo nuevos simuladores relacionados con la automatización, con posibilidad de conexión física con los controladores lógicos programables (PLC por sus siglas en inglés) encargados de controlar la automatización en fábricas. Algunos ejemplos de simuladores son: autoSIM-200, Factory I/O, PC_Simu, Zen Software, etc.

En general, cuando una empresa desea implantar nuevos sistemas de control en sus estaciones/cadenas de automatización optan por comprobar el correcto funcionamiento de dichos sistemas antes implementarlos. Es de gran ayuda tener la posibilidad de verificar estos sistemas de control en entornos en los que no se produzca ningún tipo de riesgo en caso de fallo, por necesitar alguna modificación en el código que puede ser debida a un no deseado o mal funcionamiento. Estos simuladores cumplen dicho propósito permitiendo la validación de los programas a implementar en los PLC's en un entorno virtual, que se puede modificar con gran facilidad y en el menor espacio posible. Entre otras cosas, también ayudan a comprobar la comunicación entre los diferentes equipos.

Real Games ha desarrollado un nuevo software de simulación virtual en 3D capaz de conectar con una amplia gama de autómatas pertenecientes a diversas compañías como pueden ser: Schneider, Siemens, Allen-Bradley, etc. Factory I/O permite la simulación de estaciones virtuales sobre las que poder verificar nuevos sistemas de control desarrollados mediante una conexión física con autómatas programables como ya se ha comentado.

El propósito de este trabajo fin de grado es el de evaluar la posibilidad de utilizar la simulación de una célula de fabricación, situada en el edificio Ada Byron, que se emplea para realizar prácticas de diferentes asignaturas. Por ejemplo, en el Grado de Tecnologías Industriales se utiliza en la asignatura de *Ingeniería de Control*. La finalidad de este proyecto es la de valorar la posibilidad de facilitar y mejorar el aprendizaje del alumno con nuevas herramientas como este nuevo software de simulación conocido como Factory I/O, cuyo contenido abre la puerta a una infinidad de posibilidades y aplicaciones.

Este trabajo parte de las bases establecidas en la práctica 4 que se desarrolla en la asignatura de Ingeniería de Control anteriormente mencionada. El objetivo principal que se desea aplicar sobre la práctica es que, dentro del programa Unity[1], el alumno solo diseñe un único código SFC que sirva tanto para el software de simulación como para el autómata

[1] Se dejará adjunto en la bibliografía un enlace del manual de ambos softwares[6][10]

programable que controla la estación real. Este objetivo se logrará mediante la creación de unas interfaces que consigan hacer que el código se comporte de igual manera para ambos casos. Para ello se llevará a cabo el diseño de la estación dentro del simulador Factory I/O y la posterior programación de estas interfaces dentro del software Unity Pro del fabricante Schneider ya que los autómatas con los que se trabaja en la escuela son Modicon M340 de dicha compañía. El alumno quedará ajeno a la información que posean estas interfaces así como a la creación diseño del modelo virtual (se le proporcionará ya ambos archivos), el propósito es que el estudiante solo trabaje con Unity.

Gracias a estas interfaces, una vez el alumno haya realizado correctamente el código a ejecutar (con el funcionamiento deseado), simplemente tendrá que activar o desactivar un botón dentro de la estación o dentro de Factory I/O para poder pasar de ejecutar su código en el simulador a la estación real. Por tanto, la correcta programación de las interfaces y su adecuada conexión con ambos dispositivos será la parte crucial del proyecto.

Esta conexión se debe a que Factory I/O no usa las mismas posiciones de memoria que se usan dentro de la estación real, tanto por diferencia de variables como por su cantidad.

Para evitar una posible confusión y con el fin de darle orden y claridad al código, dentro del propio Unity se trabajará con unas variables tipo Bool, que según el código que desarrolle el alumno, activarán las respectivas variables dentro de la estación o dentro de Factory I/O. Dependiendo de la interfaz que esté activada, la lectura de datos provendrá de un modelo[2] u otro, ejecutándose así las acciones correspondientes de los actuadores dentro de cada modelo, obteniéndose así el funcionamiento deseado.

Es decir, en caso de estar activada la interfaz de simulación, Unity solo trabajará con el autómata programable al cargo de la simulación y tomará la lectura de datos de los sensores colocados en esta. A su vez, solo dará valores a los actuadores de la simulación, dejando sin modificar las variables pertenecientes a la estación real. De modo gráfico se puede observar en la figura 1 que, una vez activadas dichas condiciones, se opta por utilizar dos "vías de comunicación" excluyentes la una de la otra, la activación de una elimina la posibilidad de usar la otra. Esto hace que aunque se modifiquen las variables de Unity solo afecte a las variables de un modelo o del otro.

[2] *Modelo* se refiere al conjunto autómata-estación en el caso del modelo de la estación real y en el caso de la simulación al conjunto del autómata con el diseño creado virtualmente

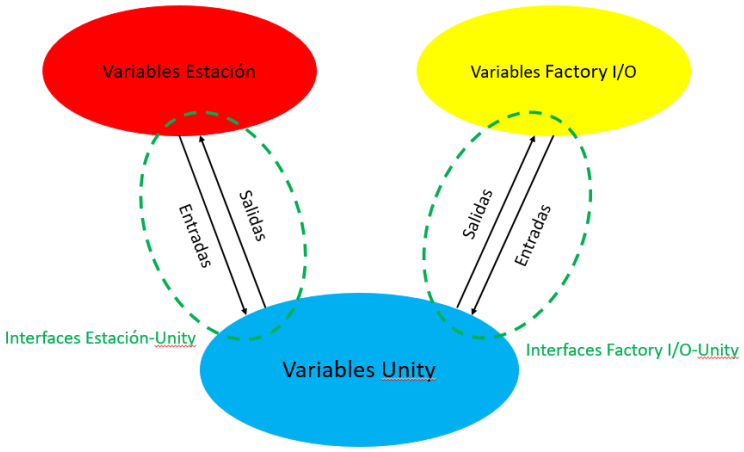


Figura 1: Esquema del empleo de variables

Este trabajo fin de grado se ha realizado en el Departamento de Informática e Ingeniería de Sistemas. La metodología principal llevada a cabo dentro de las interfaces anteriormente comentadas ha sido un estudio en profundidad de las diferencias existentes entre el modelo utilizado hasta ahora, en el cual hay una estación real y un PLC que la controla, y el modelo añadido, para lo cual el programa del PLC debe ampliarse y adaptarse a las diferencias que provienen de la utilización de un modelo alternativo basado en el simulador programado en Factory I/O.

Se ha escogido realizar el control de la planta (control que desarrollará el alumno) en lenguaje SFC (*Sequential Function Chart*, introducido en el apartado 4.4), implementando una solución concurrente en vez de hacerlo de manera secuencial. Posteriormente, en la sección de emergencia y rearme, se ha seguido la metodología que propone la guía GEMMA (comentada en el apartado 4.3) en la cual básicamente se propone hacer un sistema de eventos discretos, al igual que en el control de la planta, pero al ser este más sencillo se ha escogido hacerlo en lenguaje ST (*Structured Text* o texto estructurado), por temas que se explicarán más adelante.

Capítulo I – Motivación y objetivos

1.1 Motivación del proyecto

Este trabajo fin de grado es objeto de una búsqueda continua de la mejora de las condiciones de aprendizaje del alumno. De forma razonable, la carga de horas para un estudiante de un grado es mayor en conocimientos teóricos que en prácticos, pero con la intención de optimizar el tiempo dedicado por el estudiante a éstos últimos, se ha querido agilizar los tiempos de desarrollo de algunas prácticas, para así poder profundizar más en los temas tratados en ellas. En concreto, en este caso, son aquellas que realizan los alumnos para la aplicación de conocimientos teóricos acerca de los sistemas de eventos discretos.

El problema en sí, surge de la cantidad de alumnos que hay en los grupos de prácticas. El gran tamaño de los grupos hace que resulte un poco tediosa la realización de prácticas del estilo de la práctica 4 de Ingeniería de Control del Grado de Tecnologías Industriales; esa gran cantidad de alumnos trabaja con una sola estación. A la cual, en un momento dado, puede acceder solo un alumno para comprobar el correcto funcionamiento de su código. Por ejemplo, no se llega a implementar una sección de tratamiento de emergencias en el sistema de control, debido a la falta de tiempo.

Entonces la instalación de una nueva plataforma interactiva, con la que el alumno pudiese probar su programa sin tener que esperar a los demás alumnos y sin correr ningún riesgo (por un posible fallo en su programación), era un hecho que atraía de forma bastante llamativa para intentar introducirlo en las prácticas. Así se conseguiría una reducción de pérdidas de tiempo durante la práctica y se podría llegar a implementar dicha sección de tratamiento de emergencias como se ha realizado en el ejemplo llevado a cabo en este proyecto.

1.2 Objetivos del proyecto

Como se ha comentado en el apartado anterior, el objetivo básico de este proyecto fin de grado es aumentar y mejorar los conocimientos prácticos de los alumnos en el ámbito de la automatización industrial. Para ello, a través de una búsqueda intensiva de nuevas herramientas que permitiesen facilitar dicha tarea, se encontró este nuevo software de simulación de células y estaciones de automatización industrial conocido como Factory I/O. Se pretende desarrollar el modelo virtual de una planta industrial simple para la implementación de un sistema de control por parte de los estudiantes. Se ejecutará en un autómata programable y posteriormente se verificará su funcionamiento en el modelo real sin tener que añadir cambios al código generado. Con el fin de poder aplicar así de manera más eficiente los conocimientos teóricos adquiridos por alumno en el ámbito de modelado, análisis y diseño de sistemas de eventos discretos.

Una vez instalado el programa Factory I/O y hecho un ejemplo[7] para verificar las conexiones con los autómatas programables disponibles en el laboratorio L0.06, el siguiente punto era el poder ajustar un modelo a la realidad. Este diseño ha sido el principal reto y dificultad de este trabajo, comprobar si realmente existía la posibilidad de poder crear un modelo virtual que se asemejase a la estación real.

Otro de los objetivos es desarrollar un guion de prácticas con el añadido de la parte de simulación, así como una plantilla general del programa Unity para el desarrollo de la nueva práctica. Se estudiará además la posibilidad de utilizar un protocolo OPC (*OLE*[3] *for Process Control*) en la comunicación con el autómata.

El gran dilema de las compañías dependientes de la automatización ha sido siempre la comunicación. Cuando una empresa trabaja con diversos dispositivos especializados en la automatización o sistemas de control se ve en un gran aprieto cuando tiene que conseguir comunicar dichos componentes con el resto de la industria; para dar u obtener información por parte de ambos extremos ya que no todos los componentes se comunican de la misma manera. OPC es un estándar para la conectividad de datos basado en la metodología maestro/esclavo. Este sistema de comunicación busca abstraer la información de la implementación de las Fuentes de Datos y de los Clientes de Datos, con el objetivo de que los datos se puedan intercambiar sin tener en cuenta los protocolos de comunicación interna que tuviesen

[3] OLE: *Object Linking and Embedding*

implementados. Para ello, OPC consta de dos partes: Cliente y Servidor. Los protocolos nativos que tuviesen la Fuente de Datos y el Cliente se comunican con una de las dos partes y son estas las que se comunican entre sí para intercambiar la información entre la aplicación y el dispositivo. Consiguiendo de esta manera que ambos puedan contactar directamente entre sí sin la necesidad de drivers[8].

1.3 Estructura del documento

En esta sección se va a hablar del contenido de la memoria. Se comenzará con una pequeña introducción sobre los dos programas que se van a utilizar, su historia y el ejemplo de ciertas situaciones en las que se emplean ambos programas dentro del mundo industrial. A continuación, en el Capítulo III, se analizará una sección de las dos situaciones en las que nos encontramos, junto con una pequeña sección describiendo el tipo de variables que se utilizarán. Primero se presentará la estación de la célula de fabricación disponible en el laboratorio L0.06 que se va a considerar en este trabajo: descripción general, actuadores y sensores. Luego se pasará a detallar el diseño desarrollado en el software de simulación, con la intención de que el lector comprenda qué elementos se han escogido para simular y qué dificultades se han encontrado por el camino.

En el Capítulo IV se explica con detalle todas las secciones del programa realizado. Empezando por las secciones donde se leen las variables de entrada y se realiza el cambio de modo (simulación vs realidad), continuando con el ejemplo realizado de las secciones de emergencia y del código implementado en el autómata, y finalizando con las secciones en las que se da valor a las salidas.

Los últimos dos Capítulos servirán para darle validez a todo aquello que se ha realizado en el trabajo, en el primero se adjuntará un enlace a un video explicativo (con el que se podrá visualizar cómo funciona el simulador) junto a algunas capturas y en el último se recaban las conclusiones y resultados obtenidos al finalizar el trabajo, así como su alcance.

Capítulo II – Acerca de Unity Pro y Factory I/O

Entrada la década de los 60 el sistema de control de maquinaria se realizaba mediante sistemas controlados por relés mecánicos, contactores, temporizadores analógicos, etc. Era un sistema difícil de mantener debido a la vida útil de cada elemento y que en caso de realizar alguna variación en el proceso de producción se debía de rehacer el proceso de cableado del sistema de control cosa que consumía mucho tiempo y dinero. Se vio necesario desarrollar un sistema que realizase el mismo trabajo y no necesitase ese mantenimiento ni se viese afectado de gran manera con los cambios en el sistema de producción.

En 1968, el departamento Hydra-Matic de la corporación General Motor's solicitó una propuesta a este gran dilema. A esta solicitud acudió la compañía Bedford Associates la cual ganando el contrato se propuso crear dicho sistema. Gracias a un sistema de estado sólido flexible como una computadora pero no su nivel de coste, de fácil mantenimiento y capaz de trabajar en entornos de humedad, suciedad, vibración… Bedford logró diseñar el primer autómata programable (PLC por sus siglas en inglés *Programmable Logic Computer*), el *MOdular DIgital CONtroller* (Modicon). Dick Morley, uno de los diseñadores del Modicon 084 (fue el prototipo 84) es considerado el "padre" de los PLC.

En torno a la década de los 70, las comunicaciones entre los PLC se fueron desarrollando hasta llegar al punto en el que eran capaces de dialogar entre ellos. La red de comunicaciones se basó en una arquitectura Maestro/Esclavo pero en sus comienzos, debido a la falta de estandarización y otros problemas, estas fueron un verdadero caos. Ya en la década de los 80, se intentó mejorar dichas comunicaciones y, por ello, se creó el protocolo de automatización de fabricación MAP (*Manufacturing Automation Protocol*)[2][3].

Unity Pro surge de la integración del software Concept, con el que se podía programar los PLC's Quantum y Momentum, y del software PL7 Pro que servía para los PLC's Micro y Premium. A día de hoy es el software que se utiliza para la programación de los autómatas de Schneider. El entorno de desarrollo Unity permite trabajar con varios tipos de variables (desde tipos elementales como booleanos, enteros, etc., hasta tipos de datos estructurados como matrices y estructuras), y permite utilizar los cinco lenguajes estandarizados en la norma IEC 61131-3 para

la realización del programa del PLC. En este caso se ha escogido el lenguaje de programación SFC para la sección principal a desarrollar, por ser el más similar a lógicas como las de los autómatas finitos deterministas y las redes de Petri.

Por ejemplo, en mis prácticas del Grado que tuve la oportunidad de realizar en la compañía Cemex, pude observar cómo las grandes empresas utilizan los sistemas de control para regular los elementos principales de las fábricas, como en este caso eran el horno y los molinos para la fabricación de cemento y Clinker. Hecho que fomentó la motivación para continuar con el trabajo.

Por otro lado, la compañía Real Games lleva 12 años trabajando en el desarrollo de nuevas formas de aprendizaje con simuladores 3D[4]. Factory I/O es uno de estos simuladores, con el que se pretende posibilitar un aprendizaje acerca de los PLC fuera de un posible riesgo de lesiones, daños a equipos… y utilizando el mínimo espacio posible. Este software ofrece la oportunidad de trabajar con aplicaciones industriales comunes, o con nuevos escenarios creados por el usuario con los elementos industriales que ofrece su biblioteca, aplicando tecnologías avanzadas en su control. Funciona con todos los tipos de PLC (Allen-Bradley, Siemens, Schneider…) así como con otras tecnologías de automatización como microcontroladores, SoftPLC, Modbus, OPC…

Las características de Factory I/O descritas en el párrafo anterior nos han llevado a elegir este simulador y no otro para su empleo en este trabajo. Además de su rápido aprendizaje al ser un programa de simulación sencillo de utilizar, este software ofrece un gran campo de posibilidades al programador, pudiéndose crear desde sencillas aplicaciones industriales a complejas cadenas de montaje de un alto nivel de complejidad y programación. También incluye la opción de forzar los actuadores para poder comprobar las acciones que van a realizar antes de implementar, por ejemplo, un sistema de control como en nuestro caso. Otra herramienta de gran utilidad que ofrece el sistema, es la de ocasionar fallos en la estructura. Esto ayuda al programador a realizar un sistema de contramedidas para que, en caso de que ocurra algo similar, se esté preparado de antemano y así evitar los mínimos costes de producción posibles.

Capítulo III – Descripción de modelos

3.1 Descripción de la estación real

La estación escogida es la estación 3 de la célula de fabricación, disponible en el laboratorio L0.06 en el edificio Ada Byron. Esta célula de fabricación industrial tiene por objetivo la producción, almacenamiento y expedición de cilindros neumáticos. Posee un total de 8 estaciones separadas en 3 módulos distintos. En el primer módulo se dan las operaciones de selección del tipo de cilindro, la colocación en cilindro de émbolo y el muelle, el roscado de la tapa y la verificación final de la correcta colocación de los elementos. En el segundo módulo existe un almacén intermedio donde se guardan provisionalmente los cilindros individuales en espera de ser agrupados en pedidos conjuntos de 3 cilindros.

Por último, en el tercer módulo se selecciona el tipo de base para cada pedido conjunto, se colocan los cilindros provenientes del almacén intermedio en cada base mediante un brazo manipulador, y se almacenan los pedidos ya completos en el almacén final, listos para su expedición fuera de la célula utilizando de nuevo el brazo manipulador[9]. En particular la estación 3 es la encargada del roscado de la tapa.

La figura 2 muestra un esquema de la célula dividida en los módulos comentados y en la figura 3 se puede ver una imagen de la estación que se va a emplear:

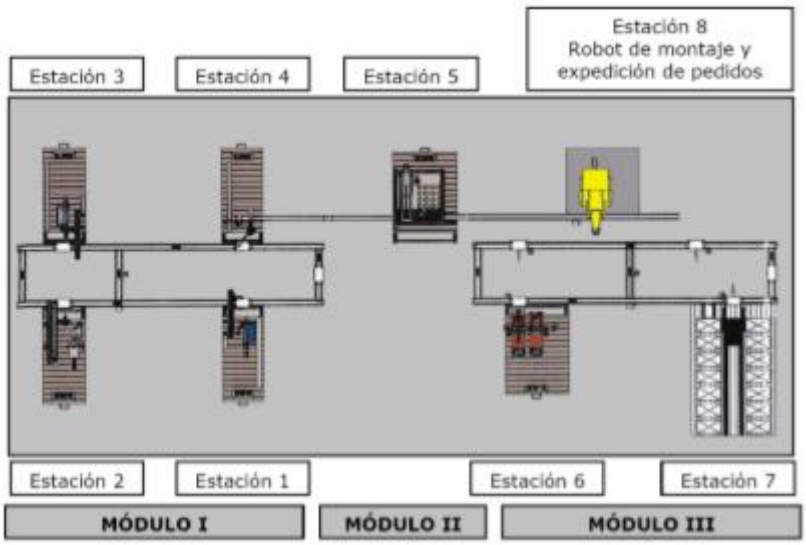


Figura 2: Esquema del conjunto de la célula de fabricación[4]

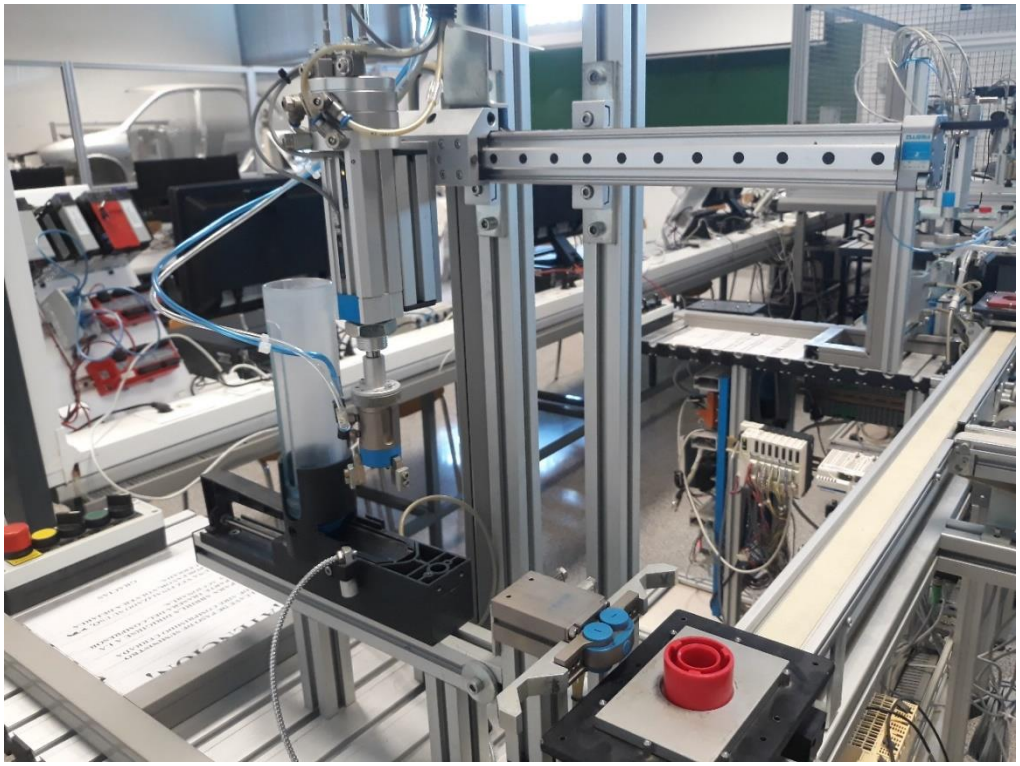


Figura 3: Foto de la estación 3 del laboratorio L0.06

Para que el autómata pueda realizar correctamente el control sobre la estación, necesita de elementos con los cuales reciba información del estado de la estación (Sensores) y de elementos sobre los cuales pueda actuar en caso de querer modificar ese estado de la estación (Actuadores). En primer lugar se van a detallar los distintos actuadores que posee la estación para posteriormente pasar al detalle de los sensores.

[4] Imagen recuperada del guion de la práctica del Grado de Tecnologías Industriales[9]

3.1.1 Actuadores

Los actuadores de esta estación son cilindros neumáticos de simple o doble efecto que se accionan con la puesta a 1 de una salida del autómata. Esta puesta a 1 de la salida abre una electroválvula, introduciendo aire a presión por una entrada del cilindro, y la puesta a 0 cierra dicha válvula. Para la actuación de un cilindro de simple efecto, en la electroválvula correspondiente, existe un muelle que la devuelve a su posición de reposo al dejar de ser activada. Por lo que el cilindro recibe aire por la otra entrada, devolviéndolo a su posición inicial. Por el contrario, en el caso de los cilindros de doble efecto se necesitan dos salidas del autómata para controlar su movimiento ya que cuando el cilindro llega a un extremo no torna a su posición inicial.

El brazo manipulador consta de un desplazador con posibilidad de movimiento en el eje paralelo a la mesa. Posee un cilindro de doble efecto que permite posicionar el desplazador delante y detrás, dos entradas de aire (como se ha mencionado anteriormente) con las cuales si se quiere posicionar el desplazador delante habrá que accionar la entrada de aire del lado izquierdo y viceversa; en la figura 4 se pueden apreciar ambas entradas de aire.

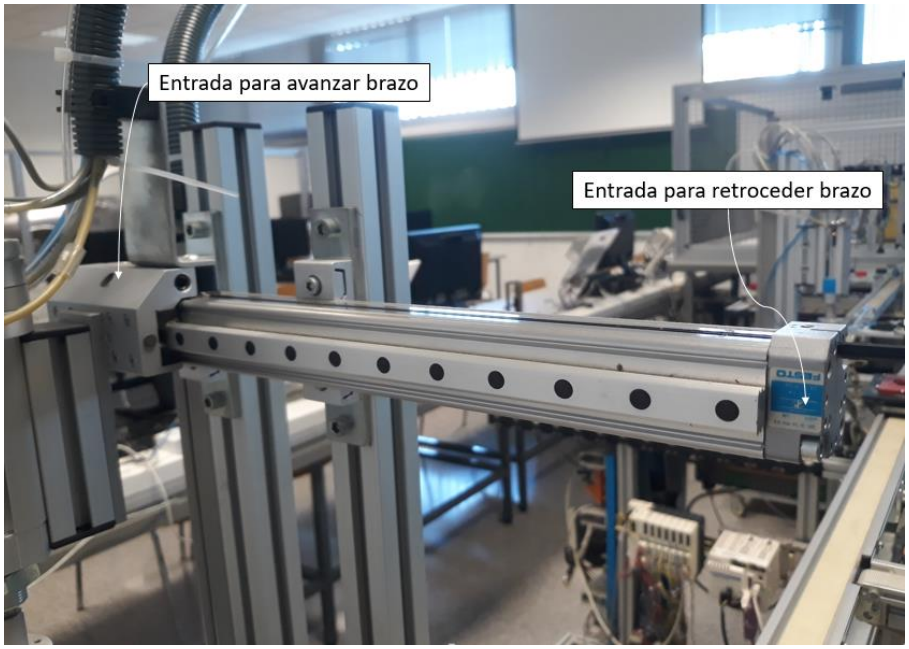


Figura 4: Actuador del brazo manipulador en el eje X

A continuación, todos los cilindros que se van a nombrar son de simple efecto.

El brazo manipulador posee un brazo vertical (ver figura 5) que es el encargado de bajar la pinza hasta la pieza deseada y con ella una vez colocada en posición, se cierra en torno a la pieza. Esta pinza puede girar en el sentido de las agujas del reloj para roscar la tapa en su base.

Figura 5: Conjunto pinza; actuador eje Z, cierre de pinza y actuador de roscado

Los dos elementos que quedan de explicar están fuera del brazo, que son: el dispensador de tapas encargado de colocar la tapa, que se va a roscar, debajo de la pinza (cuando el desplazador está atrás) y por otra parte la mordaza, que tiene la función de sujetar la base para que esté posicionada correctamente. Se pueden ver ambas en las imágenes adjuntas:

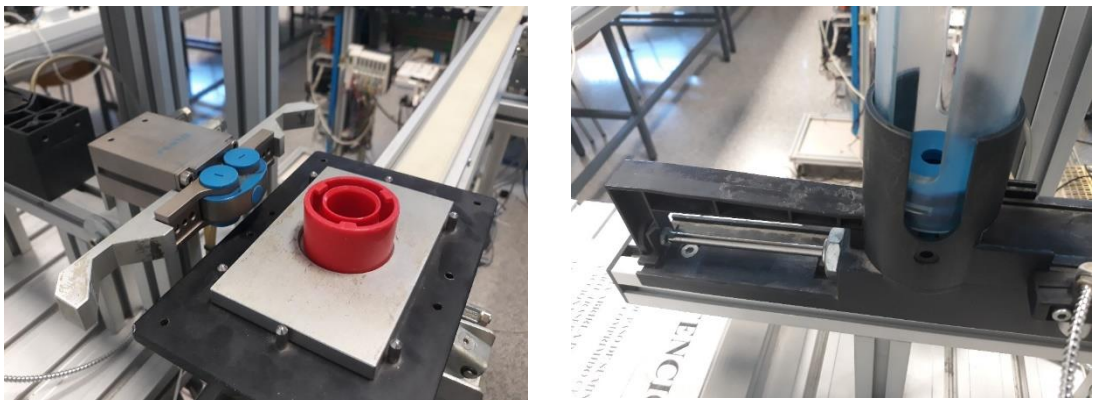


Figura 6: Mordaza (1) y dispensador de tapas (2)

3.1.2 Sensores

Los elementos utilizados para recabar información sobre el estado de la estación son los sensores. Son los encargados de determinar la posición de los distintos actuadores, detectan cuando un cilindro está colocado en cada uno de sus extremos. En el caso de la estación 3 son sensores inductivos. Están colocados en los extremos del desplazador horizontal y el brazo vertical para verificar si el brazo manipulador está delante, detrás, arriba o abajo; y en la pinza para saber si está girada a derechas o si está girada a izquierdas. Luego hay otro sensor

encargado de detectar si hay una tapa colocada en posición para que el brazo manipulador baje la pinza y la lleve a su posición de roscado. En este caso es un sensor óptico (ver figuras 7, 8 y 9).

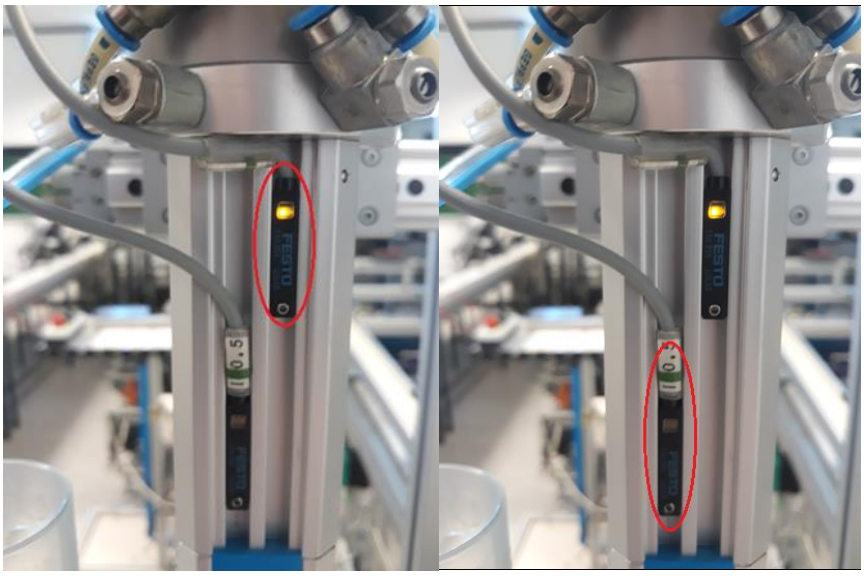


Figura 7: Sensores de `Pinza_arriba` (1) y `Pinza_abajo` (2)

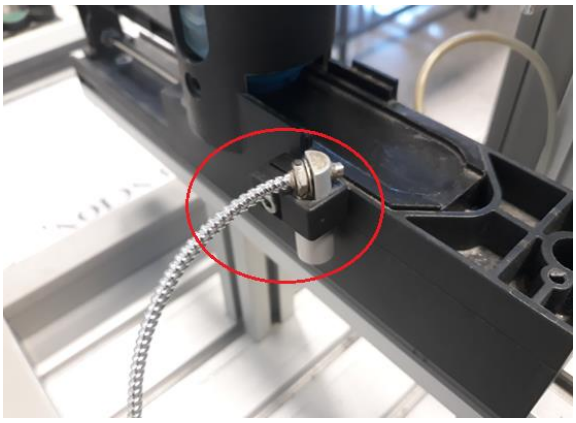


Figura 8: Sensor óptico de detección de tapa

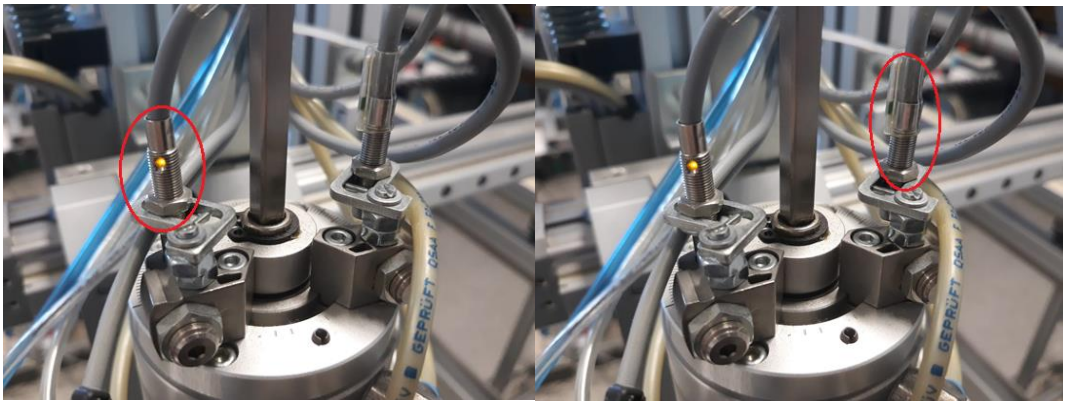


Figura 9: Sensores de giro a izquierdas (1) y a derechas (2)

Por otra parte, existen ciertos pulsadores en una botonera que son los encargados de activar la secuencia o detenerla en caso de emergencia.

3.2 Descripción de la estación en Factory I/O

Antes de comenzar, decir que en este apartado se va a explicar qué objetos/herramientas se han escogido dentro del simulador Factory I/O para poder utilizar el mismo programa SFC de Unity, tanto para simular la estación de manera virtual como para utilizarlo con la estación real. Hay que tener en consideración que no se dispone de todos los objetos presentes en el laboratorio, o en caso de tenerlos, no todos pueden ser colocados de la misma manera que en la célula. Por ello, la estructura escogida del modelo dentro de Factory I/O es el diseño que mejor se adapta a las condiciones reales (antes de empezar a trabajar con él, se llevó a cabo un proceso de selección dentro de un grupo de varios diseños y una posterior mejora del mismo). A continuación se encuentra una imagen del diseño final con el que se ha trabajado:

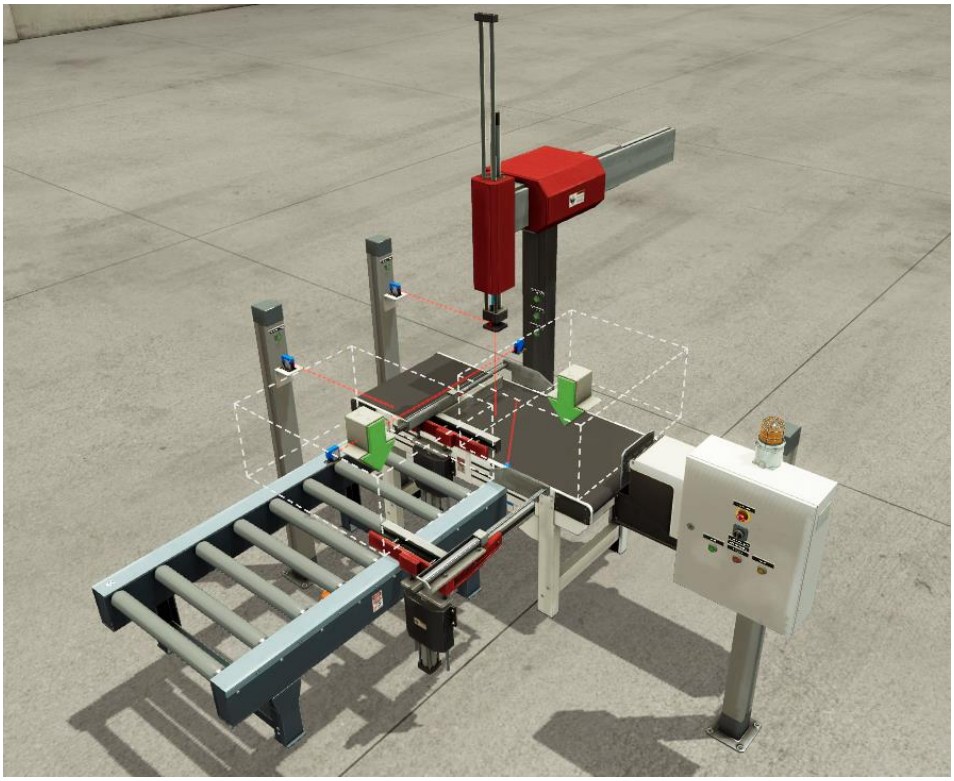


Figura 10: Estación 3 creada en el simulador

Como en la estación real, este diseño posee elementos de detección con los cuales recabar información del estado de la estación y elementos con los que actuar conforme a la información recibida.

3.2.1 Actuadores

El objeto principal de la estación es un brazo manipulador (al igual que en el modelo real) que es el encargado de colocar las tapas en la posición deseada. Dentro de él, el actuador encargado de avanzar y retroceder el brazo, en el eje paralelo a la mesa, es de tipo cilindro de simple efecto (primera diferencia con la estación real). En el apartado 4.5 de descripción de las interfaces se explicará cómo se han solucionado las distintas diferencias entre los dos modelos.

Continuando con el brazo manipulador, en uno de sus extremos tiene un cilindro de simple efecto que será el encargado de desplazar la pinza en dirección vertical. Al igual que en la estación real, la pinza tiene la capacidad de agarrar la pieza pero a diferencia de la pinza real, esta trabaja con succión en vez de con una garra. Al contrario que en la situación real, el cilindro neumático encargado de girar la pinza es de doble efecto, es decir, se necesitarán dos salidas (roscar a derechas y roscar a izquierdas) con las cuales la puesta en 1 activará el cilindro, haciendo girar la pinza un ángulo de 90º en la dirección de dicha salida. La puesta a 0 dejará la pinza como está permitiendo volver a activar dicha salida y volver a rotar, no hará que retorne a su posición de partida.

Después, del mismo modo que en la estación real, se dispone de un dispensador/empujador de cilindro de simple efecto encargado de colocar la tapa. Y de una mordaza también de cilindro de simple efecto que simulará a la mordaza real.

3.2.2 Sensores

Dentro del grupo de los sensores encontramos los siguientes tipos: sensores fotoeléctricos de reflexión difusa y sensores de movimiento. Los sensores difusores[5] se caracterizan por ser emisores de un haz de luz que se pierde en el espacio si no hay ningún objeto presente. En caso contrario, se produce una reflexión difusa sobre la superficie de dicho objeto con lo que parte del haz de luz incide sobre el receptor del sensor cambiando así la señal de salida de la fotocélula.

Como se puede observar en la figura 11, los sensores de movimiento están situados en el cuerpo del brazo manipulador, de todos ellos se utilizará el sensor de movimiento en el eje Z y el sensor de giro de la pinza. Estos sensores se encargan de detectar cuándo las partes móviles comentadas se encuentran en movimiento. A diferencia de los sensores difusores, estos no

[5] Se llamaran así para simplificar su nombre

necesitan de un objeto para modificar su señal, solo el propio movimiento de la máquina activa su señal.

Figura 11: Sensor de movimiento en eje Z (1) y sensor de movimiento de giro (2)

Por otro lado, los demás sensores por tema de comodidad en la parte de diseño son sensores difusores y están colocados en los lugares señalados en la figura 12.

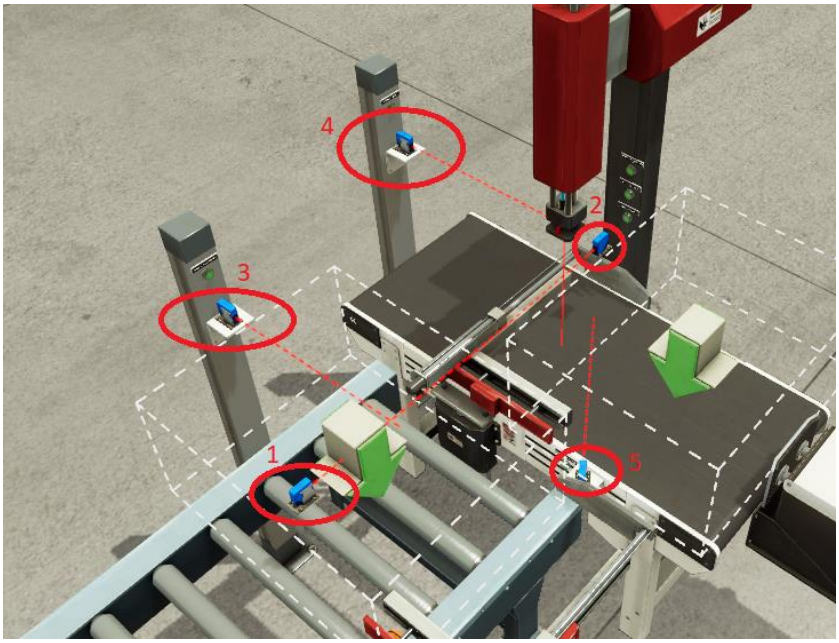


Figura 12: Numeración de los sensores difusores

Los sensores 1 y 2 se encargan de detectar cuando la pinza está abajo, los 3 y 4 para saber si el brazo está delante o detrás y el 5 detecta si la tapa ha llegado a la posición de recogida.

Con el fin de mejorar la visibilidad de la simulación, se han colocado ciertos indicadores luminosos junto a sus respectivos sensores para así saber cuándo se encuentran activados y cuándo no. Del mismo modo se ha colocado una luz de emergencia para saber si la seta está pulsada o no.

Por necesidad de adaptación de la estación real a su simulación, en el diseño virtual ha sido necesaria la incorporación de un número mayor de sensores (por ejemplo 2 sensores en vez de uno para detectar la pinza abajo). Mediante una ecuación booleana se consigue simular adecuadamente un sensor real a partir de la información obtenida por parte de varios sensores virtuales. Hecho que ha complicado un poco la programación de la solución final. Quedará mejor detallado en el apartado 4.2 cuando se habla de las variables `Pinza_Arriba` y `Pinza_Abajo`. Se puede ver a continuación dicha ecuación booleana que se explicará en profundidad en el apartado comentado.

```
if (FIO_pinza_abajo or FIO_pinza_abajo2) and not FIO_pinza_arriba
then
      Pinza_abajo:=1;
      Pinza_arriba:=0;
      else
      Pinza_abajo:=0;
end_if;
```

De la misma manera se ha simulado una especie de panel de control a modo de botonera desde el que se pondrá en funcionamiento el proceso o se detendrá en caso de fallo. Por otro lado, dentro de dicho panel de control se encuentra el botón que accionará la colocación de la base y la tapa puesto que no existe un dispensador que provea la tapa y la base para cada ciclo.

3.2.3 Objetos

De todo el diseño realizado hay objetos que poseen tanto actuadores como sensores pero no se utiliza ninguno de ellos puesto que solo sirven para ayudar al funcionamiento del resto, actúan de elementos secundarios.

La mordaza señalada en la figura 13 sirve de pared para no permitir que la tapa vaya más allá de la posición ideal, ya que el empujador se extiende más allá de esa posición y a la hora de posicionar la tapa debajo del brazo, no quedaba correctamente colocada en el lugar apropiado para su posterior ensamblaje. Por dicho motivo se tuvo que optar por colocar un objeto que sirviese de pared y a su vez permitiese el paso del haz del sensor para detectar cuando la pinza alcanzase la posición de agarre.

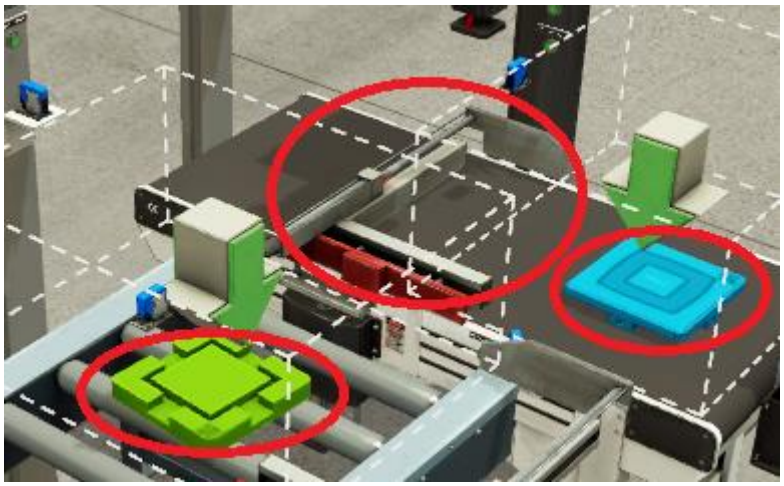


Figura 13: Objetos usados en la estación

Los otros dos elementos simulan la tapa y la base donde se quiere roscar. Como se puede observar, tienen forma rectangular (a fecha de hoy no existen tapas circulares en el entorno de desarrollo Factory I/O); este ha sido un gran problema a la hora de simular la rotación de las piezas ya que al bajar la pinza ambas piezas ya quedaban encajadas. Para simular la rotación se ha tenido que desplazar la mordaza (por tanto no sujetará la base, sino que simulará que lo hace) y hacer rotar el conjunto entero.

Con el fin de facilitar la rotación se ha colocado una cinta de rodillos para reducir así la fricción con ella y permitir un mejor giro. Esto ha sido necesario puesto que el entorno Factory I/O tiene en cuenta no solo la detección de colisiones entre los objetos, sino también su dinámica, de modo que en las ecuaciones que físicamente los modelan se consideran aspectos como: la fuerza de fricción, la gravedad terrestre, etc.

En el intento de conseguir un objeto que actuase mejor de tapa, se contactó con Real Games para preguntar si era posible introducir nuevos elementos a la biblioteca, respondieron que a día de hoy no era posible introducir nuevos objetos dentro del programa y que trabajarían en el desarrollo de esta opción para un futuro.

3.3 Descripción de las variables

Las variables usadas en Factory I/O son de tipo *extended boolean* (Ebool) caracterizadas por la ocupación de un espacio de 3 bits de memoria llevando con ellas su valor y la verificación de sus flancos de subida y de bajada. Y al contrario, las entradas y salidas de la estación son variables Bool de 1 bit de memoria, es decir, solo se tendrá la información del estado de dichas variables (puesta a 1 o a 0). Por lo tanto, al tratarse de datos a evaluar y emitir distintos se deben crear unas interfaces que consigan adaptar estas diferencias. Como se ha comentado en el resumen, se va a trabajar con unas variables dentro del código desarrollado en Unity y unas variables específicas para cada modelo.

Visualmente queda muy bien ejemplificado con la variable `Roscar`. El alumno cuando quiera roscar la tapa, solo tendrá que activar esta variable `Roscar` de Unity (tipo Bool) y desactivarla una vez roscada la tapa por ser cilindro de simple efecto (como ya se ha explicado anteriormente). Sin embargo, dentro de las interfaces, hay un trabajo más complejo para que el sistema virtual entienda que cuando está variable se active, se produzca el acto de roscar y cuando se desactive ocurra lo contrario (por las diferencias ya comentadas en los apartados 3.1 y 3.2). Debido a estas diferencias, en la interfaz de salida hacia Factory I/O, se tendrá que recurrir a los flancos de bajada y subida de las variables `FIO_roscar` y `FIO_desroscar` (al ser variables *extended boolean*) y al sensor de movimiento de dicha parte del sistema para saber si el brazo ha finalizado su giro. Todo esto (como ya se ha comentado) quedará totalmente trasparente al alumno, ya que solo se pretende que el estudiante modelice el ciclo de funcionamiento mediante la correcta programación del código SFC.

A modo de pequeño resumen, en el Anexo II se encuentra una tabla resumen del conjunto de variables usadas en Unity relacionadas con los distintos elementos, variables que se han usado en los dos modelos para poder ver así un conjunto de todas las diferencias entre unas variables y otras.

Capítulo IV – Secciones del programa para el PLC

El software utilizado para programar el PLC se basa en la codificación de secciones. Estas secciones son independientes unas de otras a no ser que, gracias a una variable guarda en las opciones de configuración, se establezca una conexión entre ellas como se verá en el apartado 4.1. Esta variable guarda ha sido de gran utilidad para poder separar las interfaces empleadas en la estación real de las utilizadas para Factory I/O. Para tener una visión general (en la figura 14) se puede observar todas y cada una de ellas. El capítulo seguirá el orden que se muestra en la figura comenzando desde la sección de cambio de modo hasta llegar a las secciones de salidas.

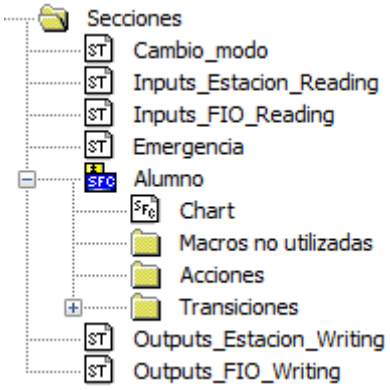


Figura 14: Secciones del programa

4.1 Sección de cambio de modalidad entre simulación y realidad

Lo que se ha pretendido hacer con esta sección ha sido básicamente buscar la comodidad del alumno para que, sin gran esfuerzo, pudiese pasar del modo "simulación" al modo "realidad". Con una variable booleana llamada `Modo_simulación` se activan o desactivan las variables guarda que permiten a las respectivas interfaces del modelo real o del modelo simulado ejecutarse (se han puesto como condiciones dentro de las opciones de sección de las interfaces de entradas y salidas, ver Anexo III). Esta variable depende de si los botones de selección de ambos modelos (ver Figura 15) están activados o no, es decir, en el caso de que cualquier botón esté activado (señal True o de valor 1) estaremos en modo simulación.

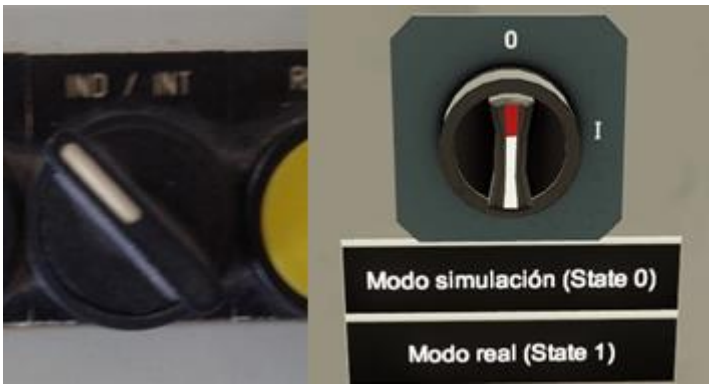


Figura 15: Selector real vs Selector virtual

Por el contrario, es necesario desactivar ambos botones para pasar al modelo real. Esto se ha hecho para que el modo simulación tenga prioridad ya que el alumno deberá empezar por este. El código desarrollado se encuentra en el Anexo III.

4.2 Secciones de entradas de ambos modelos

Como ya se ha comentado en el apartado 3.1, en la automatización es necesario saber en cada momento como se encuentra la instalación. Con la información recibida por parte de los sensores se dan valores a las variables comunes usadas en Unity (ver figura 17). Estas variables comunes son aquellas con las que trabajará el alumno. Son las variables que tienen ambos modelos en común y que sirven para dejar al estudiante ajeno a las interfaces creadas, dejando así oculto su contenido. En estas secciones se procede a la asignación de dichas variables (ver Anexo III).

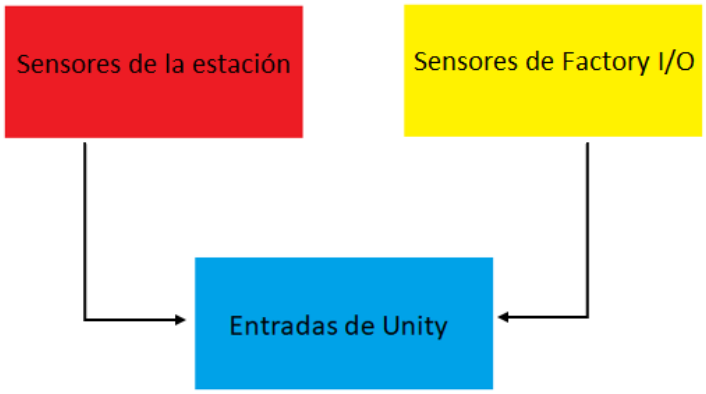


Figura 16: Lecturas de los sensores se transforman en entradas de Unity

Como se puede observar en la parte de entradas de la estación y de Factory I/O, cada sección establece una asignación distinta a la otra. Esto se debe a lo comentado anteriormente sobre los distintos objetos que poseen los dos modelos. En el caso de la estación real

simplemente se ha realizado una asignación entre variables, teniendo las variables de la estación vinculadas a sus respectivos espacios de memoria ya configurados en la parte del BUS Can. A este respecto, conviene resaltar que existen diferencias entre trabajar con esa configuración de entradas/salidas (en la que la información es depositada por el controlador del bus CAN directamente en el banco de memoria RAM del PLC). Por contraposición a la configuración de tarjetas de entradas/salidas pinchadas en el *rack* del PLC, en cuyo caso esas informaciones residen en las propias tarjetas y no en la RAM. En el primer caso no se calcula automáticamente la información de flancos, mientras que en el segundo sí está disponible dicha información (la puede proporcionar la tarjeta concreta si se le solicita).

Por otro lado, en la sección del modelo de simulación, los cambios que se han tenido que realizar implican a las variables del movimiento vertical de la pinza y a las de rotación (además de un pequeño cambio en la seta de emergencia por tener una lógica inversa a la de la estación real). Ambos movimientos se detectan gracias a un sensor de movimiento, lo cual significa que dicho sensor se pondrá a 1 cuando el brazo esté en movimiento y pasarán a valer cero cuando este se detenga.

Debido a esto, la lógica que se ha implementado para el movimiento vertical ha sido la de darle valor 1 a la variable `Pinza_Abajo` (de Unity) en el caso de que se haya finalizado el movimiento de dicha parte del brazo manipulador, y los sensores encargados de ver si la pinza iba hacia abajo se hayan activado. En caso contrario, si se produce una finalización de desplazamiento (un flanco de bajada en el sensor de movimiento) y los sensores encargados de verificar que la pinza iba hacia abajo no se hayan activado, se procederá a realizar un Reset de la variable `Pinza_abajo`. Además, se hará un Set de la variable `Pinza_arriba` indicando así que la pinza ha alcanzado su extremo superior.

Esta es la ecuación booleana (ver Anexo III) que se comentaba en el apartado 3.2.2 de descripción de sensores del modelo virtual. De ella, el alumno solo va a ver las variables de Unity, es decir, `Pinza_arriba` y `Pinza_Abajo`. Por lo que le será indiferente programar para el diseño real que para el diseño virtual. Como se ha dicho a lo largo de todo el documento, el alumno tiene que quedar al margen de lo programado en las interfaces.

Para el caso del movimiento rotacional la lógica implementada cambia un poco. En este caso, solo se dispone de un sensor de movimiento que actúa igual que el anterior pero no disponemos de sensores que indiquen en qué sentido está girando. Por tanto, para saber en qué posición se encuentra girada la pinza hay que recurrir a los actuadores. Un flanco de bajada en el sensor de movimiento junto con la activación del actuador de rotar a derechas indicará que

el brazo está girado a la derecha, en sentido de las agujas del reloj y viceversa para el otro sentido. En consecuencia, dependiendo del sentido verificado se procederá a activar la entrada `Brazo_Girado_derecha` o `Brazo_Girado_izquierda`.

4.3 Sección de tratamiento de emergencias

4.3.1 Programación simplificada del tratamiento de emergencias y rearmes

Esta sección será parte del código que desarrolle el alumno. A modo de ejemplo se ha modelado con una Red de Petri esta sección de emergencia para ver el funcionamiento que debería llevar a cabo el sistema en el caso de entrar en una parada de emergencia. Realmente el protocolo de emergencia que se debe realizar en estaciones de automatización posee una mayor complejidad. Al no ser un objetivo de este Trabajo Fin de Grado se ha realizado una pequeña simplificación de todo el proceso que se debería diseñar (para más información sobre cómo se debería proceder a desarrollar protocolos de emergencia se redactará al final de esta sección un párrafo a modo de pequeña introducción).

Al ser una pequeña simplificación, la implementación de la Red de Petri que modela el comportamiento deseado en caso de emergencia se ha realizado utilizando un código escrito en lenguaje texto estructurado en vez de SFC. Los principales lugares (variables de estados) que tenemos son: `En_emergencia`, `Listo_para_rearme` y `Rearmando`. Las transiciones estarán condicionadas por el valor de la seta de emergencia y los flancos detectados en la variable del botón de Rearme como se puede ver en la figura 17:

Figura 17: Modelo de la Red de Petri de la sección de emergencia

El estado de funcionamiento dependerá de cómo sea la secuencia que realice el alumno al trabajar con la máquina. Con esta sección se pretende parar dicho funcionamiento siempre que se active la seta de emergencia. Para poder rearmar la estación es necesario que se active la fuente de aire a presión ya que no se pueden mover las distintas partes del brazo sin dicha fuente. Con este propósito se ha añadido un lugar llamado “Listo para rearme”, en el cual han de llevarse a cabo las acciones necesarias que permitan llegar a la estación a su posición inicial (por ejemplo, liberando un posible atasco de los accionadores de la máquina). A continuación, el operario cuando haya realizado ya dichas tareas, activará el aire a presión, lo que se consigue levantando la seta de emergencia. Una vez esté todo listo para rearmar se mantendrá pulsado el botón de rearme y una vez se suelte, se realizarán todas las acciones oportunas. En este caso simplemente se necesita llevar el desplazador horizontal a su posición inicial ya que los demás actuadores, al ser cilindros simples, vuelven automáticamente a su estado inicial.

4.3.2 Guía GEMMA

Como se ha comentado y a modo de pequeña introducción se adjunta una imagen y una breve explicación sobre la guía GEMMA:

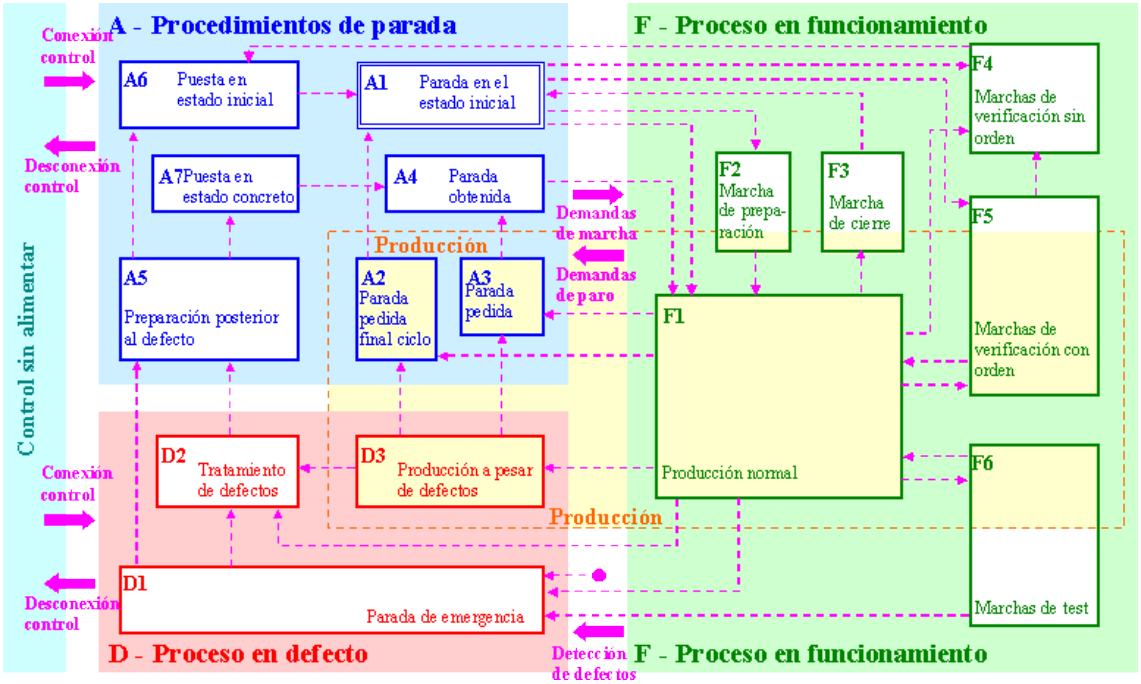


Figura 18: Esquema principal de la guía GEMMA[6]

La guía GEMMA, desarrollada por Adepa (Agence nationale pour le développement de la production automatisée, Agencia nacional francesa para el desarrollo de la producción automatizada), pretende mostrar las diversas situaciones que se dan en un proceso productivo (marchas, paradas, modos de funcionamiento…) para que a la hora de programar la automatización del proceso se tengan en cuenta. Por tanto esta guía es un soporte gráfico con el cual se pretende seguir el siguiente procedimiento en la implementación en automatismos:

- Estudiar los distintos estados anotando para cada recuadro la descripción correspondiente del proceso productivo con el que se esté trabajando y las diversas variaciones que este tenga. Los estados no utilizados se marcaran con una cruz.

- Observar entre qué estados será posible una evolución.

- Indicar cuales son las condiciones para que dicha evolución se lleve a cabo.

La guía también indica una serie de etapas a realizar para evaluar todas las posibles situaciones para así ayudar a tenerlas en cuenta en la programación SFC. En el ejemplo que hemos aplicado pasaríamos del funcionamiento en producción F1 al estado D1, la parada de emergencia, para pasar al estado A5 (que sería las operaciones realizadas por el operario de mantenimiento), A6 y A1 para volver al funcionamiento normal[5].

[6] Imagen recuperada de: https://recursos.citcea.upc.edu/grafcet/gemma/descrip.html

4.4 Sección de control secuencial/concurrente

Esta es la sección del Trabajo Fin de Grado que debe realizar el alumno, pero realmente es imposible trabajar con todas las demás partes sin tener un ejemplo sobre el que aplicarlo. Puesto que esta parte es muy flexible en lo que respecta a su orden de ejecución, se va a detallar sin hacer mucho hincapié en el ejemplo con el que se ha trabajado. Como ya se ha comentado, la idea principal era trabajar con el lenguaje SFC (*Sequential Function Chart*) que se caracteriza por ser un lenguaje de programación gráfico, similar a la lógica usada en las Redes de Petri. Ciertamente todo el diseño del proceso en el lenguaje SFC parte de un modelado con una Red de Petri, que luego se pasa al lenguaje detallando las acciones y condiciones que se deben realizar y cumplir respectivamente.

A continuación se adjunta en la figura 19 la Red de Petri que se ha modelado, en ella se indican los nombres de los lugares. Para no recargar la imagen no se han detallado las condiciones de las transiciones (acudir al Anexo III para poder encontrar dichas condiciones).

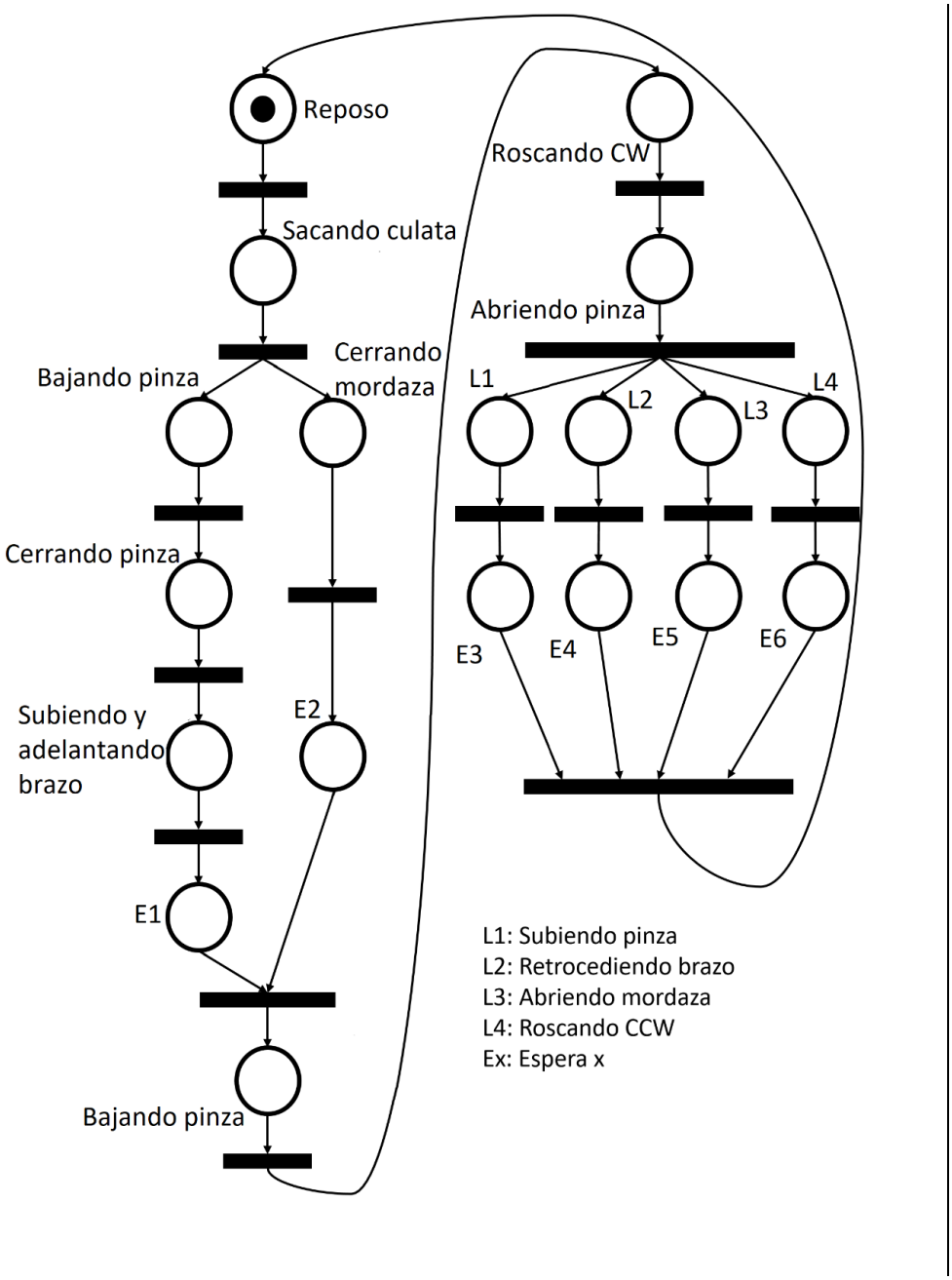


Figura 19: Red de Petri modelada del funcionamiento de la estación 3

Los elementos principales de los que consta este tipo de lenguaje son: las etapas o lugares (con acciones asociadas), las transiciones (con condiciones lógicas asociadas) y los arcos que unen los lugares y las transiciones como en las Redes de Petri[1]. En nuestro caso partimos de un lugar principal que sería el "Reposo". Este lugar, cuando está marcado, nos indica que la

máquina está preparada para iniciar su ciclo. Por lo cual, en este simple caso, activando la entrada de `Marcha` comienza el ciclo disparándose la transición de salida del lugar de "Reposo".

A continuación se empieza desde el dispensador (que sería la colocación de la tapa en su posición adecuada) sacando la culata [7] para empujarla hasta debajo del brazo, para posteriormente poder realizar todos los movimientos con el brazo. Se ha escogido realizar los movimientos en concurrencia en vez de en modo secuencial. Esta decisión se debe a que se asemeja más a la realidad, en las industrias de automatización las estaciones realizan varios movimientos simultáneamente para así reducir los tiempos de producción. En este caso, se tiene una primera concurrencia para cerrar la mordaza y hacer los primeros movimientos del brazo. Más tarde, se ejecutan en concurrencia todos los movimientos necesarios para devolver el brazo a su posición de reposo en espera para la próxima pieza.

En la figura 20, se muestra las acciones asociadas al lugar que modela la operación de roscar la tapa. Así se pueden ver de manera más clara qué acciones están asociadas a este lugar (se pueden ver todas las acciones asociadas de todos los lugares en el Anexo III):

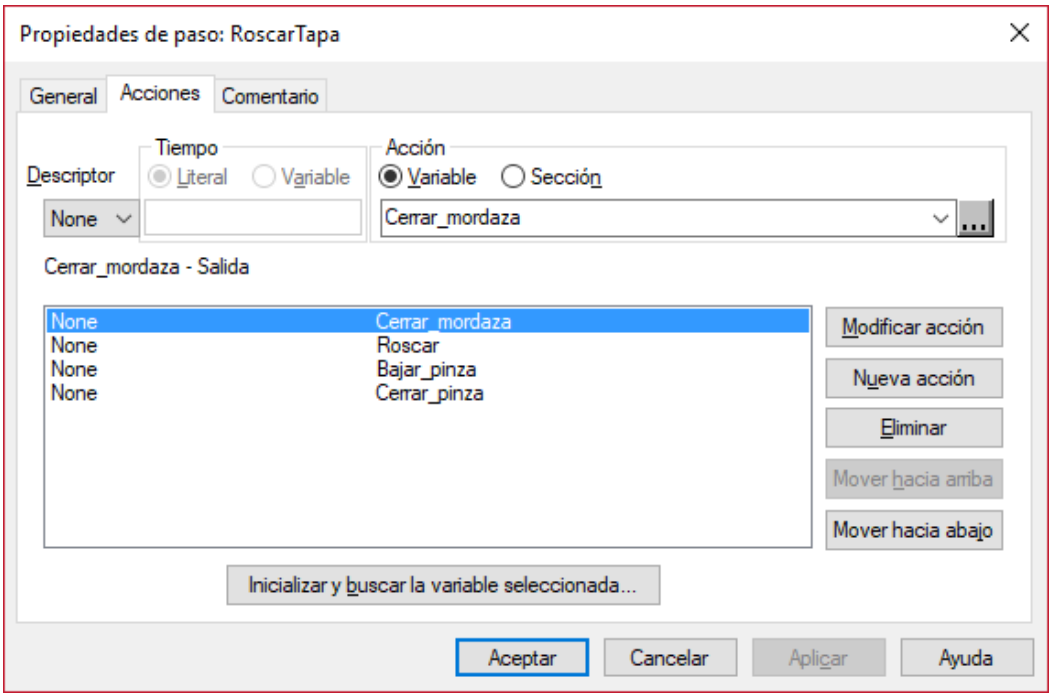


Figura 20: Ejemplo de lugar empleado en la sección SFC del código principal

[7] Con *Culata* se quiere referir a la tapa, por nomenclatura usada en el guion de prácticas del Grado, las variables referidas a la tapa irán con este nombre para así no cambiar la terminología ya usada.

4.5 Sección de salidas de ambos modelos

En el caso del modelo real, al igual que con las entradas, simplemente se ha realizado una asignación entre variables. Donde las variables de la estación tienen asignados sus respectivos espacios de memoria RAM (asignados a la hora de hacer la configuración del bus CAN). Y como anteriormente ha sucedido, para el modelo simulado han sido necesarias ciertas adaptaciones para que el código funcionase del mismo modo. En este caso los actuadores diferentes eran el deplazador del eje X (avanzar y retroceder el brazo) y el actuador encargado de rotar la pinza.

El problema principal del actuador del brazo reside en que, al ser un cilindro de simple efecto, el propio brazo no se mantenía en su posición si se ponían a cero las dos salidas. Cosa que en la estación real cuando se anulan las salidas (entradas de aire al cilindro) el brazo se mantiene donde se halle ya que no tiene la fuente que provoca su movimiento. Por ello, se ha considerado que en la simulación, cuando el brazo llega a su posición extendida, la salida dentro del Factory I/O asignada a ese actuador se mantendrá a 1 hasta que el alumno active la salida `Retroceder_brazo`. Esto se puede ver mejor en el (Anexo III). Para que así el alumno en el SFC no tenga por qué mantener esa salida activada como ocurre en el caso de la estación real.

Se podría decir que los cilindros de doble efecto tienen un comportamiento impulsional. Basta con dar un impulso en una de las salidas del PLC para abrir la electroválvula y hacer que el aire entre en un sentido o en otro, para conseguir colocar la máquina en la posición que se desee. Luego, al no tener un muelle en la electroválvula que la devuelva a su posición de cierre, no es necesario mantener esa salida activa, ya que una vez el cilindro alcance su posición, se quedará fijo independientemente de si el estado de la salida del PLC es cero.

El actuador de rotación de la estación real se trata de un cilindro de simple efecto mientras que, en la simulación, se trata de dos actuadores diferentes que controlan la rotación en un sentido y en otro. Al desactivar uno de los dos, la pinza no retorna a su posición de partida, por lo que para hacer el “contragiro” hay que desactivar el actuador que se haya activado y activar el que controla el giro opuesto.

Por tanto, cuando el alumno active la salida `Roscar` de Unity, en Factory I/O se activará el actuador que controla el giro en sentido de las agujas del reloj `FIO_roscar`. Cuando el alumno desactive la salida `Roscar` entonces con la detección de un flanco de bajada en la propia variable ebool `FIO_roscar`, se activará la salida `FIO_roscar_izquierda` que hará que la pinza se coloque en su posición inicial. Ejecutándose así como un cilindro de simple efecto (la estructura del código en la que se desarrolla lo explicado está en el Anexo III).

Con todo lo explicado hasta este capítulo, queda demostrado el cumplimiento del objetivo que se ha comentado durante todo el documento: lograr ocultar al estudiante las diferencias entre el trabajo con la máquina real y la virtual. El estudiante diseñará su solución pensando en la estación real, mientras que los cambios realizados para asemejar ambos modelos son totalmente transparentes gracias a toda la labor realizada en este Trabajo Fin de Grado.

Capítulo V – Validación del ejemplo desarrollado

Con la intención de que quede plasmado todo lo anteriormente explicado, en este apartado se adjuntan un par de enlaces de dos vídeos de YouTube. En estos vídeos, queda reflejado el funcionamiento de ambos modelos con el propósito de que el lector comprenda mejor los conceptos detallados y sirva entonces como consolidación final del trabajo.

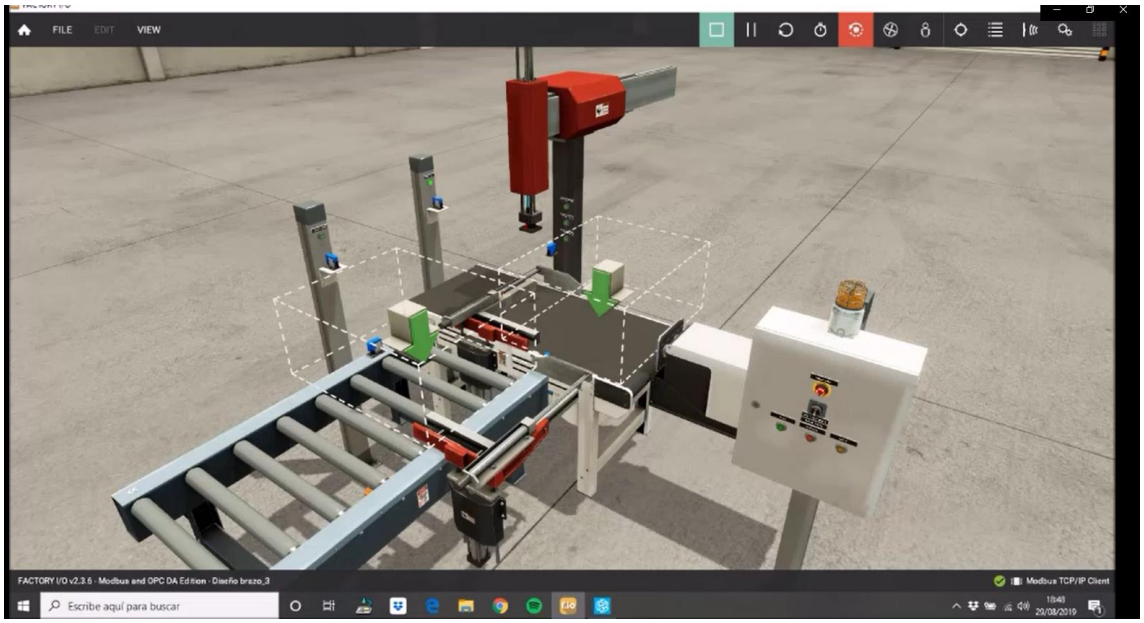


Figura 21: Modelo virtual; enlace: https://youtu.be/K3mC_zjxhWU

En caso de una mala visualización, este vídeo se encuentra también en el repositorio[8] que se ha creado para que los lectores dispongan del material obtenido como resultado de este proyecto.

[8] Enlace: https://drive.google.com/open?id=1rf5LQpIIBrgB4IBphUxMHb2GBehfFV4V

El siguiente enlace pertenece a un vídeo que realizó el tutor José Ramón Asensio Diago sobre el funcionamiento deseado que tiene que seguir la máquina real.

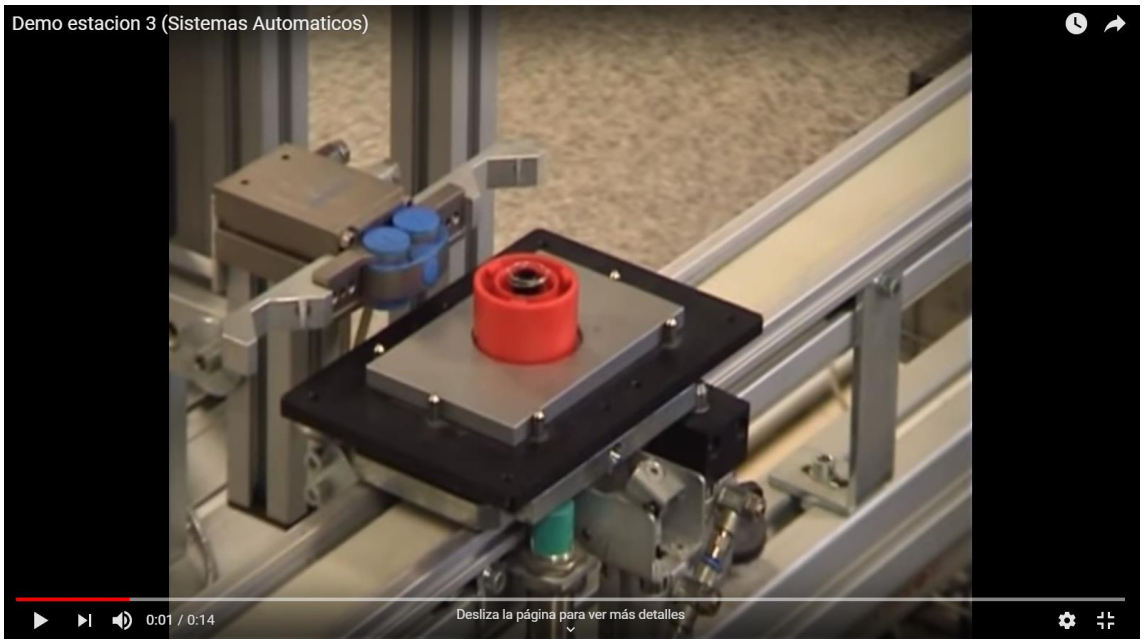


Figura 22: Modelo estación real; enlace: https://youtu.be/hBhjv3FKD5k

Con estos dos vídeos, queda verificado que los dos modelos (a pesar de sus diferencias) se comportan de manera idéntica cuando se les aplica el mismo sistema de control.

Capítulo VI – Conclusiones y resultados

Tras haber expuesto los objetivos de este proyecto y haber explicado el trabajo realizado, se puede decir que se han obtenido los resultados esperados. El alcance del trabajo ha sido el de verificar que realmente se podría implementar este nuevo software en el laboratorio y el de conseguir obtener un modelo sobre el que poder trabajar. Esto no ha sido nada sencillo puesto que, como ya se ha mencionado anteriormente, no se disponía de los objetos necesarios para construir fácilmente el modelo virtual. O en el caso de tenerlos, algunos no se comportaban de idéntica manera. Por ejemplo, algunos de los mayores problemas encontrados eran: la rotación del objeto como ya se ha mencionado en el apartado 3.2.3 y el posicionamiento de la tapa debajo del brazo también mencionado en este apartado.

Otro problema que se puede dar es que el brazo manipulador falle en la colocación de la tapa sobre la base. Esto puede deberse a variables de la propia simulación, ya que a veces cuando la tapa choca con la primera mordaza que hace de muro, realiza un pequeño retroceso que puede perjudicar en el correcto posicionamiento final de esta. Por lo que en estos casos se deberá detener la simulación o realizar un rearme en caso de haber realizado la sección de emergencia. Se espera que de aquí a unos años Real Games desarrolle una mayor gama de objetos con los que poder trabajar y entonces esta parte del modelo virtual se pueda mejorar para asemejarlo mejor a la realidad.

Un error común en la configuración de la simulación se da en el *offset* que hay que poner en la sección de variables de entrada y salida. Este *offset* sirve para no superponer valores en los bits de memoria, hay que tenerlo en cuenta para no dar sin querer valores a las entradas al modificar las salidas.

Como resultados obtenidos, está la plantilla que usará el estudiante para programar en Unity junto con el archivo de Factory I/O en el que se encuentra el modelo virtual (para poder acceder a ellos se adjunta un repositorio). También en el Anexo I, se dispone del documento que utilizarán los estudiantes como guion de prácticas (este documento también se encontrará por separado en el repositorio). Todos estos documentos serán entregados a los tutores para que en un futuro hagan uso de ellos en los diferentes Grados de la Escuela. Las futuras experiencias de la impartición de prácticas con este sistema en la Escuela dirán si es beneficiosa para los

alumnos. En cualquier caso, con este nuevo modelo se consiguen acortar los tiempos de espera para la verificación del código del sistema de control como se pretendía desde un principio. Se puede concluir que en este sentido el trabajo ha sido un éxito.

El estudio realizado para implementar el protocolo OPC mostró que el proceso para conseguir la comunicación para la recogida y emisión de datos entre el cliente y el servidor es arduo de conseguir; puesto que es necesaria la creación y programación de dicho servidor al ser en este caso los clientes el autómata programable y Factory I/O. Por tanto como conclusión del estudio sobre la inclusión de OPC en el trabajo aquí desarrollado, puede decirse que si bien es factible, dada su complejidad y envergadura, bien merece un correcto desarrollo en otro TFG, para el cual el aquí presentado ha sentado las bases.

BIBLIOGRAFÍA

[1] Lenguaje Sequential Function Chart (SFC) [Internet]. Wikipedia. [Última modificación 5 de julio 2019]. Disponible en: https://es.wikipedia.org/wiki/Sequential_Function_Chart

[2] Historia sobre los PLC's [Internet]. PLC mentor. 2019. Disponible en: https://www.plcmentor.com/Articles/Newsletters/Programmable-Logic-Controller-PLC-History

[3] Historia sobre los PLC's [Internet]. Library.AutomationDirect. [citado 16 de agosto de 2019]. Recuperado a partir de: https://library.automationdirect.com/history-of-the-plc/

[4] Historia sobre Factory I/O. Real Games [Internet]. 2019. Disponible en: https://realgames.co/

[5] Víctor Suárez. Universidad de Oviedo [Internet]. Guía GEMMA. 2019 [citado 16 de agosto 2019]. Disponible en: http://isa.uniovi.es/~vsuarez/Download/GemmaTelemecanique.PDF

[6] Manual de Unity. Schneider Electric [Internet]. 2019. Disponible en: https://www.se.com/es/es/download/document/33003538K01000/

[7] Ejemplo empleado para el aprendizaje de Factory I/O. Factory I/O, Real Games [Internet]. 2019. Disponible en: https://factoryio.com/docs/tutorials/schneider/unity-pro-modbus/

[8] Darek Kominek. Canada. 2009. Infoplc [Internet]. Protocolo OPC. Disponible en: http://www.infoplc.net/files/documentacion/comunicaciones/infoplc_net_guia_para_entender_la_tecnologia_opc.pdf

[9] Guion de prácticas. Ingeniería de Control, Grado en Tecnologías Industriales. 2017-2018. Disponible en el ADD para los estudiantes de la asignatura.

[10] Manual de Factory I/O. Factory I/O, Real Games [Internet]. 2019. Disponible en: https://factoryio.com/docs/manual/

Anexos

Anexo I: Ejemplo de guion de la práctica 4 con simulación

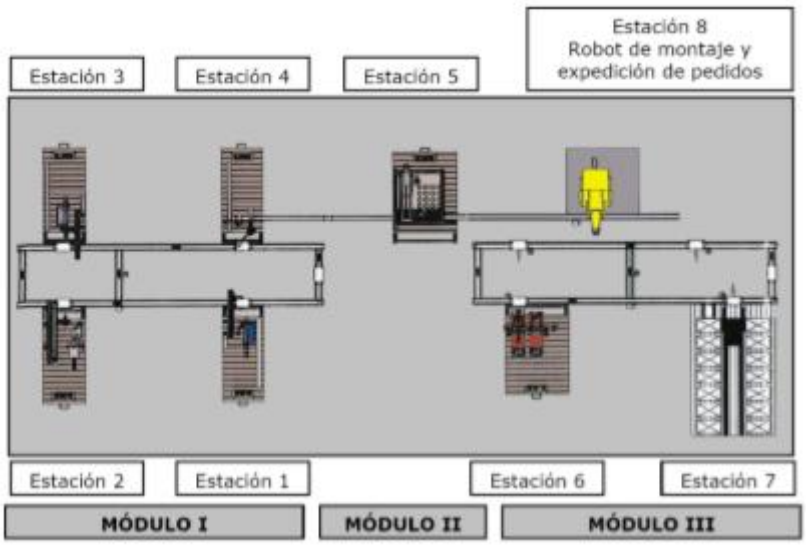
Práctica: Control de un sistema de eventos discretos

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

El objetivo de la práctica es modelar el control de un automatismo sencillo mediante un autómata finito determinista. Implementar dicho control en un autómata programable utilizando el lenguaje texto estructurado (ST) y Sequential Function Charts (SFC).

1.- Descripción general de la célula de fabricación.

La estación de la célula de fabricación industrial se encuentra situada en el laboratorio L0.06 del edificio Ada Byron. El objetivo de la célula de fabricación es la producción, almacenamiento y expedición de cilindros neumáticos.



La célula se divide en tres módulos.

En el módulo I tiene lugar la selección del tipo de cilindro, la colocación en cilindro de émbolo y muelle, el roscado de la tapa y la verificación final de la correcta colocación de los elementos.

En el módulo II hay un almacén intermedio donde se guardan provisionalmente los cilindros individuales en espera de ser agrupados en pedidos conjuntos de 3 cilindros.

En el módulo III se selecciona el tipo de base para cada pedido conjunto, se colocan los cilindros provenientes del almacén intermedio en cada base mediante un brazo manipulador, y se almacenan los pedidos ya completos en el almacén final, listos para su expedición fuera de la célula utilizando de nuevo el brazo.

2.- Descripción de la estación 3

En esta práctica se trabajará con la estación 3 de la célula de fabricación flexible, es la encargada de colocar una tapa sobre una pieza fabricada por otras estaciones en dicha célula (la tapa azul de la figura adjunta).

Para realizar el control, el autómata necesita recabar información del proceso (usando sensores) y actuar sobre él (mediante actuadores). Todos los actuadores de este automatismo son cilindros neumáticos accionados por electroválvulas (la puesta a 1 de una salida del autómata abre la electroválvula, introduciendo aire a presión por una entrada del cilindro neumático; la puesta a 0 de dicha salida cierra la electroválvula). Dichos cilindros son de dos tipos: de doble efecto y de simple efecto. Los primeros llevan una entrada de aire a presión en cada extremo, por lo que necesitan dos salidas del autómata para moverlo (una para cada sentido). Los de simple efecto llevan un muelle interno que los devuelve a su posición de reposo al dejar de introducir aire a presión, por lo que su movimiento se controla con una única salida del autómata.

Para recoger información del automatismo existen sensores que determinan la posición (extendido o recogido) de cada cilindro neumático en la propia máquina: cada vez que un cilindro llega a uno de sus extremos, se activará una entrada en el autómata. Además de los sensores ya mencionados, otras entradas son las que provienen de la botonera (ver figura adjunta). Incorporan una seta de emergencia (que se enclava y permanece así hasta que se realiza un pequeño giro para desenclavarla), dos interruptores marcados como IND-INT y MAN-AUTO, y dos pulsadores MARCHA y RESET. En esta práctica usarás el pulsador MARCHA, para indicar el inicio de una nueva secuencia de operación de la máquina, el pulsador IND-INT que deberá estar en la posición IND para trabajar con la simulación y en posición INT para trabajar con la estación real, y la seta de emergencia si necesitas detener la máquina. En esta estación la

seta de emergencia no corta la alimentación a las electroválvulas por lo que, para que su pulsación sea efectiva, deberá ser correctamente programada en el autómata. En caso de diseñarse una sección de emergencia el pulsador RESET servirá como botón de rearme.

3.- Descripción de la estación en el simulador

El diseño sobre el que se ha basado la estación 3, consta de casi todos los elementos que posee la estación real, y en caso de no haber como sucede con ciertos sensores, las interfaces se han diseñado de manera que el modo de funcionamiento sea idéntico en ambos modelos.

En este caso se va a trabajar con piezas cuadradas (Factory I/O no posee todavía piezas circulares en su librería), la única diferencia entonces será que la mordaza realmente no sujeta la pieza para facilitar así su giro. Aun así deberás trabajar con ella tal y como se muestra en el funcionamiento deseado (ver más adelante) ya que el código que diseñes con la simulación lo usarás directamente dentro de la estación real. La mordaza deberá entonces estar cerrada cuando deba, simulando así que sujeta la pieza (se tendrá muy en cuenta este hecho al evaluar).

El funcionamiento de la simulación será el mismo a excepción de una pequeña variación. Deberás colocar piezas en la estación cada vez que quieras ejecutar un ciclo (pulsador rojo `Poner_piezas`). Para ello se ha creado una botonera como en la realidad (ver figura adjunta), su funcionamiento es idéntico a la botonera real.

4.- Funcionamiento deseado

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 tapa del alimentador por gravedad, para lo cual debe extenderse un cilindro de simple efecto existente en su base (salida Sacar_culata); se entenderá que siempre hay al menos una tapa disponible en el alimentador; a pesar de existir un detector óptico (Culata_detectada) que indica la presencia de tapa, ha de mantenerse la acción durante 1 segundo más para llevar la tapa hasta su posición correcta). **Acordarse en la simulación de poner piezas antes de empezar.**

2) bajar el cabezal con la pinza (salida Bajar_pinza)

3) cerrar la pinza (salida Cerrar_pinza); esperar 1 segundo para asegurar la correcta sujeción de la tapa

4) subir el cabezal con la pinza (no activar la salida Bajar_pinza) y llevarlo hacia la pieza (salida Avanzar_brazo); estos movimientos se pueden hacer simultáneamente, ahorrando tiempo respecto a una versión puramente secuencial)

5) sujetar la pieza con la mordaza (salida Cerrar_mordaza); realmente esto puede hacerse en cualquier momento desde que se haya pulsado Marcha

6) bajar el cabezal

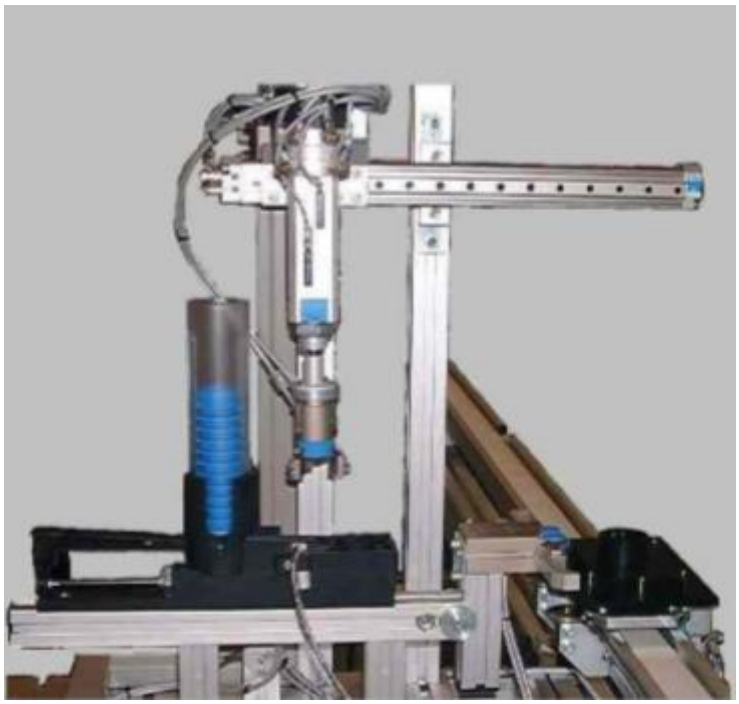
7) roscar la tapa sobre la pieza (salida Roscar), lo que realiza un giro en el sentido de las agujas del reloj (clockwise)

8) abrir pinza (no activar la salida Cerrar_pinza); esperar 0.1 segundos para asegurar que está completamente abierta (no existe detector para ello)

9) subir el cabezal, girar la pinza en sentido contrario a las agujas del reloj (counter-clockwise) y llevar el cabezal hacia el alimentador por gravedad; tras esto, la máquina estará preparada para un nuevo ciclo.

En la figura adjunta se muestra la máquina en su posición de reposo, con el empujador de tapas sin extender, y el brazo con el cabezal atrás (es decir, listo para coger la tapa que ha de

montar) y arriba. En la parte derecha, abajo, se aprecia un palet negro con una pieza (también negra) sobre la que hay que montar la tapa.



En el repositorio de la asignatura en moodle existe un vídeo correspondiente al funcionamiento deseado.

5.- Descripción de las entradas y salidas

A continuación se tabulan las entradas y salidas de este automatismo. En este caso, al utilizar una red CANOpen, no son entradas/salidas directas sino que están mapeadas en palabras de la memoria del autómata. En el software a utilizar los identificadores de las variables se llaman etiquetas. Nótese que el símbolo Reset no puede usarse, al coincidir con una palabra reservada del lenguaje de programación, y por ello se ha utilizado el símbolo Rearme. Al igual que en cualquier otro lenguaje de programación, no se deben utilizar acentos, espacios, ni eñes en los identificadores.

ENTRADAS		
Objeto	**Etiqueta**	**Condición de puesta a 1**
%MW35.0	Brazo_atras	El brazo está sobre el alimentador de tapas
%MW35.1	Brazo_adelante	El brazo está sobre el palet con la pieza para montar la tapa
%MW35.2	Brazo_Girado_izquierda	El cabezal está girado a *counter-clockwise*
%MW35.3	Brazo_Girado_derecha	El cabezal está girado a *clockwise*

%MW35.4	Pinza_arriba	El cabezal con la pinza está arriba
%MW35.5	Pinza_abajo	El cabezal con la pinza está abajo
%MW37.0	Culata_detectada	El detector óptico detecta la existencia de tapa
%MW37.1	Seta_Emergencia	Seta de emergencia enclavada
%MW37.2	Marcha	Pulsador MARCHA apretado
%MW37.3	Ind_int	En posición IND
%MW37.4	Rearme	Pulsador RESET apretado
%MW37.5	Manual_automatico	En posición AUTO

Respecto a las salidas, sólo las número 0 y 1 corresponden a cilindros neumáticos de doble efecto; esto hace que si ambas están activas, el cilindro neumático no se moverá. Para el resto, cuando la salida se pone a 0 se realiza el movimiento opuesto (por la acción del muelle interno).

SALIDAS		
Objeto	**Símbolo**	**Acción ejecutada si su valor es 1**
%MW134.0	Avanzar_brazo	Llevar el brazo hacia el palet
%MW134.1	Retroceder_brazo	Llevar el brazo hacia el alimentador de tapas
%MW134.2	Roscar	Girar el cabezal *clockwise*
%MW134.3	Bajar_pinza	Bajar la pinza
%MW134.4	Sacar_culata	Extender el empujador del alimentador de tapas
%MW134.5	Cerrar_mordaza	Cierra la mordaza de sujeción de la pieza
%MW135.0	Cerrar_pinza	Cierra la pinza que sujeta la tapa

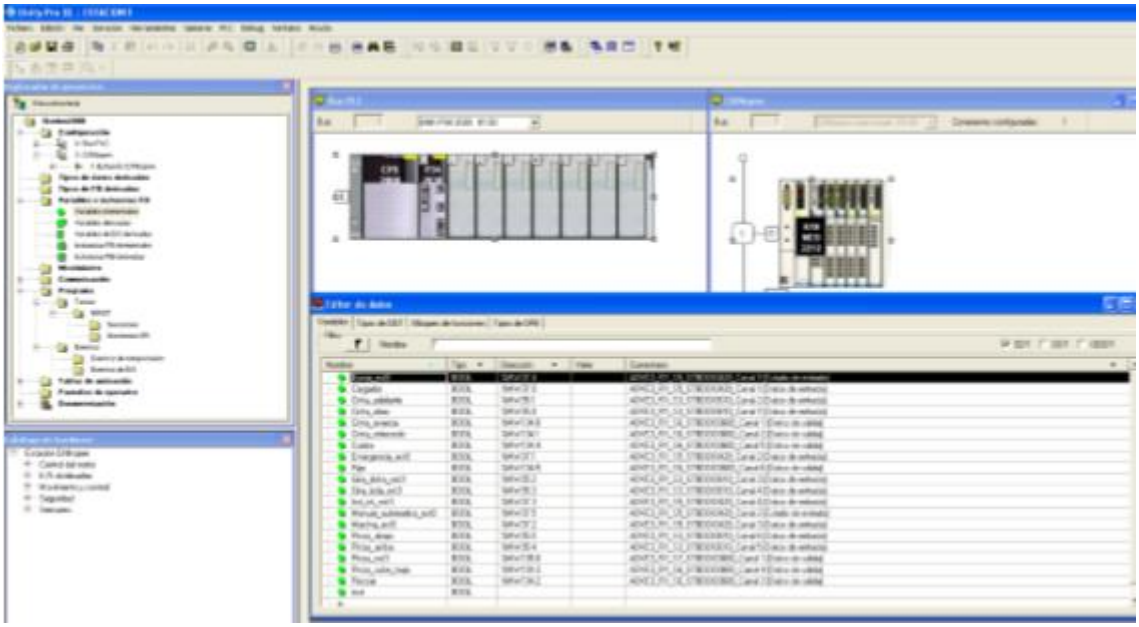
Estudio previo

Es necesario que el alumno acuda a la sesión con el problema estudiado y una solución consistente en el modelo el sistema y su control (asignando las entradas a las transiciones y las salidas correspondiente a los estados) mediante autómata de tipo Moore.

Sesión de laboratorio

Se realizará la implementación del grafo de estados elaborado 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: estacion3.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 correcta de hardware, 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 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.

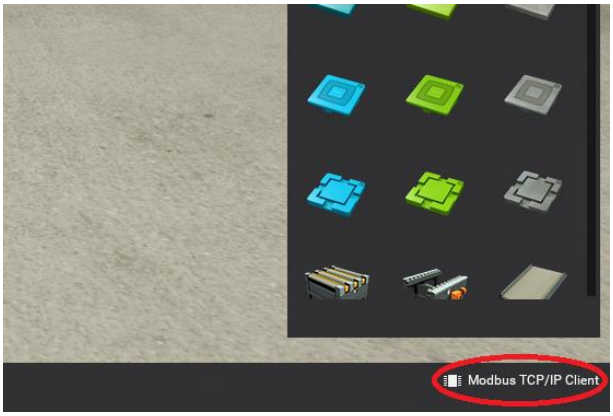
Existen dos tipos de variables lógicas:

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.

Análisis, transferencia y ejecución del código

Si se desea comprobar el correcto comportamiento del código mediante simulación se elegirá el menú PLC/Modalidad de estándar y se abrirá el modelo de la estación dentro de Factory I/O (dentro de Scenes/My Scenes). Se deberá establecer la dirección física del autómata mediante PLC/Establecer dirección, del mismo modo haremos en Factory. Dentro del menú Modbus TCP/IP Client, situado en la esquina inferior derecha, accedemos a Configuración y en ModbusTCP_IPClient/Server/Host establecemos la dirección del autómata que vamos a emplear.



Si por el contrario se desea ejecutar el código en la estación real se elegirá PLC/Modalidad estándar, se introducirá la dirección del autómata de la célula.

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. 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.

Estos pasos servirán para ejecutar el código sobre la estación real. Para la simulación hay que añadir los que se explicarán a continuación. Una vez abierto el fichero con el diseño de la estación y ya establecida la dirección física del autómata, se debe conectar a este. Dentro del menú Modbus TCP/IP Client hay que darle a la opción Connect. Una vez se haya establecido la conexión aparecerá un tick verde que nos indica que la conexión con la dirección establecida es correcta. Para proceder a ejecutar solo hará falta haber realizado los pasos anteriores en Unity y pulsar la opción Run en Factory situada en el menú de la barra desplegable superior.

Notas:

- Dentro de Factory I/O, en caso de que alguna etiqueta de los actuadores este en azul significa que ese actuador está forzado a un valor. Para liberarlo habrá que entrar en la etiqueta del actuador dándole doble click en el nombre del actuador y hacer click en FORCED, quedándose así el nombre en blanco y la etiqueta en RELEASED.

- Se han dado casos de que el brazo manipulador falle a la hora de colocar la pieza, esto se debe a variaciones de la simulación, en caso de ocurrir hacer un Rearme si se ha desarrollado el código de emergencia o detener la simulación en Factory y volver a ejecutarla, habrá que inicializar en dicho caso las variables dentro de Unity.

- En el menú Modbus TCP/IP Client dentro de la parte de configuración/I/O Points hay que comprobar que el *offset* de las salidas digitales coincida con el número de entradas, es decir 12, para que así no se superpongan los espacios de memoria.

Anexo II: Tabla resumen con actuadores y sensores de los dos modelos según su función

Para evitar que este apartado sea muy farragoso, los nombres que se han dado a las variables dentro de los dos modelos han sido el mismo que el usado en Unity pero con "Estacion" delante del nombre para el modelo real y "FIO" para el modelo virtual. A modo de ejemplo:

Variable de Unity: `Brazo_atras` → Variable estación: `Estacion_brazo_atras` → Variable Factory: `FIO_brazo_atras`

Variable en Unity	**Modelo de la simulación**	**Estación del laboratorio**
`Brazo_atras`	1 sensor difusor	1 sensor inductor
`Brazo_adelante`	1 sensor difusor	1 sensor inductor
`Brazo_girado_izquierda`	1 sensor de movimiento	1 sensor inductor
`Brazo_girado_derecha`	1 sensor de movimiento (mismo que el anterior)	1 sensor inductor
`Pinza_arriba`	1 sensor de movimiento	1 sensor inductor
`Pinza_abajo`	2 sensores difusores	1 sensor inductor
`Culata_detectada`	1 sensor difusor	1 sensor óptico
`Seta_emergencia`	1 seta de emergencia	1 seta de emergencia
`Marcha`	1 pulsador	1 pulsador
`Rearme`	1 pulsador	1 pulsador
`Modo_simulacion` [9]	1 botón selector	1 botón selector
`Avanzar_brazo`	1 cilindro de simple efecto	1 cilindro de doble efecto
`Retroceder_brazo`	1 cilindro de simple efecto (mismo que el anterior)	1 cilindro de doble efecto (mismo que el anterior)
`Roscar`	2 cilindros de doble efecto[10]	1 cilindro de simple efecto
`Bajar_pinza`	1 cilindro de simple efecto	1 cilindro de simple efecto
`Sacar_culata`	1 cilindro de simple efecto	1 cilindro de simple efecto
`Cerrar_mordaza`	1 cilindro de simple efecto	1 cilindro de simple efecto
`Cerrar_pinza`	1 sistema de succión	1 cilindro de simple efecto

[9] Para esta variable los nombres respectivamente son: Modo_simulación para Factory I/O e Ind_int para la estación.

[10] Al segundo actuador de Factory I/O se le ha llamado desroscar para indicar que el actuador realiza el giro opuesto.

Anexo III: Documentación del programa desarrollado en Unity

DOCUMENTACIÓN TÉCNICA

TrabajoFinGradoFernandoGrima

Proyecto	Trabajo_Fin_Grado_Fernando_Grima_Montesa
Diseñador	
Aplicación	UNION_DE_PLANTILLA_CON_TRABAJO.stu
Versión del software	Unity Pro XLS V13.0
Fecha de creación	06/07/2019 16:08:54
Fecha de la última modificación	14/08/2019 11:24:24
PLC de destino	BMX P34 20302 02.30CPU 340-20 Ethernet CANopen2

Contenido

Total: 26 páginas

Variables e instancias FB

BOOL

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
Avanzar_Brazo	NO		Salida		7	NO
Bajar_pinza	NO		Salida		9	NO
Borrar_charts	NO		Variable interfaz de emergencia		3	NO
Brazo_adelante	NO		Entrada		9	NO
Brazo_atras	NO		Entrada		7	NO
Brazo_Girado_Derecha	NO		Entrada		7	NO
Brazo_Girado_Izquierda	NO		Entrada		5	NO
Cerrar_mordaza	NO		Salida		9	NO
Cerrar_pinza	NO		Salida		9	NO
Culata_detectada	NO		Entrada		5	NO
En_emergencia	NO		Variable interfaz de emergencia		3	NO
Estacion_Avanzar_brazo	NO	%MW134.0	Actuador		1	NO
Estacion_Bajar_pinza	NO	%MW134.3	Actuador		1	NO
Estacion_Brazo_adelante	NO	%MW35.1	Sensor		1	NO
Estacion_Brazo_atras	NO	%MW35.0	Sensor		1	NO
Estacion_Brazo_girado_derecha	NO	%MW35.3	Sensor		1	NO
Estacion_Brazo_Girado_izquierda	NO	%MW35.2	Sensor		1	NO
Estacion_Cerrar_mordaza	NO	%MW134.5	Actuador		1	NO
Estacion_Cerrar_pinza	NO	%MW135.0	Actuador		1	NO
Estacion_Culata_detectada	NO	%MW37.0	Sensor		1	NO
Estacion_Marcha	NO	%MW37.2	Botón		1	NO
Estacion_Pinza_abajo	NO	%MW35.5	Sensor		1	NO
Estacion_Pinza_arriba	NO	%MW35.4	Sensor		1	NO
Estacion_Rearme	NO	%MW37.4	Botón		1	NO
Estacion_Retroceder_brazo	NO	%MW134.1	Actuador		1	NO
Estacion_Roscar	NO	%MW134.2	Actuador		1	NO
Estacion_Sacar_culata	NO	%MW134.4	Actuador		1	NO
Estacion_seta_emergencia	NO	%MW37.1	Botón		1	NO
Guarda_inputs_est	NO		Condición de interfaz		3	NO
Guarda_inputs_FIO	NO		Condición de interfaz		3	NO
Ind_int	NO	%MW37.3	Botón para cambio de modo		2	NO
Listo_para_Rearme	NO		Variable interfaz de emergencia		4	NO
Marcha	NO		Entrada		5	NO
Modo_simulacion	NO		Sección modo		2	NO
Obten_salidas_est	NO		Condición de interfaz		3	NO
Obten_salidas_FIO	NO		Condición de interfaz		3	NO
Pinza_abajo	NO		Entrada		10	NO
Pinza_arriba	NO		Entrada		7	NO
Rearmando	NO		Variable interfaz de emergencia		3	NO
Rearme	NO		Entrada		5	NO
Resultado_clear	NO		Variable interfaz de emergencia		1	NO

Variables e instancias FB

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
Resultado_init	NO		Variable interfaz de emergencia		1	NO
Retroceder_Brazo	NO		Salida		8	NO
Roscar	NO		Salida		6	NO
Sacar_culata	NO		Salida		5	NO
Seta_Emergencia	NO		Entrada		6	NO
T1	NO				2	NO
T2	NO				2	NO
T3	NO				2	NO
T4	NO				2	NO
T5	NO				2	NO
T6	NO				2	NO

EBOOL

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
Copia_Rearme	NO		Variable interfaz de emergencia		3	NO
FIO_avanzar_retroceder_brazo	NO	%M12	Actuador		3	NO
FIO_bajar_pinza	NO	%M13	Actuador		1	NO
FIO_boton_simulacion	NO	%M10	Sensor/cambio de modo		1	NO
FIO_brazo_adelante	NO	%M4	Sensor		1	NO
FIO_brazo_atras	NO	%M3	Sensor		1	NO
FIO_brazo_girando	NO	%M9	Sensor		2	NO
FIO_cerrar_mordaza	NO	%M14	Actuador		1	NO
FIO_cerrar_pinza	NO	%M15	Actuador		1	NO
FIO_culata_detectada	NO	%M8	Sensor		1	NO
FIO_dispensador_bases	NO	%M28	Emisor piezas para un segundo ciclo		2	NO
FIO_dispensador_tapas	NO	%M27	Emisor piezas para un segundo ciclo		2	NO
FIO_emergency_stop	NO	%M0	Botón		1	NO
FIO_indicador_brazo_adelante	NO	%M19	Luz_sensor		1	NO
FIO_indicador_brazo_atras	NO	%M20	Luz_sensor		1	NO
FIO_indicador_brazo_girado	NO	%M21	Luz_sensor		1	NO
FIO_indicador_culata_detectada	NO	%M22	Luz_sensor		1	NO
FIO_indicador_marcha	NO	%M23	Luz_botón		1	NO
FIO_indicador_pinza_abajo	NO	%M24	Luz_sensor		1	NO
FIO_indicador_pinza_arriba	NO	%M25	Luz_sensor		1	NO
FIO_indicador_poner_piezas	NO	%M26	Luz_botón		1	NO
FIO_indicador_seta_emergencia	NO	%M29	Luz_botón		1	NO
FIO_marcha	NO	%M2	Botón		1	NO
FIO_pinza_abajo	NO	%M6	Sensor		2	NO
FIO_pinza_abajo2	NO	%M7	Sensor		2	NO
FIO_pinza_arriba	NO	%M5	Sensor		2	NO
FIO_poner_piezas	NO	%M11	Botón para un segundo ciclo		2	NO
FIO_rearme	NO	%M1	Botón		1	NO
FIO_roscar	NO	%M17	Actuador		4	NO

Variables e instancias FB

Nombre	Const	Dirección	Comentario	Valor	Utilizado	DG
FIO_roscar_izquierda	NO	%M16	Actuador		3	NO
FIO_sacar_culata	NO	%M18	Actuador		1	NO

SFCCHART_STATE

Nombre	Const	Dirección	Comentario	Utilizado
Alumno	NO			2

SFCSTEP_STATE

Nombre	Const	Dirección	Comentario	Utilizado
AbrirMordaza	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
AbrirPinza	NO			2
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
BajarCabezal	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
BajarCabezal2	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
CerrarMordaza	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
CerrarPinza	NO			2
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Desroscar	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Empujador	NO			2
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Espera1	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Espera2	NO			1
t	NO			
x	NO			
tminErr	NO			

Variables e instancias FB

Nombre	Const	Dirección	Comentario	Utilizado
tmaxErr	NO			
Espera3	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Espera4	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Espera5	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Espera6	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
Reposo	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
RetrocederBrazo	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
RoscarTapa	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
SubirCabezal	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			
SubirCabezal2	NO			1
t	NO			
x	NO			
tminErr	NO			
tmaxErr	NO			

Estructura de la aplicación

VISTA ESTRUCTURAL

SECCIÓN	CONDICIÓN DE VALIDACIÓN	COMENTARIO DE SECCIÓN	MÓDULO	LENGUAJE
Cambio_modo				ST
Inputs_Estacion_Reading	Guarda_inputs_est			ST
Inputs_FIO_Reading	Guarda_inputs_FIO			ST
Emergencia				ST
Alumno				SFC
Chart				SFC
T2				ST
T1				ST
T4				ST
T3				ST
T5				ST
T6				ST
Outputs_Estacion_Writing	Obten_salidas_est			ST
Outputs_FIO_Writing	Obten_salidas_FIO			ST

CALL TREE

Programa
- Tareas
  - MAST
    - Secciones
      - Cambio_modo
      - Inputs_Estacion_Reading
      - Inputs_FIO_Reading
      - Emergencia
      - Alumno
        - Chart
        - Transiciones
          - T2
          - T1
          - T4
          - T3
          - T5
          - T6
      - Outputs_Estacion_Writing
      - Outputs_FIO_Writing

MAST

Propiedades específicas

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

Cambio_modo : [MAST]

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

```
 1  Modo_simulacion := FIO_boton_simulacion or Ind_int;
 2
 3  if Modo_simulacion then
 4    Guarda_inputs_est := FALSE;
 5    Guarda_inputs_FIO := TRUE;
 6    Obten_salidas_est:= FALSE;
 7    Obten_salidas_FIO:= TRUE;
 8  else
 9    Guarda_inputs_est := TRUE;
10    Guarda_inputs_FIO:= FALSE;
11    Obten_salidas_est:= TRUE;
12    Obten_salidas_FIO:= FALSE;
13
14  end_if;
15
```

Inputs_Estacion_Reading : [MAST]

Propiedades específicas

Nombre de la condición	Guarda_inputs_est

```
    1|      10|      20|      30|      40|      50|      60|      70|      80|      90|     100|     110|
 1  Brazo_atras:=Estacion_Brazo_atras;
 2  Brazo_adelante:=Estacion_Brazo_adelante;
 3  Brazo_Girado_izquierda:=Estacion_Brazo_Girado_izquierda;
 4  Brazo_Girado_derecha:=Estacion_Brazo_girado_derecha;
 5  Pinza_arriba:=Estacion_Pinza_arriba;
 6  Pinza_abajo:=Estacion_Pinza_abajo;
 7  Culata_detectada:=Estacion_Culata_detectada;
 8  Seta_Emergencia:=Estacion_seta_emergencia;
 9  Marcha:=Estacion_Marcha;
10  Rearme:=Estacion_Rearme;
11
12  Copia_Rearme:=Rearme;(* con la intención de detectar flancos *)
13
```

Inputs_FIO_Reading : [MAST]

Propiedades específicas

Nombre de la condición	Guarda_inputs_FIO

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

  1  Brazo_adelante:=FIO_brazo_adelante;
  2  Brazo_atras:=FIO_brazo_atras;
  3  Culata_detectada:=FIO_culata_detectada;
  4  Seta_Emergencia:=not FIO_emergency_stop; (*en factory el sensor de la seta está a 1 cuando no está pulsada*)
  5  Marcha:=FIO_marcha;
  6  Rearme:=FIO_rearme;
  7
  8  Copia_Rearme:=Rearme;(* con la intención de detectar flancos *)
  9
 10  (*el sensor de movimiento está apagado, no se mueve la pinza y los sensores están activos quiere decir que la
10>>pinza está abajo*)
 11  if (FIO_pinza_abajo or FIO_pinza_abajo2) and not FIO_pinza_arriba then
 12          Pinza_abajo:=1;
 13          Pinza_arriba:=0;
 14          else
 15          Pinza_abajo:=0;
 16  end_if;
 17
 18  (*Caso de que no haya detectado la pinza abajo y se produzca un flanco de bajada que indica un movimiento en
18>>la pinza, es decir habrá subido*)
 19  if (FE(FIO_pinza_arriba) and not (FIO_pinza_abajo or FIO_pinza_abajo2)) then
 20          Pinza_arriba:=1;
 21  end_if;
 22
 23
 24  (*Para el caso del sensor de giro se leen las salidas roscarCW y roscarCCW ya que solo hay un sensor de
24>>movimiento en el robot*)
 25  if FE(FIO_brazo_girando) and FIO_roscar_izquierda then
 26          Brazo_Girado_izquierda:=1;
 27          Brazo_Girado_derecha:=0;
 28  end_if;
 29
 30  if FE(FIO_brazo_girando) and FIO_roscar then
 31          Brazo_Girado_derecha:=1;
 32          Brazo_Girado_izquierda:=0;
 33  end_if;
 34
```

Emergencia : [MAST]

```
    1|       10|       20|       30|       40|       50|       60|       70|       80|       90|      100|      110|
  1 if Seta_Emergencia then
  2   En_emergencia:=TRUE;
  3   Listo_para_Rearme := FALSE;
  4
  5   Borrar_charts:=TRUE;
  6   Resultado_clear := clearchart(Alumno, Borrar_charts);
  7   (* borrar_charts:=False; *)
  8 end_if;
  9
 10 Bajar_pinza:=0;
 11 Roscar:=0;
 12 Sacar_culata:=0;
 13 Cerrar_mordaza:=0;
 14 Cerrar_pinza:=0;
 15 Avanzar_Brazo:=0;
 16 Retroceder_Brazo:=0;
 17
 18 (* no puedo mover el robot sin aire, por eso hay que esperar a que la seta se levante *)
 19 if En_emergencia and not Seta_Emergencia then
 20   Listo_para_Rearme := TRUE;
 21 end_if;
 22
 23 if Listo_para_Rearme and RE(Copia_Rearme) then
 24   Rearmando := TRUE;
 25 end_if;
 26
 27 if Rearmando then
 28   if not Brazo_atras then
 29     Retroceder_brazo := TRUE;
 30   else
 31     Retroceder_brazo := FALSE;
 32
 33     En_emergencia:=FALSE;
 34     Listo_para_Rearme := FALSE;
 35     Rearmando := FALSE;
 36     Borrar_charts:=FALSE;
 37     Resultado_init:=initchart(Alumno,TRUE);
 38   end_if;
 39 end_if;
 40
```

Alumno : [MAST]

Comentario

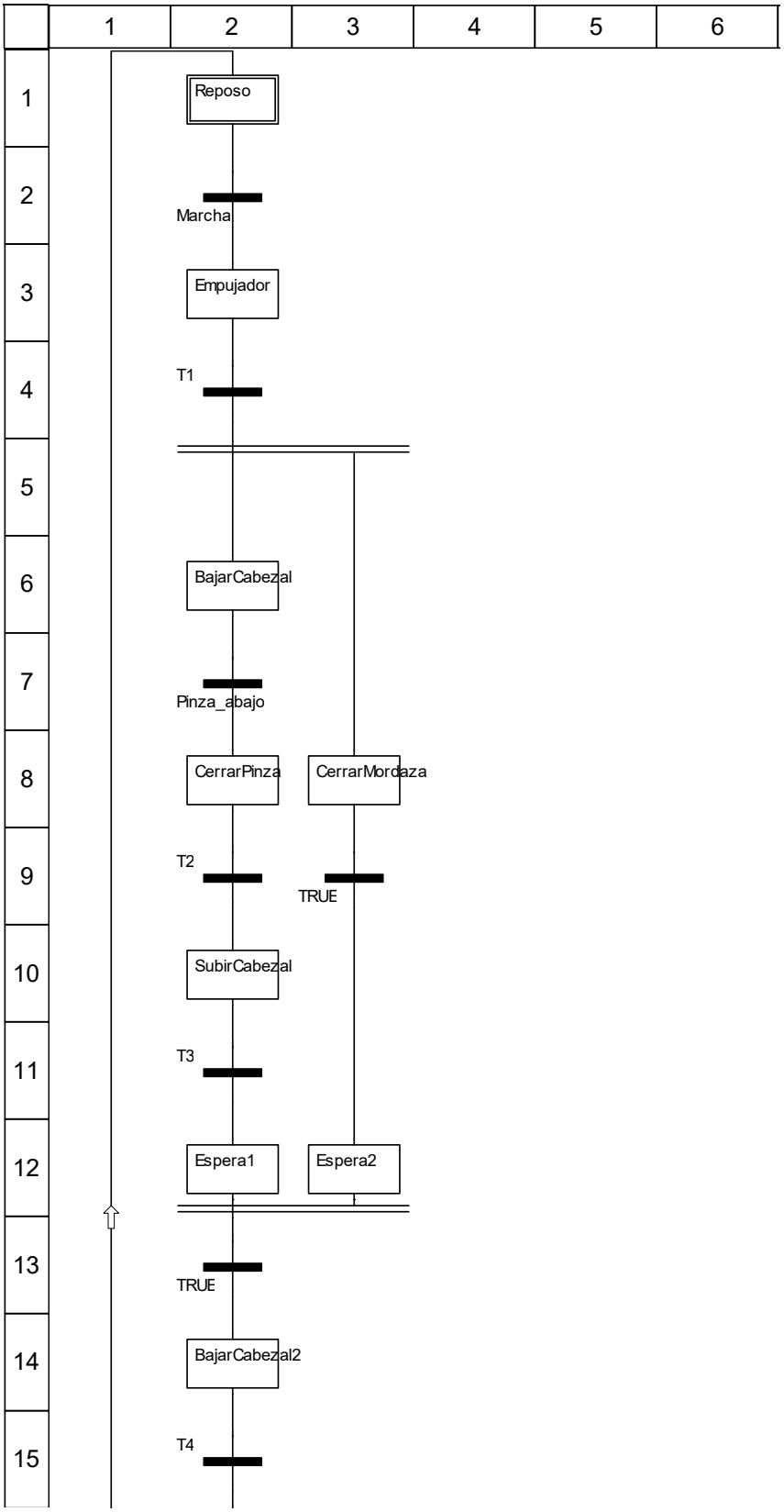
Propiedades comunes

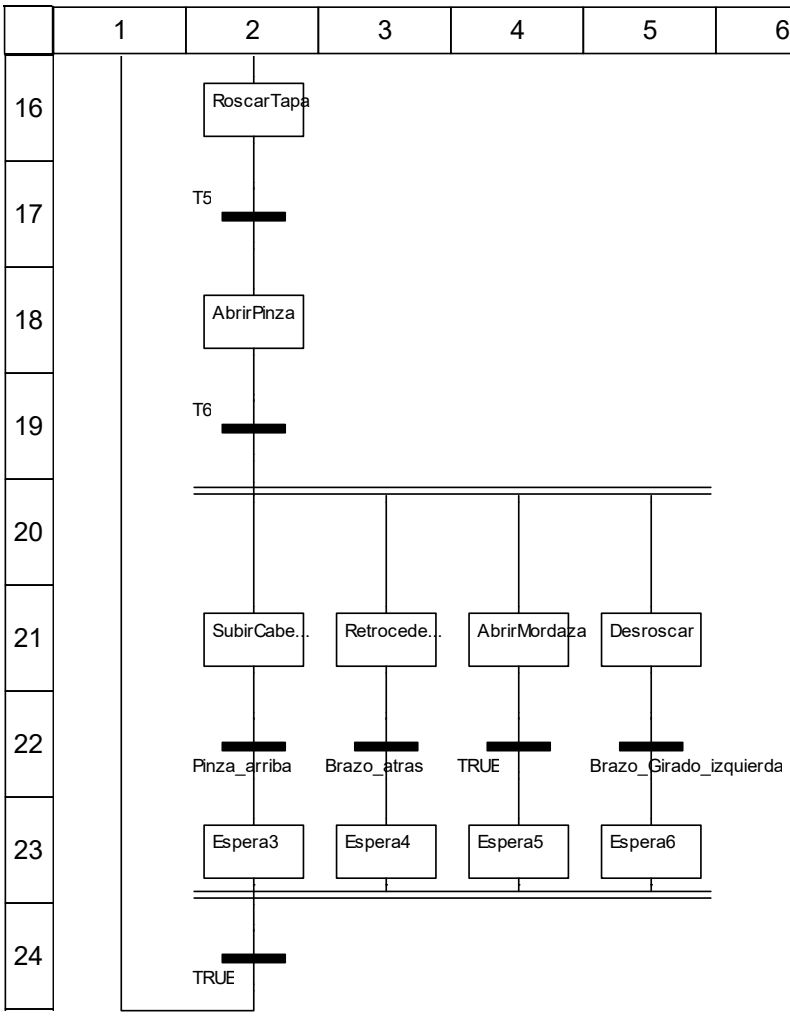
Módulo funcional	
Nombre de la condición	

Propiedades específicas

Control de operador	No
Número de área	0

Chart : [MAST - Alumno]





Descripción de objeto

Pasos:

AbrirMordaza	(4, 21)
Tiempo de supervisión mín./máx.:	Tiempo de retardo:
Comentario:	

AbrirPinza		(2, 18)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Cerrar_mordaza
Descriptor: None	Tiempo:	Variable: Bajar_pinza
Descriptor: None	Tiempo:	Variable: Roscar

BajarCabezal		(2, 6)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Bajar_pinza

BajarCabezal2		(2, 14)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Bajar_pinza
Descriptor: None	Tiempo:	Variable: Cerrar_mordaza
Descriptor: None	Tiempo:	Variable: Cerrar_pinza

CerrarMordaza		(3, 8)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Cerrar_mordaza

CerrarPinza		(2, 8)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Bajar_pinza
Descriptor: None	Tiempo:	Variable: Cerrar_pinza

Desroscar		(5, 21)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		

Empujador		(2, 3)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Sacar_culata

Espera1		(2, 12)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Cerrar_pinza

Espera2		(3, 12)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Cerrar_mordaza

Espera3		(2, 23)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		

Espera4		(3, 23)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		

Espera5		(4, 23)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		

Espera6		(5, 23)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		

Reposo (paso inicial)		(2, 1)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		

RetrocederBrazo		(3, 21)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Retroceder_Brazo

RoscarTapa		(2, 16)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Cerrar_mordaza
Descriptor: None	Tiempo:	Variable: Roscar
Descriptor: None	Tiempo:	Variable: Bajar_pinza
Descriptor: None	Tiempo:	Variable: Cerrar_pinza

SubirCabezal		(2, 10)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		
Acciones:		
Descriptor: None	Tiempo:	Variable: Cerrar_pinza
Descriptor: None	Tiempo:	Variable: Avanzar_Brazo

SubirCabezal2		(2, 21)
Tiempo de supervisión mín./máx.:		Tiempo de retardo:
Comentario:		

Transiciones:

Nombre	Tipo de condición	Posición	Comentario
Brazo_Girado_izquierda	Variable	(5, 22)	
Brazo_atras	Variable	(3, 22)	
Marcha	Variable	(2, 2)	
Pinza_abajo	Variable	(2, 7)	
Pinza_arriba	Variable	(2, 22)	
ST :: T1	Sección	(2, 4)	
ST :: T2	Sección	(2, 9)	
ST :: T3	Sección	(2, 11)	
ST :: T4	Sección	(2, 15)	
ST :: T5	Sección	(2, 17)	
ST :: T6	Sección	(2, 19)	
TRUE	Constante	(2, 13)	
TRUE	Constante	(2, 24)	
TRUE	Constante	(3, 9)	

TRUE	Constante	(4, 22)	

T2 <Transición> : [MAST - Alumno]

```
   1|       10|       20|       30|       40|       50|       60|       70|       80|       90|       100|       110|
1  CerrarPinza.t>=t#1s and Pinza_abajo and Brazo_atras
```

T1 <Transición> : [MAST - Alumno]

```
   1|       10|       20|       30|       40|       50|       60|       70|       80|       90|       100|       110|
1  Empujador.t>=t#1.5s and Culata_detectada
```

T4 <Transición> : [MAST - Alumno]

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

T3 <Transición> : [MAST - Alumno]

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

T5 <Transición> : [MAST - Alumno]

```
   1|       10|       20|       30|       40|       50|       60|       70|       80|       90|       100|       110|
1  Pinza_abajo and Brazo_adelante and Brazo_Girado_derecha
```

T6 <Transición> : [MAST - Alumno]

```
   1|       10|       20|       30|       40|       50|       60|       70|       80|       90|       100|       110|
1  AbrirPinza.t>=t#0.1s and Brazo_adelante and Pinza_abajo and Brazo_Girado_derecha
```

Outputs_Estacion_Writing : [MAST]

Propiedades específicas

Nombre de la condición	Obten_salidas_est

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

 1  Estacion_Avanzar_brazo:=Avanzar_Brazo;
 2  Estacion_Retroceder_brazo:=Retroceder_Brazo;
 3  Estacion_Roscar:=Roscar;
 4  Estacion_Bajar_pinza:=Bajar_pinza;
 5  Estacion_Sacar_culata:=Sacar_culata;
 6  Estacion_Cerrar_mordaza:=Cerrar_mordaza;
 7  Estacion_Cerrar_pinza:=Cerrar_pinza;
 8
```

Outputs_FIO_Writing : [MAST]

Propiedades específicas

Nombre de la condición	Obten_salidas_FIO

```
    1|       10|        20|        30|        40|        50|        60|        70|        80|        90|       100|       110|
  1  FIO_sacar_culata:=Sacar_Culata;
  2  FIO_bajar_pinza:=Bajar_Pinza;
  3  FIO_cerrar_pinza:=Cerrar_Pinza;
  4  FIO_roscar:=Roscar;
  5  FIO_cerrar_mordaza:=Cerrar_Mordaza;
  6  FIO_avanzar_retroceder_brazo:=Avanzar_Brazo;
  7
  8  if Retroceder_Brazo=1 and not Avanzar_Brazo then
  9          FIO_avanzar_retroceder_brazo:=0;
 10  end_if;
 11
 12  (*Caso de que el alumno no active avanzar brazo porque ya está delante, pero en factory io si no se activa
12>>avanzar brazo vuelve detras "cilindro simple efecto"*)
 13  if Retroceder_brazo=0 and Avanzar_Brazo=0 and Brazo_adelante=1 then
 14          FIO_avanzar_retroceder_brazo:=1;
 15  end_if;
 16
 17  if FE(FIO_roscar) then
 18          FIO_roscar_izquierda:=1;
 19  end_if;
 20
 21  (*para cuando se haya activado desroscar como no se pone a cero, en caso de roscar otra vez se ponga a cero
21>>para poder desroscar despues*)
 22  if RE(FIO_roscar) then
 23          FIO_roscar_izquierda:=0;
 24  end_if;
 25
 26  (*Supongo que aqui se escribirá la parte del caso de rearme y/o emergencia*)
 27
 28  (*PARTE DE LA ILUMINACIÓN DE LOS SENSORES*)
 29
 30  FIO_indicador_brazo_adelante:=Brazo_adelante;
 31  FIO_indicador_brazo_atras:=Brazo_Atras;
 32  FIO_indicador_brazo_girado:=Brazo_girado_derecha; (*el indicador apagado muestra que el brazo está girado a
32>> izquierdas*)
 33  FIO_indicador_pinza_abajo:=Pinza_abajo;
 34  FIO_indicador_pinza_arriba:=Pinza_arriba;
 35  FIO_indicador_culata_detectada:=Culata_detectada;
 36  FIO_indicador_marcha:=Marcha;
 37  FIO_indicador_seta_emergencia:=Seta_emergencia;
 38
 39  (*PARTE DE PONER LAS PIEZAS SOBRE LA ZONA DE TRABAJO*)
 40  if not FIO_poner_piezas then (*not porque de normal el sensor está activo, y al pulsar se apaga lo que quiere
40>> decir que el usuario desea meter piezas*)
 41          FIO_dispensador_tapas:=TRUE;
 42          FIO_dispensador_bases:=TRUE;
 43          else
 44          FIO_dispensador_tapas:=FALSE;
 45          FIO_dispensador_bases:=FALSE;
 46  end_if;
 47
 48  FIO_indicador_poner_piezas:=not FIO_poner_piezas;
```