UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

CONTROL DE POSICIÓN DE UN PARKING AUTOMÁTICO

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELÉCTRICA

Autor:

Rego Máñez, Guillem

Tutor:

Puche Panadero, Rubén

Cotutor:

Sapena Bañó, Ángel

SEPTIEMBRE DE 2019

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

CONTROL DE POSICIÓN DE UN PARKING AUTOMÁTICO

1. MEMORIA

SEPTIEMBRE DE 2019

GUILLEM REGO MÁÑEZ

RUBÉN PUCHE PANADERO Y ÁNGEL SAPENA BAÑÓ

Tabla de contenido

1.1 OBJETIVOS

El objetivo principal de este Trabajo Fin de Grado es realizar el control de un parking robotizado, de tal forma que se controlen los movimientos en los 3 ejes que posee el parking mediante el control de sus accionamientos a través de un autómata programable. A partir de este objetivo general, se pueden generar una serie de objetivos específicos:

- Definir el funcionamiento del parking y los requisitos
- Realizar el programa de control industrial para el autómata programable encargado del control de los 3 accionamientos de los 3 ejes empleados para el aparcamiento automático de vehículos.
- Realizar el sistema de visualización y monitorización de los procesos mediante el uso de un sistema SCADA en un ordenador personal.
- Realizar una simulación o emulación a través de un sistema SCADA, ya que se dispone de unos accionamientos de baja potencia, pero similares a los necesarios.
- Añadir un ítem sobre genericidad debido al programa que se podrá adaptar a otro sistema de aparcamiento.

1.2 ANTECEDENTES

La ingeniería lleva varios años aplicando la tecnología de los PLC's a la industria y también a otros campos, debido, principalmente, a la enorme versatilidad, adaptabilidad y relativa sencillez de implementación.

Debido a que la automatización está llegando cada vez a más campos, no es extraño ver que ha llegado ya a los parkings para facilitar el aparcamiento a los usuarios y otorgarle la comodidad de no tener que buscar plaza.

En Valencia tenemos un ejemplo, el primer parking automatizado de la ciudad se llevó a cabo en 2012, es el parking Pizarro 10 situado en la calle del mismo nombre. El parking cuenta con 182 plazas. El sistema de este parking recoge el coche de la plataforma en que el conductor lo ha dejado, lo eleva y lo traslada, mediante unos

ascensores, hasta alguno de los espacios de almacenamiento donde se posiciona el vehículo en el punto designado, sin que el conductor tenga que conducir el coche hasta la plaza concreta. Una vez finalizado el plazo, el mismo sistema automático localiza el coche y lo transporta hasta la cabina de salida, donde el conductor sólo tiene que recogerlo y abandonar el aparcamiento.

Muchas personas, al estar automatizado un parking piensan que por ese hecho sus tarifas van a ser más caras, sin embargo, éste, tiene unas tarifas similares e, incluso, más reducidas, que la de los aparcamientos convencionales.

Figura 1.1 Entrada Parking Automático

1.3 JUSTIFICACIÓN

Se puede demostrar la ejecución de este proyecto desde dos dimensiones bien diferenciadas que se podrían encuadrar en:

1.3.1 ACADÉMICA

El desarrollo de este proyecto tiene como propósito la obtención del título de Graduado en Ingeniería Eléctrica, de acuerdo con la normativa vigente de educación y ciencia por el ministerio.

El proyecto ha sido realizado en la Escuela Técnica Superior de Ingeniería del Diseño de la Universitat Politècnica de València, dirigido por el profesor Rubén Puche Panadero y Ángel Sapena Bañó.

1.3.2 FUNCIONAL

La realización del proyecto es muy importante para un graduado ya que debe poseer los conocimientos y facultades óptimos sobre la automatización y control de procesos industriales debido a la gran utilidad que tiene hoy en día.

Adquiriremos una serie de conocimientos muy importantes para un ingeniero:

- Conocimientos sobre autómatas programables y sus lenguajes de programación.
- Conocimientos sobre sistemas SCADA y monitorización de datos.

1.4 ESTUDIO DE LA VIABILIDAD

El estudio de viabilidad consiste en realizar una valoración para determinar la conveniencia y posibilidad de llevar a término el proyecto. Es importante tener en cuenta todos los aspectos de carácter técnico, académico y económico que fijan exactamente la ejecución material del presente proyecto.

En este proyecto solo vamos a tener en cuenta el aspecto académico ya que predomina sobre el aspecto económico en el diseño del proyecto, los elementos que se usan en el proyecto son utilizados en la industria en gran medida y tienen una gran difusión en el mundo industrial.

La viabilidad económica se va a reducir al estudio económico realizado en el presupuesto donde se van a incluir los precios de los dispositivos, accesorios eléctricos, así como la mano de obra.

La viabilidad humana es un factor a tener en cuenta, debido a los costes de aprendizaje, los métodos y medidas físicas necesarias para el control de los procesos realizados en el proyecto.

Debido a no tener datos reales del proceso no se ha podido realizar el anterior punto del estudio de la viabilidad.

1.5 FACTORES A CONSIDERAR: ESTUDIO DE NECESIDADES, LIMITACIONES Y CONDICIONANTES

Cabe destacar que el proceso de control de los servomotores está realizado de modo que podría servir para distintas aplicaciones.

1.5.1 ESPECIFICACIONES DEL ENCARGO

Se trata de un parking en el que hay 110 plazas donde habría 109 disponibles y 1 que sería por donde accederían los clientes y dejarían su vehículo para que fuera estacionado automáticamente en alguna de las 109 plazas disponibles.

Las plazas estarían dispuestas en 2 hileras, una detrás de otra, separadas por el espacio por dónde se moverá la plataforma que controlamos. Cada hilera tendría 5 pisos de altura y cada piso tendría 11 plazas de aparcamiento. Podemos ver una disposición similar en la figura 1.2.

Figura 1.2 Disposición del parking

El parking contará además con una barrera que subirá y bajará dependiendo de la disponibilidad de aparcamientos y donde al estacionar un vehículo el cliente extraerá un ticket con un código.

Dicho ticket lo usará el cliente para retirar el vehículo ya que ese ticket nos dará la posición del vehículo dentro del parking, y para prevenir que no se dé mal uso la máquina que expende el ticket y lee el código de este marcará cuando se ha usado un ticket.

De esta última parte solo se nos pide usar el código que leerá la máquina, que nos dará el número de fila, columna y profundidad, para retirar el vehículo estacionado, pero no entrará en nuestro encargo realizar el sistema de extracción y lectura de tickets.

El software tiene que permitir un control total sobre el posicionado del eje X, Y y Z.

Cabe destacar que la introducción en el sistema de autómatas posibilita que se puedan ir produciendo mejoras en el diseño a posteriori, sin ningún aumento del coste en material.

1.5.2 ESTUDIO DE NECESIDADES PROPIAS

En este apartado hay que señalar las necesidades, limitaciones y condiciones que se han tenido durante la realización del presente proyecto. Se debe tener en cuenta que hay que intentar conseguir las mejores prestaciones, tanto en el ámbito técnico como en el ámbito económico, lo que supone intentar alcanzar un punto medio entre ambos niveles.

Se deberá disponer de todo el equipamiento necesario para la programación, así como las mejores condiciones del mismo para que se elabore con robustez, para asegurar una fiabilidad y prestaciones suficientes que permitan realizar el proyecto con seguridad.

También se requiere de una asistencia técnica para la atención de cualquier duda relacionada con la programación del proyecto. Para ello es necesario el contacto telefónico y por correo electrónico con los agentes técnicos de ABB.

En cuanto a las necesidades o limitaciones propias al equipo: serán las impuestas por el propio equipo utilizado.

En cuanto a la programación del software: no se han encontrado limitaciones que no se hayan resuelto con ingenio y con la ayuda de manuales y libros sobre el mismo. Se han delimitado unos márgenes de seguridad para las acciones de control que intervienen en cada uno de los actuadores para garantizar la durabilidad de estos dispositivos al máximo.

1.6 DESCRIPCIÓN DEL PROCESO A CONTROLAR

Nuestro proceso a controlar se trata del movimiento de una plataforma que se mueve mediante guías a lo largo de 3 ejes (x, y, z) en un parking automatizado.

Deberemos diseñar un sistema en el que cuando se reciba una petición de aparcamiento, en primer lugar busquemos una plaza vacía y luego movamos la plataforma en el eje x para moverla al espacio entre ambas hileras de plazas de aparcamiento y tras esto se moverá en el eje y para llegar a la altura requerida, tras esto se hará el movimiento en horizontal a través del eje z y por último tendremos que volver a mover la plataforma en el eje x para estacionar el vehículo.

Deberemos también regresar la plataforma a la posición inicial, que será realizar el trayecto para el estacionamiento del vehículo, pero a la inversa.

Siguiendo un método parecido deberemos desarrollar un sistema en el que con el código que nos dé la máquina que lee el ticket de petición de retirada de vehículo, deberemos ser capaces de retirar dicho vehículo. El código que obtendremos del ticket nos dará 3 números que corresponderán a la fila, columna y profundidad.

Para el movimiento de la plataforma el sistema será similar al de estacionamiento del vehículo.

Figura 1.3 Parking automático

También deberemos realizar un SCADA donde podamos visualizar el proceso y realizar una simulación.

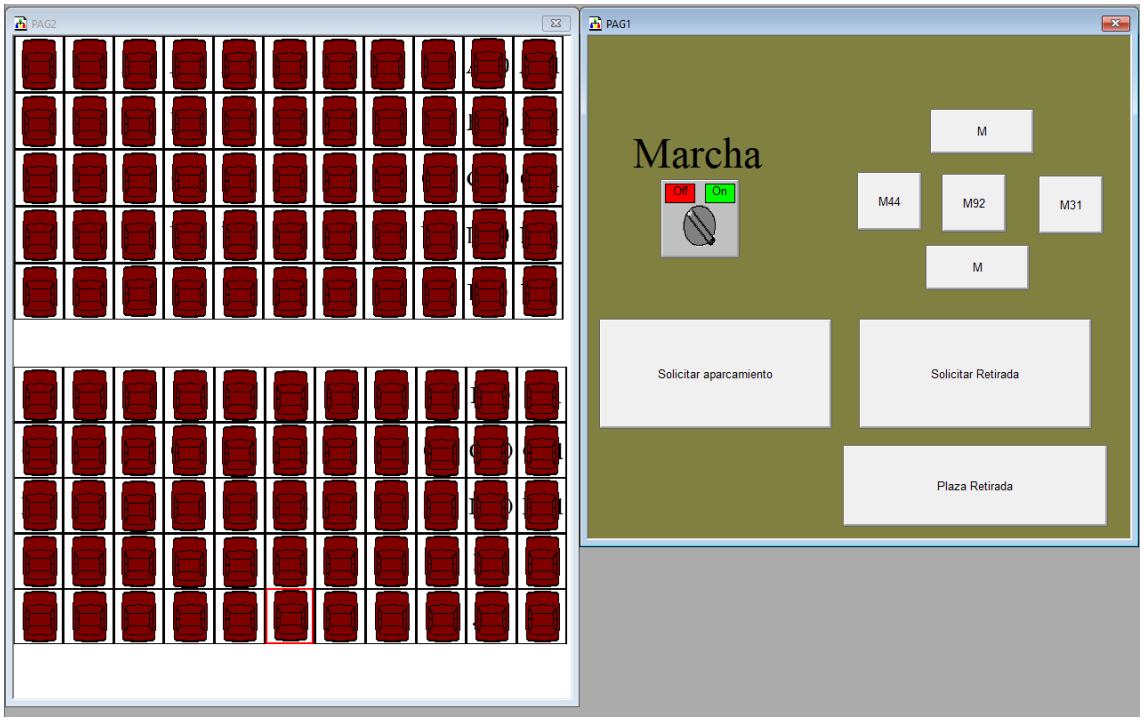


Figura 1.4 Diseño del SCADA

1.6.1 BARRERA AUTOMÁTICA Y EXPENDORAS DE TICKETS

En la entrada del parking se encontrará una barrera automática que estará bajada cuando la plataforma no se encuentre y subida cuando la plataforma se encuentre en la posición inicial.

Precediendo a esta barrera, al lado de la puerta de salida habrá una máquina expendedora de tickets que tendrá una ranura, dos botones y un lector de código de barras. Un botón que será el de solicitar aparcamiento que solo pondrá en marcha el movimiento de nuestra plataforma de deslizamiento si la plataforma elevadora detecta que hay un vehículo sobre nuestra plataforma, y si es el caso una vez haya detectado nuestro PLC la plaza donde se va a estacionar el vehículo nos proporcionará el ticket, con el día y hora de estacionamiento y un código de barras. El otro botón será para solicitar retirada del vehículo, que se realizará una vez hayamos pasado el código de barras de nuestro ticket por el lector.

1.6.2 PLATAFORMA DE DESPLAZAMIENTO

Nuestro parking contará con una plataforma que se desplazará a través de los ejes X, Y y Z teniendo acoplados y asignados a cada uno de ellos 1 servomotor. Esta plataforma se encontrará en la entrada del parking precedida por la barrera automática anteriormente mencionada. El control de la posición de esta plataforma será nuestro encargo. Esta plataforma tendrá unas medidas de 2,5 metros de ancho por 5,5 metros de largo.

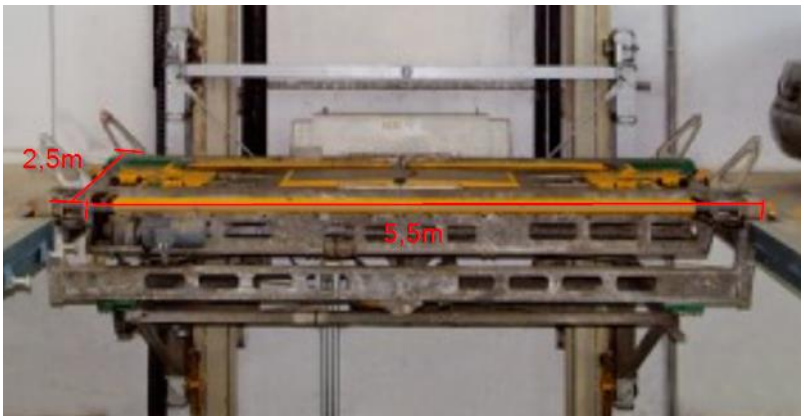


Figura 1.5 Plataforma de desplazamiento y sus medidas

1.6.3 PLATAFORMA ELEVADORA

En nuestra plataforma habrá una plataforma elevadora que será de ancho 30 centímetros cada lado, con ranuras y piezas metálicas de 25 centímetros de ancho y 3 centímetros de largo conforme los ejes de nuestra plataforma de desplazamiento, que corresponderá un lado a la posición de los neumáticos izquierdos y el otro lado a la de los derechos.

Su funcionamiento será elevar el vehículo cuando se vaya a estacionar ya que en cada plaza habrá a los laterales unas plataformas donde descansarán los neumáticos y los vehículos de tal forma que la plataforma elevadora descenderá cuando la plataforma de desplazamiento haya alcanzado la plaza requerida y por tanto la plataforma de desplazamiento se podrá retirar sin desplazar el vehículo.

A la hora de la retirada, cuando la plataforma esté en la plataforma elevadora se elevará y entrará en contacto con los neumáticos del vehículo, descansando estos ahora en nuestra plataforma de forma que se realizará el movimiento de la plataforma de desplazamiento y el vehículo será retirado.

Figura 1.6 Plataforma elevadora junto a sus ranuras

1.7 PLANTEAMIENTO DE SOLUCIONES ALTERNATIVAS. DESCRIPCIÓN DE LOS CRITERIOS DE SELECCIÓN Y JUSTIFICACIÓN DE LA SOLUCIÓN ADOPTADA.

Debido a la flexibilidad de los elementos y a la gran variedad de componentes y software existentes en el mercado nuestro proceso de control y automatización se puede realizar de múltiples formas diferentes. Se ha elegido este modo porque proporciona una gran flexibilidad al permitir controlar el proceso desde cualquier PC.

Las soluciones adoptadas en la realización de este proyecto van en función de las necesidades y limitaciones expuestas en el apartado 1.5 y del material existente en el Departamento de Ingeniería Eléctrica de la Universitat Politècnica de València.

1.7.1 ELECCIÓN DEL AUTÓMATA

CJ2M de Omron: Es un PLC modular. Esta unidad dispone de una gran cantidad de módulos que permiten la ampliación para la comunicación serie o la ampliación de entradas y salidas. Esta serie dispone de total compatibilidad con las tecnologías de redes abiertas más usuales.

Figura 1.7 PLC CJ2M de Omron

Entre las características del CJ2M se encuentran:

- Puerto Ethernet
- Puerto USB
- Módulos de E/S de pulsos, con estos módulos se dispondrá de control de frecuencia de pulso, control de ancho de pulso y entradas de interrupción entre otras.

SIMATIC S-7 1200 de Siemens: Es un PLC compacto que nos ofrece potencia y flexibilidad para controlar una amplia variedad de dispositivos.

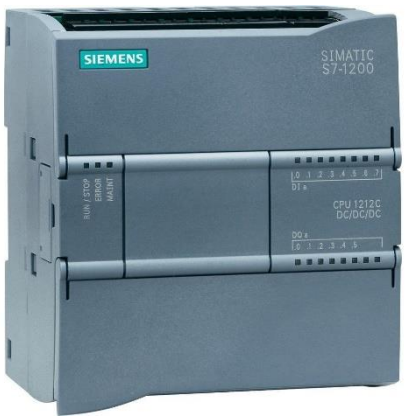


Figura 1.8 PLC SIMATIC S-7 1200 de Siemens

Entre las características del SIMATIC S-7 1200 se encuentran:

- Alta capacidad de procesamiento. Cálculo de 64 bits.
- Interfaz Ethernet / PROFINET integrado.
- Bloques de función para control de ejes conforme a PLCopen.
- Entradas analógicas integradas.
- Entradas de alta velocidad para contaje y medición
- Salidas de alta velocidad para regulación de velocidad, posición y punto de operación
- Funcionalidad PID para lazos de regulación

AC500 PM571 de ABB: La plataforma AC500 es una gama de PLC's escalable, flexible y de alto rendimiento, que tiene una gran capacidad de expansión. Permite una multitud de combinaciones (entradas/salidas analógicas y digitales, bus de campo y/o redes de comunicación) para cumplir los requisitos con un número mínimo de referencias.
Integra todas las nuevas tecnologías en cuanto a capacidades, canales digitales configurables como entrada o salida, una gran variedad de comunicaciones, una amplia memoria de datos y programa, una tarjeta SD de memoria opcional y estándar para registrar datos y operaciones de mantenimiento (carga/descarga de programa o firmware, etc). Una de sus ventajas principales es la compatibilidad total entre todas las CPUs AC500 y AC500-eCo y los módulos de E/S S500 lo que hace de la plataforma AC500 una plataforma única.

Figura 1.9 PLC AC500 PM571 de ABB

Entre las características del AC500 PM571 se encuentran:

- 64kB de memoria de programa
- Tiempo de ciclo para 1 instrucción: 0,3 ms si se trata de un bit o una palabra y 6ms si se trata de coma flotante.
- Posibilidad de expansión con hasta 7 módulos de E/S S500 y/o S500-eCo

Para la realización de este proyecto se ha escogido el AC500 PM571 de ABB ya que cumple con las características necesarias para la realización del encargo y disponemos de material en el laboratorio para conectar los motores y realizar el control de posición.

1.7.2 LENGUAJE DE PROGRAMACIÓN

En la actualidad existen abundantes lenguajes de programación completamente diferentes que han ido evolucionando con el paso del tiempo, en un principio cada fabricante desarrolla su propio lenguaje y por tanto el usuario debía estudiar ese lenguaje. Con el paso del tiempo se han ido unificando los lenguajes y estandarizando una serie de ellos que permiten programar cualquier tipo de autómata independientemente de su fabricante y por lo tanto poder sustituir autómatas que controlen un proceso por otros diferentes sin tener que modificar el programa.

La norma IEC 61131-3 establece los siguientes lenguajes de programación:

- Lenguaje por lista de funciones (IL)

Se trata de un lenguaje de texto, nemotécnico, en el cual cada línea de programa contiene una sola instrucción y su ejecución es secuencial. Todos los operadores trabajan con un registro especial, denominado acumulador.

```
D00   LD    %I0.1   Bp. inicio ciclo
      AND   %I0.0   Dp. presencia vehículo
      AND   %M3     Bit autorización reloj calendario
      AND   %I0.5   Fc. alto rodillo
      AND   %I0.4   Fc. detrás pórtico
D05   S     %M0     Memo inicio ciclo
      LD    %M2
      AND   %I0.5
      OR    %I0.2   Bp. parada ciclo
      R     %M0
D10   LD    %M0
      ST    %Q0.0   Piloto ciclo
```

Figura1.10 Programación en IL

- Diagrama de contactos (LD)

Es un lenguaje de programación gráfico que proviene del antiguo lenguaje de relés utilizado por electricistas para elaborar los antiguos cuadros de automatismo. Su principal ventaja es que todos los símbolos están normalizados según la norma IEC 61131-3 y son empleados por todos los fabricantes, por lo tanto, es el lenguaje más universal.

Los softwares de programación de los autómatas en modo monitorización muestran las entradas y salidas que están conectadas en cada momento iluminándolas, lo que facilita detectar de forma muy gráfica y rápida averías y errores en las instalaciones.

Los principales símbolos que componen este lenguaje los podemos observar en la figura 1.11

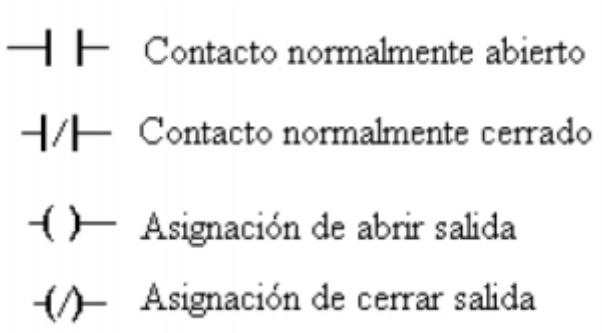


Figura 1.11 Símbolos en programación en LD

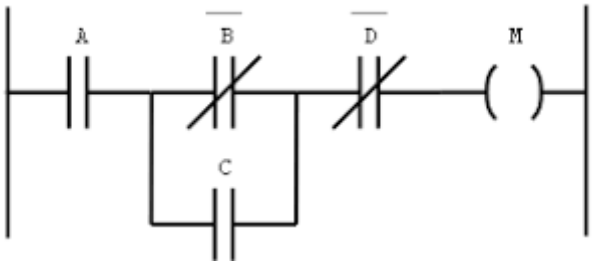


Figura 1.12 Programación en LD

- Lenguaje por plano de funciones (FBD)

Es otro tipo de lenguaje gráfico, sin embargo, éste proviene del campo de procesamiento de señales. Permite elementos de programa que se unen en forma análoga a compuertas lógicas en un circuito electrónico.
Cada función lógica tiene asociado un bloque funcional que realiza la operación que corresponde con grafos estándar.

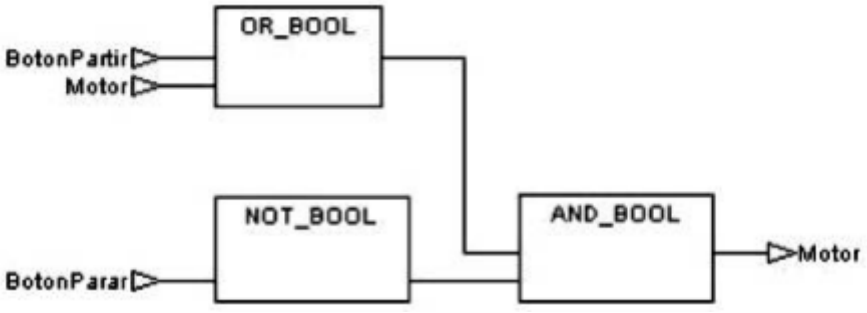


Figura 1.13 Programación en FBD

- Lenguaje grafcet (SFC)

Es un lenguaje diseñado para la automatización de procesos secuenciales. Está basado en los llamados Gráficos de orden de Etapa Transición. Es un lenguaje muy práctico porque simplifica muchísimo la programación de automatismos secuenciales haciéndolos más fácil de interpretar y menos engorrosos que utilizando otros lenguajes.

Una vez automatizado el proceso en lenguaje Grafcet es muy sencillo transformarlo en diagramas de contactos, aunque actualmente la mayoría de los autómatas ya permiten la utilización del Grafcet como tal y no es necesario transformarlo en diagramas de contactos.
Los softwares de programación de los autómatas en modo monitorización iluminan los estados y transiciones activos, lo que permite saber de manera gráfica y rápida en qué parte del proceso se encuentra el sistema en cada momento, pudiendo detectar así fácilmente posibles averías y errores en el sistema que se está controlando.

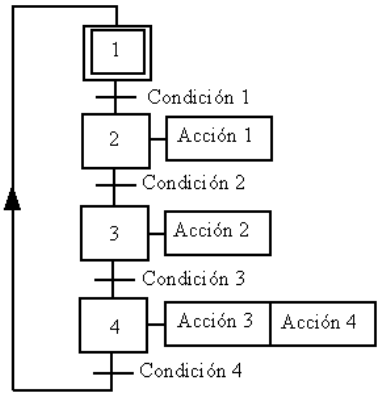


Figura 1.14 Programación en Grafcet

- Texto estructurado (ST)

El texto estructurado es un lenguaje que se compone de una serie de instrucciones que se pueden ejecutar de forma condicionada o en bucles secuenciales. Se caracteriza por tener un código dividido en divisiones fácilmente legibles llamadas funciones y por ser un lenguaje donde las variables deben ser siempre declaradas para poder hacer uso de ellas.

```
if aux_1 = true then
      desplaza_izq := true;
      velocidad_salida := velocidad;
else
desplaza_izq :=false;
velocidad_salida := 0;
end_if;
if sensor_p0 = true then
      desplaza_der := true;
      velocidad_salida := velocidad * 10 / 100;
      contador_pulsos1 := encoder;
```

Figura 1.15 Programación en ST

1.7.2 SOFTWARE DE PROGRAMACIÓN

El software que utilizaremos para programar el autómata AC500 PM571 es CoDeSys, de la marca ABB, que es utilizado para programar todos sus modelos, con opción de programar en cualquiera de los lenguajes de programación establecidos por la norma IEC-61131-3, además también dispone de un editor gráfico que no está definido en la norma IEC y que se trata del CFC. Más adelante procederemos a explicar brevemente el funcionamiento del programa en el Anexo 1.

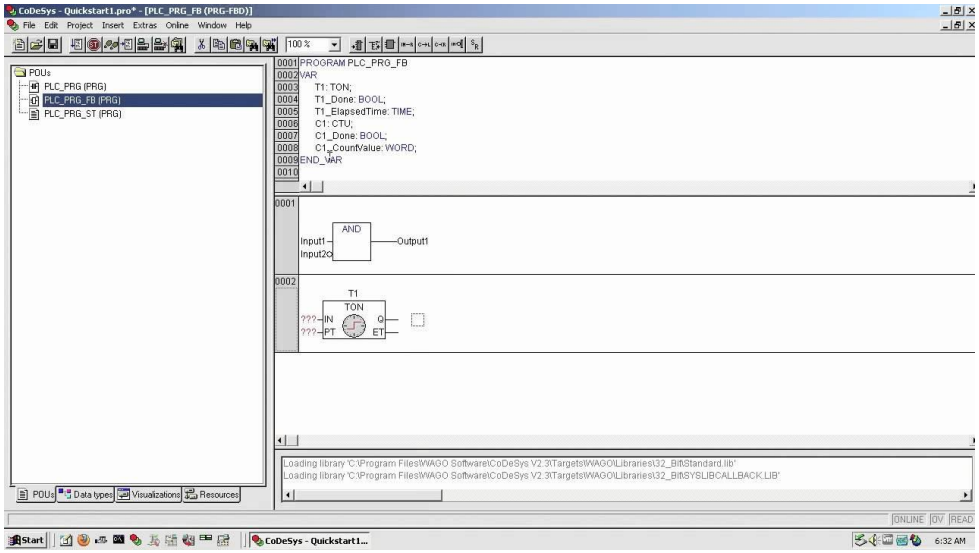


Figura1.16 Programa CodeSys

1.7.3 SCADA

El nombre SCADA significa: (Control Supervisor y Adquisición de datos). Un sistema SCADA es una aplicación o conjunto de aplicaciones software especialmente diseñada para funcionar sobre ordenadores de control de producción. Aunque inicialmente solo era un programa que permitía la supervisión y adquisición de datos en procesos de control, en los últimos tiempos han ido surgiendo una serie de productos hardware y buses especialmente diseñados o adaptados para este tipo de sistemas. La interconexión de los sistemas SCADA también es propia, se realiza una interfaz del PC a la planta centralizada, cerrando el lazo sobre el ordenador principal de supervisión.

El sistema permite comunicarse con los dispositivos de campo (controladores autónomos, autómatas programables, sistemas de dosificación, etc.) para controlar el proceso en forma automática desde la pantalla del ordenador, que es configurada por el usuario y puede ser modificada con facilidad. Además, provee de toda la información que se genera en el proceso productivo a diversos usuarios.

Existen SCADA de distintos tipos y marcas, posiblemente dos de los más conocidos sean WinCC de Siemens y CX-Supervisor de la marca OMRON

WinCC de Siemens:

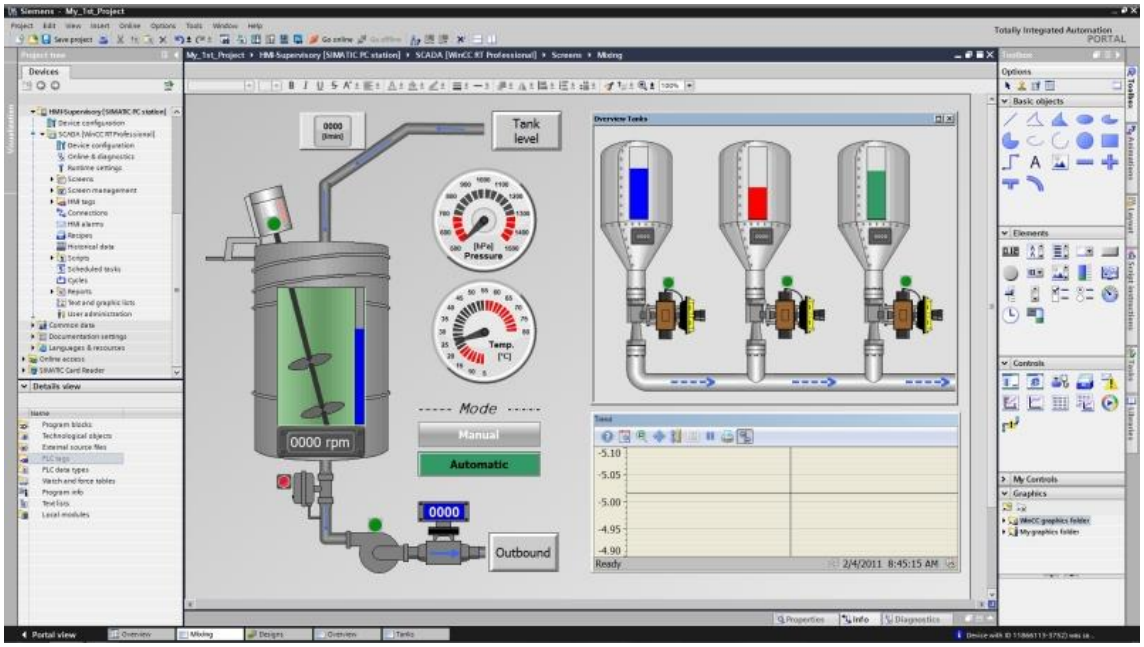


Figura 1.17 Programa WinCC de Siemens

- Uso universal
    - Soluciones para todos los sectores
    - Cumple los requisitos exigidos en la norma 21 CFR, parte 11
    - Multilingüe, para uso en todo el mundo
    - Posibilidad de integración en todas las soluciones de automatización y de TI
- Todas las funciones de manejo y visualización a bordo
- Escalabilidad en toda la línea, también vía web
- Microsoft SQL Server integrado para registro histórico de datos como plataforma de información
- Ampliable con opciones y add-ons

CX-Supervisor de OMRON:

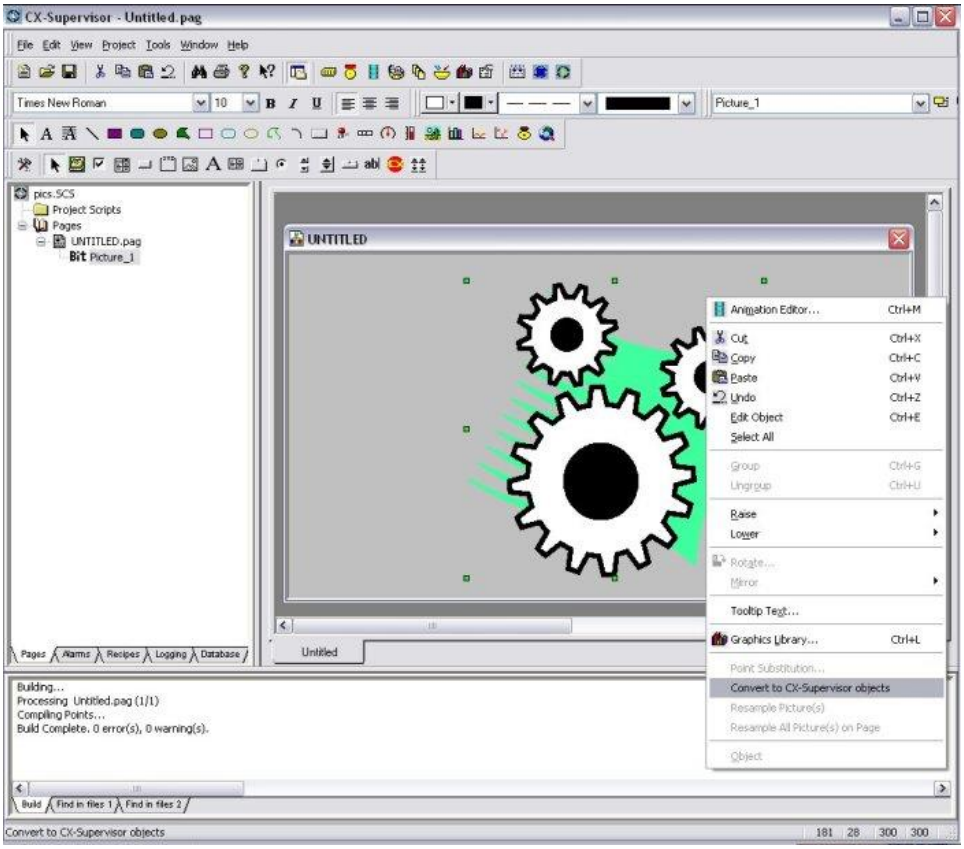


Figura 1.18 Programa CX-Supervisor de OMRON

Las principales características y funciones del paquete SCADA Cx-Supervisor son:

- Programación: La programación es en un entorno Windows de forma intuitiva mediante scripts y ventanas. Los scripts pueden ser propios del programa o Java y Visual Basic Scripts. Las ventanas nos ayudan a configurar las E/S y las funciones de los distintos objetos en la pantalla de aplicación.

- Soporta Drivers Microsoft: COM/DCOM, DDE, OPC, ActiveX (OLE), y tecnologías estándar ODBC/ADO.

- COM/DCOM: Estándar comunicaciones externas de Windows.

- DDE: Estándar comunicaciones internas entre programas en entorno Windows.

Con la incorporación del estándar OPC puede enlazar con cualquier servidor de datos. Con el servidor SIMATIC NET podemos utilizar el control de comunicaciones Cx OPC

Comunication Control para el intercambio de datos entre Cx-Supervisor y nuestro PLC de la marca SIEMENS.

Con el driver ActiveX podemos incluir controles de éste tipo y objetos OLE (browsers Web, Controles o displays MP3...) en las pantallas de aplicación para crear tareas preprogramadas.

- Objetos y Animaciones: Con librerías de más de 3000 objetos los cuales solo tenemos que cogerlos de la librería y ponerlos (Drag & Drop) en la pantalla de aplicación. Podemos asociar a las librerías objetos más complejos tipo OLE o crear nuestras propias librerías. Podemos realizar todo tipo de animaciones con el editor de animaciones (Cambiar de tamaño, movimientos, parpadeos, cambios de color...) sobre los objetos seleccionados o los controles ActiveX de Omron u otros controles de Windows.

- Recetas: Para crear secuencias de programa enteras que pueden ser transferidas hacía o desde un autómata programable concreto para realizar procesos similares. Son colecciones de variables de puntos de E/S a las cuales se les asocia unos valores predeterminados que serán los mismos en cualquier proceso que incluya dichas E/S. Estas pueden ser almacenadas en el disco y utilizadas en conjunto al editar un nuevo proyecto.

- Alarmas: Proporciona en la pantalla de aplicación una notificación de problemas durante el runtime del proceso y/o visualización de entradas salidas concretas. Tiene tres tipos: Simple, banda muerta (deadband) y rango de cambio. Tiene aplicaciones de historial de alarmas y aplicación, durante runtime, de visor de conocimiento de alarma (alarm confirmation viewer) para asegurar que el operario ha visto la alarma y actúa en consecuencia. Es tan simple como una ventana que surge durante el proceso, cuando una alarma salta.

Como este es un proyecto académico se va a utilizar el SCADA CX-Supervisor para utilizar protocolos estándar de comunicación que permitan conexiones multi-marca.

Cx-Supervisor es el software de supervisión para sistemas SCADA de la marca Omron. Es suficientemente flexible para trabajar sobre un solo autómata programable o sobre un sistema entero de producción. La programación es en un entorno Windows de forma intuitiva mediante scripts y ventanas. Los scripts pueden ser propios del programa o Java y Visual Basic Scripts.

La comunicación entre el PLC de SIEMENS y el SCADA de OMRON se realizará mediante el protocolo de comunicaciones OPC.

1.8 DESCRIPCIÓN DETALLADA DE LA SOLUCIÓN ADOPTADA

1.8.1 INTRODUCCIÓN

Se trata básicamente de un sistema con una plataforma móvil controlada mediante tres servomotores, cada cual realizando un movimiento en cada eje (X, Y, Z) así pudiendo controlar precisamente su posición, y estacionar correctamente los vehículos.

Nosotros únicamente nos encargaremos de la posición de la plataforma consiguiendo que vaya a la posición requerida para el estacionamiento o retirada de un vehículo.

1.8.1.1 ESTRATEGIA DE CONTROL UTILIZADA

En nuestro proceso de control nos basaremos en la siguiente pirámide de control:

Figura 1.19 Pirámide de control

En nuestro caso usaremos los 3 primeros niveles, que se tratan de los siguientes:

- Primer nivel (sensores, actuadores, hardware): En este nivel se engloban todos los elementos que interactúan directamente con el proceso, ya sea para recibir datos, mediante sensores, o para activar o desactivar actuadores. Los sensores pueden ser analógicos o digitales y podremos realizar mediciones eléctricas, térmicas, mecánicas, etc. Los actuadores también pueden ser analógicos o digitales y podremos encontrarlos de diversos tipos, ya sean neumáticos, eléctricos, hidráulicos, etc.

- Sistemas de control (PLC, DCS e iPC): En este nivel encontramos los controladores, en nuestro caso el PLC que gestionará los actuadores, entradas y salidas.

- Sistemas de supervisión (SCADA): En este nivel encontramos al sistema encargado de acceder a los datos de proceso para poder monitorizarlo y/o controlarlo.

1.8.1.2 PRIMEROS PASOS

Lo primero que haremos será determinar el punto inicial de nuestro sistema que se tratará también de donde estacionará el usuario su vehículo. Como explicaremos más adelante este punto será el intermedio entre las columnas y al nivel de la superficie.

A partir de este punto inicial determinaremos nuestro sistema de ejes. Para nombrar las coordenadas de nuestras plazas de aparcamiento usaremos una matriz unidimensional, en la que descompondremos los números de tal forma como descompondríamos un número de 3 cifras para que cada plaza tenga asignado un número único. Lo hacemos de esta forma para poder tener sincronizada la matriz del programa de CoDeSys con el SCADA de CX – Supervisor.

Como explicación de lo dicho anteriormente, un número de tres cifras lo descompondríamos de la siguiente forma:

$$Vc \cdot Nd \cdot Nu + Vd \cdot Nu + Vu$$

Siendo:
Vc = Valor del número que ocupa la posición de las centenas
Vd = Valor del número que ocupa la posición de las decenas
Vu = Valor del número que ocupa la posición de las unidades
Nd = Número de decenas
Nu = Número de unidades

Trasladado esto a nuestro proyecto nos queda la siguiente fórmula que determina la posición donde guardaremos el valor de cada plaza.

$$(Vp - 1) \cdot Nc \cdot Nf + (Vf - 1) \cdot Nc + (Vc - 1)$$

Siendo:

Vp = Profundidad de la plaza
Vf = Fila de la plaza
Vc = Columna de la plaza
Nf = Número de filas
Nc = Número de columnas

En nuestro caso restamos 1 a los distintos valores debido a que nuestras filas, columnas y profundidad empieza en 1 y no en 0.

Debido a que nuestro número de plazas es de 110, nuestra matriz debería tener ese número de valores, empezando en M[0] y terminando en M[109]. Sin embargo, vamos a realizar un programa más flexible que se pueda adaptar compartiendo cierta estructura similar al nuestro que más adelante detallaremos.

Tomamos como punto inicial el punto intermedio entre las columnas en el eje Z que equivale a la posición 5 de las filas que estará al nivel de la superficie ya que para el usuario será más rápido y cómodo estacionar en la plataforma el vehículo a esa altura y debido a que en nuestro caso no tenemos plazas subterráneas todas las filas de plazas de estacionamiento se encontrarán por encima de este punto. En el eje Y corresponderá a la posición 6 de las columnas y en el eje X corresponderá al valor 2 de la profundidad de la matriz. Siendo por tanto el valor del número de plaza el 104, por tanto, el valor que modificaremos en la matriz será M[104] cuyo valor siempre será 1.

1.8.2 PASOS DE NUESTRO SISTEMA DE CONTROL

1.8.2.1 CONFIGURACIÓN DE LOS SERVOMOTORES EN BSD

Antes de nada, realizaremos la configuración en BSD Configurator de nuestros servomotores. Configuraremos todos los parámetros, menos β, igual para los 3 servomotores.

No	Set item	Set value	Set range	Initial Value
1	Command pulse correction Alpha	7282	1 to 32767	16
2	Command pulse correction Beta	2	1 to 32767	1
3	*Pulse train input form	0	0:Command pulse/Sign 1:Forward/Reverse 2:Tv	1
4	*Rotate Direction/Out pulse pha	1	0:CCW/B-phase advance 1:CW/B-phase advan	0
5	Tuning mode	0	0:Auto tuning 1:Semi auto tuning 2:Manual tunin	0
6	Load inertia ratio	3.3	0.0 to 100.0	5.0
7	Auto tuning gain	10	1 to 20	10
8	Auto forward gain	5	1 to 20	5
9	*Control mode change	0	0:Position 1:Speed 2:Torque 3:Pos./Speed 4:Po:	0
10	*CONT1 signal assignment	1	0:Not assigned 1:RUN 2:RST 3:+OT 4:-OT 5:I	1
11	*CONT2 signal assignment	2	7:Deviation clear 8:External fault input 9:Anti-res	2
12	*CONT3 signal assignment	0	10:Anti-resonant freq. 1 11:INH 12:Alpha select I	0
13	*CONT4 signal assignment	0	14:CSEL 15:FWD 16:REV 17:X1 18:X2 19:AC	0
14	*CONT5 signal assignment	0	22:Reserved for maker (Please refer to parameter	0
15	*OUT1 signal assignment	1	0:Not assigned 1:RDY 2:PSET 3:ALM(contact-/	1
16	*OUT2 signal assignment	2	5:Dynamic braking 6:OT detection 7:Forced stop	2
17	*OUT3 signal assignment	4	9:NZERO 10:Torque limit detection 11:Brake tim	4
18	*OUT4 signal assignment	0	13:Reserved for maker (Please refer to parameter	0
19	*Output pulse count	2048	16 to 32768[pulse]	2048
20	*Z-phase offset	0	0 to 65535[*2pulse]	0
21	Deviation zero width	400	1 to 2000[pulse]	400
22	Deviation over width	20000	10 to 65535[*100pulse]	20000
23	Speed zero width	50	10 to 5000[r/min]	50
24	Judgment time of positioning enc	0.000	0.000 to 1.000[sec]	0.000
25	Maximum current	300	0 to 300[%]	300
26	*Alarm detection at under voltag	1	0:No detection 1:Detect	1
27	*Operation at under voltage	0	0:Rapidly decelerates to stop 1:Free-run	0
28	*Electronic thermal of braking res	0	0:Invalid 1:Valid(for thin form resistor(WSR-***-TD	0
29	Parameter rewriting inhibit	0	0:Rewriting allowed 1:Rewriting inhibit	0
30	*Initial display of keypad panel	0	0 to 21 (Please refer to parameter information for c	0
31	Manual feed speed 1(test runnin	750.0	0.1 to 5000.0[r/min]	100.0
32	Manual feed speed 2	1500.0	0.1 to 5000.0[r/min]	500.0
33	Manual feed speed 3	2250.0	0.1 to 5000.0[r/min]	1000.0
34	Maximum speed	2500.0	0.1 to 5000.0[r/min]	5000.0

Figura 1.20 Parámetros BSD Configurator

Los parámetros más importantes que tenemos que revisar y asegurarnos de que están puestos correctamente para el funcionamiento de nuestro programa son:

Parámetro nº3: Este parámetro debemos tenerlo en valor 0 para que mediante la modificación de un bit por CoDeSys podamos cambiar el sentido de giro de nuestros servomotores.

Parámetro nº9: El valor de este parámetro deberá ser 0 para realizar el control de posición.

Parámetro nº10: Debe tener asignado valor 1 este parámetro para que podamos poner en modo RUN nuestros servomotores mediante CoDeSys.

Tras la asignación de esos parámetros deberemos calcular mediante la fórmula:

$$\frac{\alpha}{\beta} = \frac{131072}{Distancia} \cdot Paso$$

$$\frac{\alpha \cdot 360}{\beta \cdot 2^{17}} = Paso$$

En nuestro caso hemos decidido que nuestro mayor paso, el del eje Z, sea de 20˚ y calcularlo para una β=1 para que reducirlo simplemente nos conlleve cambiar el valor de β.

Por tanto, nuestra α será:

$$\alpha = \frac{131072}{360} \cdot 20 \cong 7282$$

EJE X

El paso de este eje será de 5˚, por lo que:

α = Parámetro nº1 = 7282
β = Parámetro nº2 = 4

EJE Y

En este eje nuestro paso será de 10˚, por tanto:

α = Parámetro nº1 = 7282
β = Parámetro nº2 = 2

EJE Z

Como hemos dicho anteriormente en este eje nuestro paso será de 10˚, por tanto:

α = Parámetro nº1 = 7282
β = Parámetro nº2 = 1

1.8.2.2 FLUJOGRAMA DE NUESTRO PROGRAMA

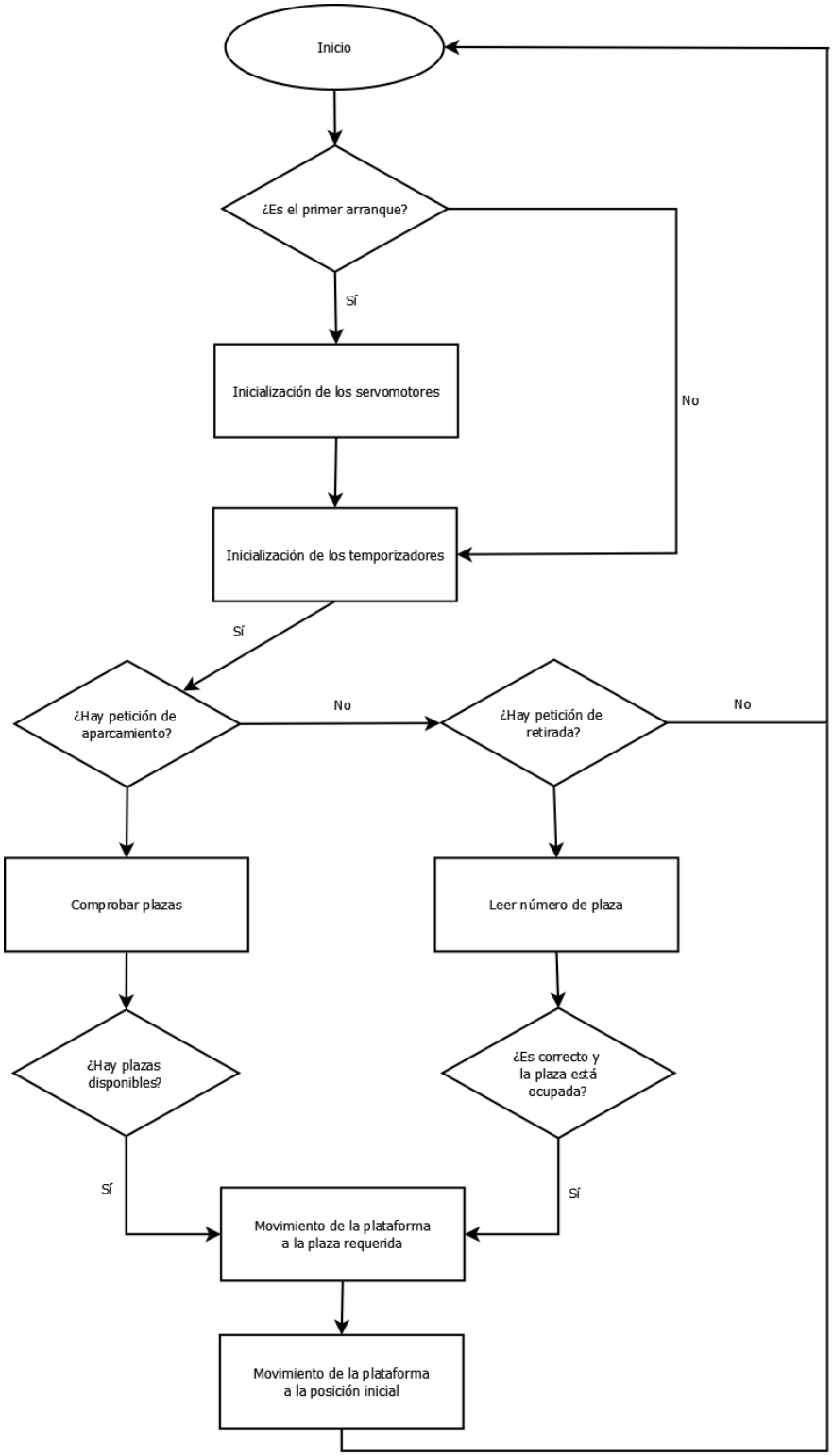


Figura 1.21 Flujograma del programa

1.8.2.3 INICIALIZACIÓN DE LOS SERVOMOTORES

En primer lugar, en CoDeSys, realizaremos una inicialización de los servomotores para asegurarnos de su buen funcionamiento. Esto se llevará a cabo en la primera ejecución de nuestro programa o bien cuando terminemos un movimiento completo de los ejes.

Para realizar la inicialización de los servomotores lo que haremos será en primer lugar ponerlos todos en modo *Run* editando el valor 1.0 del PLC a través del acceso al DC541 con la función DC541_IO.

Tras esto, pondremos los valores de stop de cada eje en 1 y los de start en 0 para que cualquier movimiento que pudiera tener los servomotores fuera del funcionamiento normal pare y puedan arrancar de nuevo con nuestra configuración.

Para ello además de cambiar esos valores deberemos ejecutar la función del frequency out de DC541 para que los motores reciban las instrucciones.

Luego, invertiremos los valores de start y stop de cada eje para que los motores arranquen cuando tengamos que realizar un movimiento de nuestra plataforma.

Por último, cambiaremos los valores de Inicialización y PrimerArranque para que no se ejecute esta secuencia de comandos en cada ciclo, solo cuando sea necesario.
Cambiamos el valor de movX a 0 finalmente para en una función posterior cambiar el movimiento del eje X.

Por tanto, la estructura de nuestra función para inicializar los servomotores será la de la figura 1.22.

```
IF Inicializacion=1 OR PrimerArranque=0 THEN
    miDC541_IO (EN  := TRUE, SLOT := 1, OUT0 := 1);
    stopX:=1;
    stopY:=1;
    stopZ:=1;
    startX:=0;
    startY:=0;
    startZ:=0;
    miDC541_Freq_Y(SLOT := 1, CH:=1,EN_VISU :=TRUE, EN :=marcha, START :=startY, STOP := stopY, FREQ := frecY, PULSE := PulsY);
    miDC541_Freq_Z(SLOT := 1, CH:=7,EN_VISU :=TRUE, EN :=marcha, START :=startZ, STOP := stopZ, FREQ := frecZ, PULSE := PulsZ);
    miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha, START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
    startX:=1;
    startY:=1;
    startZ:=1;
    stopX:=0;
    stopY:=0;
    stopZ:=0;
    Inicializacion:=0;
    PrimerArranque:=1;
    movX:=0;
END_IF
```

Figura 1.22 Función de inicialización de los servomotores

1.8.2.4 INICIALIZACIÓN TEMPORIZADORES

Para el buen funcionamiento de nuestros programas y para evitar que nuestros 3 servomotores se muevan al mismo tiempo, lo que crearía un desajuste en el movimiento de nuestra plataforma, introduciremos 4 temporizadores (TON) que servirán para temporizar los 4 movimientos que realizaremos y explicaremos más adelante.

Aunque se encuentren inicializados los temporizadores, no iniciarán su cuenta hasta que no le demos la orden en otras funciones.

La estructura correspondiente será la de la figura 1.23

```
Wait(IN:=Temp , PT:=T#17s);
Espera(IN:=Tempo , PT:=T#11s*Mult);
Sleep(IN:=Tempor , PT:=T#14s*(Filas-o));

IF Pausa.Q=0 THEN
    Pausa(IN:=Temp2 , PT:=T#17s);
END_IF
```

Figura 1.23 Función de inicialización de los temporizadores

En el último temporizador añadimos una condición debido a pruebas realizadas es la forma que hemos encontrado de que se sincronice con funciones de las que hablaremos más adelante y funcione correctamente.

El segundo y el tercer temporizador tendrá un multiplicador en su variable de tiempo, que hará que sea mayor o menor dependiendo de cuantas plazas se deba desplazar la plataforma.

1.8.2.5 COMPROBAR SI LA PLATAFORMA ESTÁ EN LA POSICIÓN INICIAL

Para comprobar si la plataforma está en nuestra posición utilizaremos una variable BOOL de nombre PosicionInicial, en el que si el valor de la variable es 1 indica que la plataforma está en la posición inicial y si el valor es 0 es que no está en la posición inicial y por lo tanto no está disponible.

Para asegurarnos el buen funcionamiento y la seguridad deberemos añadir la condición de que esté disponible, por lo que en cada petición de un cliente para estacionar o retirar un vehículo se comprobará mediante un condicional IF la condición PosicionInicial = 1 para que no se pueda empezar una maniobra sin estar nuestra plataforma en la posición inicial.

1.8.2.6 COMPROBAR PLAZAS VACÍAS

En primer lugar, mediante un anidamiento de bucles FOR comprobaremos plaza a plaza si la plaza en cuestión está ocupada o no, y mediante una condicional IF dejaremos de comprobar las plazas cuando ya se haya encontrado una vacía. La función tendrá la estructura de la figura 1.24.

```
FOR i:=Filas TO 1 BY -1 DO
    FOR j:=1 TO Columnas DO
        FOR k:=1 TO 2 DO
            IF M[(k-1)*Columnas*Filas + (i-1)*Columnas + (j-1)]=0 AND Aparcado=0 THEN
```

Figura 1.24 Comprobación plazas vacías. Estructura FOR.

1.8.2.7 MOVER PLATAFORMA

En primer lugar, para comprobar si es requerido el movimiento de nuestra plataforma crearemos un condicional IF en el que se ejecute nuestra función de movimiento de plataforma solo cuando las funciones de estacionar o retirar vehículo lo requieran y así evitar que se ejecute el movimiento de la plataforma en cada ciclo de programa.

La estructura del IF será la de la figura 1.25

```
IF Movimientoservo=1 THEN
...
END_IF
```

Figura 1.25 Mover plataforma. Estructura IF 1

Para mover nuestra plataforma deberemos de determinar primero el número de pulsos que necesita el servomotor para moverse una plaza respecto a cada eje y en función de ese número podremos determinar el número de pulsos que se necesita en cada eje para llegar a cualquiera de las plazas. Para ello utilizaremos las siguientes fórmulas:

$$Pulsos_{necesarios} = \frac{Distancia_{deseada}}{Distancia_{\ por\ pulso}}$$

Donde la $Distancia_{por\ pulso}$ nos la dará el reductor. Sin embargo, en nuestro caso vamos a teorizar que en el eje X 1 pulso será equivalente a 1 cm, en el eje Y 1 pulso será equivalente a 2cm y en el eje Z 1 pulso equivaldrá a 4cm.

Dando lugar a los siguientes pulsos necesarios para movernos 1 plaza:

PulsX = $\frac{5500cm}{1}$= 5500 pulsos

PulsY = $\frac{2500cm}{2}$ = 1250 pulsos

PulsZ = $\frac{2500cm}{4}$ = 575 pulsos

Tras haber calculado el número de pulsos ya podremos utilizar la función de la figura 1.26 en CodeSys para darle a nuestros servomotores la frecuencia y los números de pulsos necesarios para mover nuestra plataforma a donde se requiera.

```
miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha, START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
miDC541_Freq_Y(SLOT := 1, CH:=5,EN_VISU :=TRUE, EN :=marcha, START :=startY, STOP := stopY, FREQ := frecY, PULSE := pulsY);
miDC541_Freq_Z(SLOT := 1, CH:=7,EN_VISU :=TRUE, EN :=marcha, START :=startZ, STOP := stopZ, FREQ := frecZ, PULSE := pulsZ );
```

Figura 1.26 Función FREQ_OUT

Donde pulsY, pulsZ serán el número de pulsos necesarios para movernos una plaza en cada eje cuyo valor lo tendremos almacenado en las variables pulsY1 y pulsZ1, multiplicado por el número de plazas que debemos mover la plataforma. Debido a que en X nos moveremos de 1 plaza en 1 plaza, el valor de pulsX equivaldrá directamente al número de pulsos necesarios para movernos una plaza en el eje X.

Usaremos también la función DC541_IO para cambiar los sentidos, cambiando dentro de esta función el valor de OUT2, OUT4 y OUT6. Siendo, en nuestro caso, OUT2 el sentido del eje X, OUT4 el sentido del eje Y y OUT6 el sentido del eje Z.

```
miDC541_IO (EN  := TRUE, SLOT := 1, OUT2 := 0);
miDC541_IO (EN  := TRUE, SLOT := 1, OUT4 := 0);
miDC541_IO (EN  := TRUE, SLOT := 1, OUT6 := 0);
```

Figura 1.27 Función DC541_IO

Con esta base separaremos el movimiento de la plataforma en 2, el primero en el que mediante 4 movimientos con nuestros servomotores, que detallaremos, moveremos la plataforma a la plaza requerida y el segundo mediante el mismo número de movimientos pero invertidos devolveremos la plataforma a la plaza original.

Una vez con esta base vamos a detallar el primer conjunto de movimientos, al que le asignaremos la variable Movimientoservo la cual condicionará si se realiza o no el conjunto de movimientos.

Para empezar, se realizará el movimiento en eje X para mover la plataforma de la posición inicial a la posición intermedia de las dos hileras de plazas. Vendrá determinada por una variable llamada movX que nos servirá para evitar la interferencia de este movimiento en el cuarto de este mismo conjunto, ya que se trata de un movimiento en el mismo eje.

Tampoco iniciaremos la cuenta del temporizador, aunque podríamos, porque lo haremos en la orden de estacionamiento o retorno del vehículo.

Debido a que el sentido para este movimiento siempre será el mismo, la salida 1.2 le asignaremos el valor 1 que en nuestro sistema de ejes equivaldrá a moverse hacia dentro.

Siendo la estructura de nuestro primer movimiento la de la figura 1.28.

```
IF movX=0 THEN
miDC541_IO (EN  := TRUE, SLOT := 1, OUT2 := 1);
miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha, START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
movX:=1;
END_IF
```

Figura 1.28 Mover Plataforma. Movimiento 1.

Cuando termine este movimiento equivaldrá a que el valor de la salida de nuestro temporizador valga 1. Con dicha información procederemos a realizar dos listas de instrucciones mediante IF.

La primera es cambiar el valor de una variable para más adelante, como en la inicialización, devolver a su situación inicial el servo del eje X para evitar problemas en el siguiente movimiento del servomotor. Tendrá la estructura de la figura 1.29.

```
IF Wait.Q =1 THEN
    con:=1;
END_IF
```

Figura 1.29 Mover plataforma. Estructura IF 2.

La segunda se trata del movimiento en el eje Y, bajo la condición de que se tenga que mover de columna, en el que trasladaremos la plataforma a la columna correspondiente mediante un movimiento horizontal. Para ello, lo que haremos será en primer lugar reiniciar el primer temporizador cambiando el valor de la variable de su iniciación a 0 y cambiando a 1 la del segundo temporizador, activaremos este.

Tras esto asignaremos un sentido u otro según el valor almacenado en la variable senY en las funciones de estacionamiento o retirada del vehículo. Por último, ordenaremos el movimiento del servomotor terminando con la estructura de la figura 1.30.

```
IF Wait.Q=1 AND w<>((Columnas+1)/2) THEN
    Temp:=0;
    Tempo:=1;
    IF senY=1 THEN
        miDC541_IO (EN  := TRUE, SLOT := 1, OUT4 := 1);
    ELSE
        miDC541_IO (EN  := TRUE, SLOT := 1, OUT4 := 0);
    END_IF
    miDC541_Freq_Y(SLOT := 1, CH:=1,EN_VISU :=TRUE, EN :=marcha, START :=startY, STOP := stopY, FREQ := frecY, PULSE := PulsY);
END_IF
```

Figura 1.30 Mover Plataforma. Movimiento 2.

Como observamos la condición de que no se mueva la columna es que la columna sea distinta a la mitad del número de columnas +1, esto se debe a que la estructura de nuestro parking tiene un número de columnas impar, ya que esa columna equivale a la céntrica.

El tercer movimiento se realizará en el eje Z mediante movimiento vertical, y tendrá 2 grupo de condiciones: la primera se trata de que tenga que moverse a una fila distinta y la segunda es que si es así o bien se haya ejecutado el movimiento en el eje X e Y o bien que se haya ejecutado en el eje X y en el Y no sea necesario ningún movimiento.

Debido a que el sentido para este movimiento siempre será el mismo, puesto que estamos en el nivel más bajo, la salida 1.6 le asignaremos el valor 0 que en nuestro sistema de ejes equivaldrá a moverse hacia arriba.

Con esto y siguiendo la lógica del movimiento anterior, reiniciando y parando la cuenta del segundo contador e iniciando la del tercero tenemos la estructura de la figura 1.31.

```
IF o<Filas AND Espera.Q=1 OR (o<Filas AND w=(Columnas+1/2) AND Wait.Q=1) THEN
    Tempo:=0;
    Tempor:=1;
    miDC541_IO (EN  := TRUE, SLOT := 1, OUT6 := 0);
    miDC541_Freq_Z(SLOT := 1, CH:=7,EN_VISU :=TRUE, EN :=marcha, START :=startZ, STOP := stopZ, FREQ := frecZ, PULSE := PulsZ);
END_IF
```

Figura 1.31 Mover Plataforma. Movimiento 3

Por último, volveremos a realizar otro movimiento en el eje X que introducirá el vehículo en la plaza correspondiente. La estructura será similar a la del eje Y solo que reiniciando el temporizador 3 y poniendo en marcha el 4. También tendremos que usar el sentido que equivaldrá a la variable senX cuyo valor obtendremos de las funciones de retirada o estacionamiento del vehículo.

Las diferencias en este movimiento estarán en las condiciones, que serán que se hayan realizado ya el movimiento en el eje X, y en el eje Y y Z si ha sido necesario, también se contempla en las condiciones el caso de que en solo uno de esos ejes haya sido necesario realizar un desplazamiento. La otra diferencia estará en que usando la variable con del primer movimiento realizaremos una inicialización del motor para asegurarnos del buen funcionamiento del servo en este movimiento.

Finalmente nos queda la estructura de la figura 1.32.

```
IF Sleep.Q=1 OR (Espera.Q=1 AND o=Filas) OR (Wait.Q=1 AND o=Filas AND w=((Columnas+1)/2) )THEN
    IF con=1 THEN
        stopX:=1;
        startX:=0;
        miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha, START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
        stopX:=0;
        startX:=1;
        con:=0;
    END_IF
    Tempo:=0;
    Tempor:=0;
    Temp2:=1;
    IF senX=1 THEN
    miDC541_IO (EN  := TRUE, SLOT := 1, OUT2 := 1);
    ELSE
    miDC541_IO (EN  := TRUE, SLOT := 1, OUT2 := 0);
    END_IF
    miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha, START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
END_IF
```

Figura 1.32 Mover Plataforma. Movimiento 4.

Tras estos 4 movimientos realizamos otra vez la función de inicialización de los servomotores y pasamos a la de retorno cambiando el valor de las variables que condicionan el funcionamiento de ambas. Por tanto, la estructura final de nuestra función de movimiento de la plataforma a la plaza requerida es la de la figura 1.33.

```
IF Movimientoservo=1 THEN
    IF movX=0 THEN
    miDC541_IO (EN  := TRUE, SLOT := 1, OUT2 := 1);
    miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha, START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
    movX:=1;
    END_IF
    IF Wait.Q =1 THEN
        con:=1;
    END_IF
    IF Wait.Q=1 AND w<>((Columnas+1)/2) THEN
        Temp:=0;
        Tempo:=1;
        IF senY=1 THEN
            miDC541_IO (EN  := TRUE, SLOT := 1, OUT4 := 1);
        ELSE
            miDC541_IO (EN  := TRUE, SLOT := 1, OUT4 := 0);
        END_IF
        miDC541_Freq_Y(SLOT := 1, CH:=1,EN_VISU :=TRUE, EN :=marcha, START :=startY, STOP := stopY, FREQ := frecY, PULSE := PulsY);
    END_IF
    IF o<Filas AND Espera.Q=1 OR (o<Filas AND w=(Columnas+1/2) AND Wait.Q=1) THEN
        Tempo:=0;
        Tempor:=1;
        miDC541_IO (EN  := TRUE, SLOT := 1, OUT6 := 0);
        miDC541_Freq_Z(SLOT := 1, CH:=7,EN_VISU :=TRUE, EN :=marcha, START :=startZ, STOP := stopZ, FREQ := frecZ, PULSE := PulsZ);
    END_IF
    IF Sleep.Q=1 OR (Espera.Q=1 AND o=Filas) OR (Wait.Q=1 AND o=Filas AND w=((Columnas+1)/2) )THEN
        IF con=1 THEN
            stopX:=1;
            startX:=0;
            miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha, START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
            stopX:=0;
            startX:=1;
            con:=0;
        END_IF
        Tempo:=0;
        Tempor:=0;
        Temp2:=1;
        IF senX=1 THEN
        miDC541_IO (EN  := TRUE, SLOT := 1, OUT2 := 1);
        ELSE
        miDC541_IO (EN  := TRUE, SLOT := 1, OUT2 := 0);
        END_IF
        miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha, START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
    END_IF
    IF Pausa.Q=1 THEN
        Temp2:=0;
        Pausa(IN:=Temp2 , PT:=T#17s);
        Retorno:=1;
        Inicializacion:=1;
        Movimientoservo:=0;
    END_IF
END_IF
```

Figura 1.33 Función Mover Plataforma

1.8.2.8 RETORNO

La función de retorno es similar a la anterior invirtiendo los sentidos. El único cambio significativo es que el sentido del primer movimiento dependerá del sentido del último de la anterior función, ya que será el sentido contrario, y que el sentido del último movimiento de esta función siempre será el mismo siendo 1.2 = 0.

Añadimos también la activación del primer temporizador al principio y una variable cuenta para que no se reinicie en cada ciclo y añadimos también el que cuando termina nuestra función el valor de PosicionInicial cambia a 1.

Por tanto, nos queda la estructura de la función será equivalente a la de la figura 1.34.

```
IF Retorno=1 THEN
    IF cuenta=0 THEN
        Temp:=1;
        cuenta:=1;
    END_IF
    IF movX=0 THEN
        IF senX=1 THEN
            miDC541_IO (EN  := TRUE, SLOT := 1, OUT2 := 0);
        ELSE
            miDC541_IO (EN  := TRUE, SLOT := 1, OUT2 := 1);
        END_IF
        miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha, START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
        movX:=1;
    END_IF
    IF Wait.Q =1 THEN
        con:=1;
    END_IF
    IF Wait.Q=1 AND w<>((Columnas+1)/2) THEN
        Temp:=0;
        Tempo:=1;
        IF senY=0 THEN
            miDC541_IO (EN  := TRUE, SLOT := 1, OUT4 := 1);
        ELSE
            miDC541_IO (EN  := TRUE, SLOT := 1, OUT4 := 0);
        END_IF
        miDC541_Freq_Y(SLOT := 1, CH:=1,EN_VISU :=TRUE, EN :=marcha, START :=startY, STOP := stopY, FREQ := frecY, PULSE := PulsY);
    END_IF
    IF o<Filas AND Espera.Q=1 OR (o<Filas AND w=((Columnas+1)/2) AND Wait.Q=1) THEN
        miDC541_IO (EN  := TRUE, SLOT := 1, OUT6 := 1);
        miDC541_Freq_Z(SLOT := 1, CH:=7,EN_VISU :=TRUE, EN :=marcha, START :=startZ, STOP := stopZ, FREQ := frecZ, PULSE := PulsZ);
        Tempo:=0;
        Tempor:=1;
    END_IF
    IF Sleep.Q=1 OR (Espera.Q=1 AND o=Filas) OR (Wait.Q=1 AND o=Filas AND w=((Columnas+1)/2)) THEN
        IF con=1 THEN
            stopX:=1;
            startX:=0;
            miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha, START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
            stopX:=0;
            startX:=1;
            con:=0;
        END_IF
        Tempo:=0;
        Tempor:=0;
        Temp2:=1;
        miDC541_IO (EN  := TRUE, SLOT := 1, OUT2 := 0);
        miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha, START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
    END_IF
    IF Pausa.Q=1 THEN
        Temp2:=0;
        Pausa(IN:=Temp2 , PT:=T#17s);
        Retorno:=0;
        PosicionInicial:=1;
        Inicializacion:=1;
        cuenta:=0;
    END_IF
```

Figura 1.34 Función Retorno

1.8.2.9 ESTACIONAR VEHÍCULO

Una vez ya con la función necesaria del paso anterior y paso donde comprobábamos las plazas vacías podemos realizar una función utilizando las funciones anteriores de forma que podamos estacionar el vehículo en cualquier plaza que se encuentre vacía. Además, para ello cuando realicemos la acción de estacionar el vehículo almacenaremos la información en una variable ARRAY OF BOOL cuyos elementos corresponden a las diferentes plazas, cada elemento puede valer 1 o 0, equivaliendo 1 a que está ocupada la plaza y 0 a que está libre. Mediante los tres bucles FOR de la primera función iremos variando las variables i, j, k que serán las que definan en qué plaza vamos a estacionar el vehículo, siendo i las filas, j las columnas y k la profundidad.

Para determinar el sentido del movimiento en los ejes Y y X haremos dos comprobaciones mediante condicionales IF ELSIF, en el que determinaremos que, si j es menor que la mitad del número de columnas + 1, como nos moveremos hacia la izquierda, el valor del sentido de Y, senY, será 0, que es lo que tomamos como positivo, y si j es mayor que la mitad del número de columnas + 1, el valor de senY será 1. Respecto al eje X, si k=1 significará que la plataforma se moverá hacia fuera, lo que consideramos movimiento positivo en el eje X, por lo que senX tendrá de valor 0, mientras que si k=2 la plataforma se moverá hacia dentro y por lo tanto tomaremos el valor de senX como 1.

Añadiremos también dos variables, o y w para guardar el valor de i y j para las comprobaciones en la función de mover la plataforma y usaremos esa variable o y otra llamada Mult para ajustar el tiempo de nuestros temporizadores y el número de pulsos de nuestros servomotores según lo requiera el movimiento.

Por último, en esta función, utilizaremos una variable para que una vez aparcado el vehículo ya no se cumpla la condición IF y no intente aparcar de nuevo el vehículo cuando realmente ya lo ha hecho. A esta variable la llamaremos Aparcado.

Finalmente, la función nos queda de la forma de la estructura 1.35.

```
IF PeticionAparcamiento=1 AND PosicionInicial=1 AND marcha=1 AND Movimientoservo=0THEN
    PeticionAparcamiento:=0;
    Aparcado:=0;
    FOR i:=Filas TO 1 BY -1 DO
        FOR j:=1 TO Columnas DO
            FOR k:=1 TO 2 DO
                IF M[(k-1)*Columnas*Filas + (i-1)*Columnas + (j-1)]=0 AND Aparcado=0 THEN
                    senZ:=0;
                    PosicionInicial:=0;
                    Movimientoservo:=1;
                    Temp:=1;
                    Posicion:=(k-1)*11*5 + (i-1)*11 + (j-1);
                    o:=i;
                    w:=j;
                    IF w<((Columnas+1)/2) THEN
                    Mult:=((Columnas+1)/2)-w;
                    ELSIF w>((Columnas+1)/2) THEN
                    Mult:=w-((Columnas+1)/2);
                    END_IF
                    pulsZ:=pulsZ1*(Filas-i);
                    pulsY:=pulsY1*(Mult);
                    IF j<((Columnas/2)+1) THEN
                        senY:=0;
                    ELSIF ((Columnas/2)+1)>6 THEN
                        senY:=1;
                    END_IF;
                        IF k=1 THEN
                        senX:=0;
                    ELSIF k=2 THEN
                        senX:=1;
                    END_IF;
                    M[Posicion]:=1;
                    Aparcado:=1;
                END_IF;
            END_FOR;
        END_FOR;
    END_FOR;
END_IF;
```

Figura 1.35 Función Estacionar Vehículo

1.8.2.10 RETIRAR VEHÍCULO

A la hora de retirar un vehículo, simularemos que el usuario tiene un ticket con un código que se le proporcionará al estacionarlo y ese valor constará de tres cifras, la primera nos dará el valor de la fila (variable f) en la que está estacionado el vehículo, la segunda proporcionará el valor de la columna (variable c) y la tercera el valor de la profundidad (variable p).

Una vez teniendo los valores de f, c y p, comprobamos que haya un vehículo ahí con un condicional IF, y de ser así calculamos igual que en los otros apartados, el número de pulsos necesario para mover la plataforma a la posición donde está estacionado el vehículo y traerlo de vuelta.

Para estimar el sentido en el que nos moveremos en los ejes Y y X al igual que en el apartado de estacionar vehículo haremos dos condicionales IF, uno para cada eje, la única diferencia respecto al apartado mencionado es que en lugar de j comprobaremos el valor de c, y en lugar de k comprobaremos el valor de p.

Con todo eso la función nos queda con la estructura de la figura 1.36.

```
IF PeticionRetirada=1 AND PeticionAparcamiento=0 AND PosicionInicial=1 AND Movimientoservo=0  THEN
    PeticionRetirada:=0;
    IF M[(p-1)*Columnas*Filas + (f-1)*Columnas + (c-1)]=1 AND (f=5 AND c=6 AND p=1)=FALSE THEN
        pulsZ:=pulsZ1*(Filas-f);
        o:=f;
        w:=c;
        IF w<6 THEN
        Mult:=6-w;
        ELSIF w>6 THEN
        Mult:=w-6;
        END_IF
        pulsY:=pulsY1*(Mult);
        senZ:=0;
        Movimientoservo:=1;
        Temp:=1;
        PosicionInicial:=0;
        IF c<6 THEN
            senY:=0;
        ELSIF c>6 THEN
            senY:=1;
        END_IF;
        IF p=1 THEN
            senX:=0;
        ELSIF p=2 THEN
            senX:=1;
        END_IF;
        M[(p-1)*Columnas*Filas + (f-1)*Columnas + (c-1)]:=0;
    END_IF;
END_IF;
```

Figura 1.36 Función Retirar Vehículo

1.8.2.11 CAMBIO DE NÚMERO DE PLAZAS

Para hacer nuestro programa más flexible introducimos las variables filas y columnas y realizamos un pequeño SCADA en CoDeSys donde poder cambiar el valor de estas y de los pulsos y frecuencia de los 3 servomotores. Esto nos servirá para para llevar el programa a otros parkings con una configuración similar a la nuestra, es decir, con la plaza inicial en el nivel más bajo y centrada, y con un número impar de columnas.

Si hubiera más cambios deberíamos trabajar en las distintas líneas de código de CoDeSys, que es algo menos intuitivo, pero realmente no demasiado complicado.

Nuestra SCADA tendrá la apariencia de la figura 1.37.

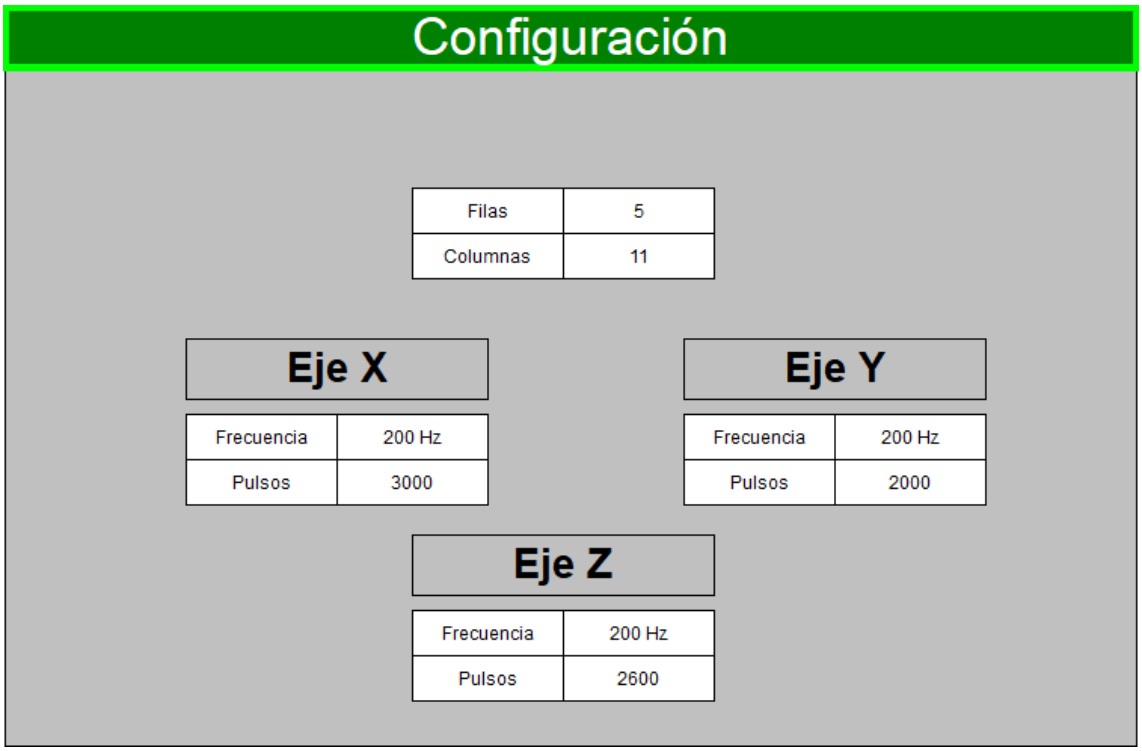


Figura 1.37 Diseño SCADA de configuración

1.8.2.12 LISTA DE VARIABLES UTILIZADAS

En este apartado recogeremos las distintas variables utilizadas en el programa, el tipo de variable y una breve descripción:

Variable	Tipo	Descripción
i	INT	Variable que cambiará de valor por el primer bucle FOR y equivaldrá al número de la columna.
j	INT	Variable que cambiará de valor por el segundo bucle FOR y equivaldrá al número de la fila.
k	INT	Variable que cambiará de valor por el tercer bucle FOR y equivaldrá a la profundidad.
f	INT	Variable que vendrá determinada por el código del usuario a la hora de retirar su vehículo y equivaldrá al número de la fila.
c	INT	Variable que vendrá determinada por el código del usuario a la hora de retirar su vehículo y equivaldrá al número de la columna.
p	INT	Variable que vendrá determinada por el código del usuario a la hora de retirar su vehículo y equivaldrá al número de la profundidad.
o	INT	Variable que almacenará el valor de i o de f y que nos servirá para enviar el número de pulsos adecuado en el eje Y
w	INT	Variable que almacenará el valor de j o de c y que nos servirá para enviar el número de pulsos adecuado en el eje Z
M	ARRAY [1..11, 1..5, 1..2] OF BOOL	Matriz cuyos elementos equivalen a cada una de las plazas y en la que se almacenarán valores que determinarán si están ocupadas o no
PosicionInicial	BOOL	Variable con la que controlaremos sí la plataforma está en la posición inicial.
PeticionAparacamiento	BOOL	Variable cuyo valor será 1 cuando un usuario solicite el estacionamiento del vehículo

PeticionRetirada	BOOL	Variable cuyo valor será 1 cuando un usuario solicite la retirada del vehículo
Aparcado	BOOL	Variable que nos indicará si el vehículo ya se ha estacionado
Marcha	BOOL	Variable que determinará la marcha de los 3 servomotores
senX	BOOL	Variable que determinará el sentido del movimiento del servomotor del eje X. 0 = sentido positivo. 1= sentido negativo
senY	BOOL	Variable que determinará el sentido del movimiento del servomotor del eje Y. 0 = sentido positivo. 1= sentido negativo
senZ	BOOL	Variable que determinará el sentido del movimiento del servomotor del eje Z. 0 = sentido positivo. 1= sentido negativo
FreqX	LREAL	Frecuencia de funcionamiento del servomotor del eje X. Equivale a la frecuencia de nuestra corriente 50Hz.
FreqY	LREAL	Frecuencia de funcionamiento del servomotor del eje Y. Equivale a la frecuencia de nuestra corriente 50Hz.
FreqZ	LREAL	Frecuencia de funcionamiento del servomotor del eje Z. Equivale a la frecuencia de nuestra corriente 50Hz.
PulsX	DWORD	Número de pulsos que le enviaremos al servo del eje X para mover la plataforma a donde se requiera.
PulsY	DWORD	Número de pulsos que le enviaremos al servo del eje Y para mover la plataforma a donde se requiera.
PulsZ	DWORD	Número de pulsos que le enviaremos al servo del eje Z para mover la plataforma a donde se requiera.
PulsY1	DWORD	Número de pulsos que se han de envíar para mover la plataforma en el eje Y la distancia equivalente a una plaza de estacionamiento.
PulsZ1	DWORD	Número de pulsos que se han de envíar para mover la plataforma en el eje Z la distancia equivalente a una plaza de estacionamiento.

Mult	UINT	Variable sin signo que equivaldrá al número de plazas que se tiene que mover la plataforma en el eje Y.
startX	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje X.
stopY	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje Y.
StartZ	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje Z.
stopX	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje X.
startY	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje Y.
stopZ	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje Z.
Temp	BOOL	Variable que sirve para activar el temporizador Wait.
Tempo	BOOL	Variable que sirve para activar el temporizador Espera.
Tempor	BOOL	Variable que sirve para activar el temporizador Sleep.
Temp2	BOOL	Variable que sirve para activar el temporizador Pausa.
Posición	INT	Variable que sirve para almacenar el valor de la plaza requerida.
Filas	INT	Variable con la que podemos variar el número de filas del parking si así se requiriera a través del SCADA.
Columnas	INT	Variable con la que podemos variar el número de columnas del parking si así se requiriera a través del SCADA.
Inicialización	BOOL	Variable que sirve para ejecutar la inicialización de los servomotores.
PrimerArranque	BOOL	Variable que sirve para ejecutar la inicialización de los servomotores la primera vez que ejecutamos el programa
movX	BOOL	Variable que evitará la repetición del primer movimiento en el eje X.
con	BOOL	Variable que inicializará los servomotores entre el primer y el segundo movimiento en el eje X.

Tabla 1.1 Lista de variables

1.9 CONCLUSIONES

Gracias a la realización del parking automático se obtienen algunas ventajas respecto a un parking convencional. Estas ventajas son tanto a nivel del cliente como del propietario del parking. A nivel de cliente la ventaja principal es la comodidad ya que no se tienen que ocupar de buscar la plaza vacía para estacionar el vehículo ni de buscar el vehículo para retirarlo. A nivel tanto de cliente como de propietario, como solo el personal podrá acceder al interior del parking, las instalaciones serán mucho más seguras.

La programación de la plataforma de desplazamiento se ha llevado a cabo con el objetivo de obtener un control automático sobre los tres ejes de dicha plataforma, así como una monitorización y un control del estado de las plazas del parking.

La programación se ha realizado de forma que en caso de querer utilizarla en un parking de características similares pero distinto número de plazas los cambios sean mínimos y de esa manera poder rentabilizar el trabajo y no tener que realizar el proyecto desde cero en el caso de querer utilizarlo en más parkings. En parkings con más cambios estructurales respecto al nuestro requerirán algunos cambios más, pero se podrá reutilizar gran parte del proyecto y no tener que comenzar de cero.

El lenguaje seleccionado para llevar a cabo la programación ha sido el texto estructurado (ST). Se ha seleccionado este lenguaje ya que es muy habitual ver este lenguaje en la industria y con él se pueden realizar acciones complejas con diferentes variables y datos de una manera más rápida.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

CONTROL DE POSICIÓN DE UN PARKING AUTOMÁTICO

2. PLIEGO DE CONDICIONES

SEPTIEMBRE DE 2019

GUILLEM REGO MÁÑEZ
RUBÉN PUCHE PANADERO Y ÁNGEL SAPENA BAÑÓ

Tabla de contenido

2.1 CONDICIONES GENERALES

2.1.1 OBJETIVO DEL PLIEGO DE CONDICIONES

El objetivo del pliego de condiciones es definir las condiciones y cláusulas, económicas, legales que son de carácter obligatorio y se deben aceptar en un contrato de servicios.
Este documento abarca cuatro tipos básicos de condiciones

- Condiciones técnicas: hacen referencia a los trabajos que hay que realizar, las características y calidad de los materiales, cuidados especiales y detalles concretos a tener presente durante la ejecución, y a los controles y ensayos de calidad preceptivos.

- Condiciones facultativas: hacen referencia a las garantías, la formación de precios, las formas de abono y las indemnizaciones por incumplimiento.

- Condiciones económicas: hacen referencia a las garantías, la formación de precios, las formas de abono y las indemnizaciones por incumplimiento.

- Condiciones legales: hacen referencia al perfil de contratista, la forma de adjudicación, el tipo de contrato, la obligatoriedad de suscripción de seguros de responsabilidad civil y otros asuntos relacionados.

Se realizará una descripción detallada, con todas estas condiciones, de la normativa legal a la que está sujeta el proyecto y la seguridad y calidad tanto del proceso de montaje como la ejecución del mismo.
Suponiendo que, durante la instalación, montaje, puesta a punto o utilización del sistema apareciera algún contratiempo que no esté reflejado en el documento, es imprescindible que se consulte con el proyectista para la solución del mismo.
Debido a que este es un proyecto académico con el objetivo de obtener el título de graduado en ingeniería eléctrica, por parte del proyectista, muchos de los siguientes apartados no tienen sentido ya que la fabricación del mismo no se realizará. Sin embargo, para la redacción del documento se supone y se explica las normas que se deberían cumplir en el caso de que el proyecto se llevara a cabo.

2.1.2 NORMATIVA VIGENTE

El sistema de control del presente proyecto está directamente conectado a la red de corriente alterna (220V, 50Hz), por este motivo, todos sus elementos, componentes y aparatos se rigen según las normas del Reglamento Electrotécnico de Baja Tensión (RBT) y sus Instrucciones Complementarias. Además, se deben tener en cuenta las normas UNE y DIN.

Estas normas son las siguientes:

- ITC-BT-18: Instalaciones de puesta a tierra
- ITC-BT-19: Instalaciones interiores o receptoras. Prescripciones generales.
- ITC-BT-20: Instalaciones interiores o receptoras. Sistemas de instalación.
- ITC-BT-22: Instalaciones interiores o receptoras. Protección contra sobreintensidades.
- ITC-BT-23: Instalaciones interiores o receptoras. Protección contra sobretensiones.
- ITC-BT-24: Instalaciones interiores o receptoras. Protección contra contactos directos e indirectos.
- ITC-BT-51: Instalaciones de sistemas de automatización, gestión técnica de la energía y seguridad.
- ITC-BT-43: Instalaciones de receptores. Prescripciones generales.
- ITC-BT-47: Instalación de receptores. Motores.
- PNE-prEN IEC 62368-1: Equipos de audio y vídeo, de tecnología de la información y la comunicación. Parte1: Requisitos de seguridad.

2.2 CONDICIONES A SATISFACER POR LOS COMPONENTES

2.2.1 COMPONENTES PASIVOS

2.2.1.1 CONDUCTORES

Con el fin de evitar cortocircuitos, derivaciones a tierra y proteger a los usuarios, todos los conductores utilizados en el sistema de control deberán estar aislados mediante PVC. Estos conductores pueden ser de cobre o aluminio en función de la disponibilidad, la sección de cada uno de ellos se deberá adaptar a la potencia que circule por ese conductor.
Los cables que soporten tensiones de 24V provenientes de la fuente de alimentación del autómata se conectarán mediante cables de 0,5 mm^2 de sección.
Estos cables podrán ser independientes o formar parte de mangueras.
Además, todos los elementos utilizados y a los que se hace referencia en este apartado deben cumplir el RD 2267/2004: Reglamento de seguridad contra incendios en los establecimientos industriales, por el que los cables deberán ser no propagadores de incendio y con baja emisión de humo y opacidad reducida.

2.2.3 ELEMENTOS ELECTRÓNICOS ADICIONALES

2.2.3.1 INTERRUPTORES Y PULSADORES

Todos los interruptores y pulsadores que se utilicen deberán cumplir con las normas UNE 20 353 y UNE 20 378.

2.2.3.2 PULSADORES DE EMERGENCIA

Estos elementos serán pulsadores en forma de seta de color rojo. El pulsador de emergencia siempre deberá estar colocado en un lugar accesible rápidamente desde la posición del usuario.
Al tratarse de un elemento de seguridad los pulsadores de emergencia siempre serán normalmente cerrados.
Además, todos los pulsadores de emergencia deberán cumplir con la norma EN 408 para la seguridad en maquinaria.

2.2.4 ORDENADOR PERSONAL

El ordenador utilizado deberá contar con una CPU, un monitor, un ratón y un teclado.
Debido a su utilización en el ámbito industrial se deberán tener en cuenta las condiciones ambientales en las que dicho ordenador deberá trabajar como son la temperatura, la humedad, la suciedad, elementos corrosivos, etc.
La temperatura de trabajo de este elemento estará comprendida entre los 0ºC y los 60ºC.
El PC utilizado deberá tener las siguientes características técnicas como mínimo:

- Microprocesador Intel Core i3 520 a 2.2 GHz.
- Memoria RAM de 4096 Mb.
- Espacio libre en el disco duro de 50Gb.
- Sistema operativo Windows 10.

2.2.5 EL AUTÓMATA PROGRAMABLE

El autómata programable estará alimentado por una fuente de alimentación de corriente continua de 24V conectada a la red. Todos los módulos que componen el autómata estarán unidos en el rack correspondiente y este se conectará a tierra.
El autómata deberá cumplir la normativa DIN VDE 0160.

2.2.5.1 EL SOFTWARE

Para la realización, compilación y transferencia de programas al autómata se utilizará el software CoDeSys versión 2.3 (o superior) de 3S-Smart. Para la programación del Scada se utilizará el software CX-Supervisor de OMRON.
Si el autómata o Scada escogidos fueran de otra marca, se utilizará el software correspondiente de dicha marca previa transformación del código del proyecto.

2.3 CONDICIONES CONSTRUCTIVAS

2.3.1 MONTAJE ELÉCTRICO

2.3.1.1 PRUEBAS DE FUNCIONAMIENTO

Antes de conectar a la red todos los elementos que componen el proyecto se realizarán unos ensayos para verificar que todo está colocado y conectado correctamente. A continuación, se van a explicar estos ensayos.
Todas estas pruebas se deben hacer teniendo en cuenta la normativa expuesta en el RBT en la ITC-BT-05: verificaciones e inspecciones.

2.3.1.2 REVISION VISUAL Y DE CONTINUIDAD

Esta revisión consiste en verificar que todos los componentes, elementos y sensores están bien conectados, y sus conexiones y cableado está bien sujeto y en su sitio correspondiente.
También se verificarán como se ha explicado anteriormente todas las soldaduras.

2.3.1.3 PRUEBAS EN TENSIÓN

Estas pruebas consisten en conectar el proceso a la red de alimentación y comprobar que los dispositivos de seguridad funcionan correctamente.
Una vez comprobados los elementos de seguridad comprobaremos que el resto de componentes están en funcionamiento y realizan su función correctamente.

2.3.1.4 PRUEBA FINAL

Se realizará una prueba de movimiento de los ejes por separado…

Una vez asegurado el desplazamiento correcto de cada uno de los ejes se pasará a comprobar que el programa funciona correctamente.

2.3.2 CONTROL DE CALIDAD

Como todos los elementos utilizados en la realización del proceso son elementos comerciales, la tarea de verificar su calidad es del fabricante y por lo tanto no es de la competencia del contratista ni del proyectista.
Pero de todos modos en el momento del montaje del sistema se deberá comprobar que todos los elementos realizan su función correctamente para evitar que una mala función causara daños en otros elementos.
Una vez montados todos los componentes se comprobará que no haya cortocircuitos entre ellos ni derivaciones a tierra.
Se deberá verificar que la tensión de salida de la fuente de alimentación del autómata es la adecuada, en este caso 24V.
Se debe comprobar que todo el sistema funciona correctamente y sobre todo que los elementos de seguridad funcionan.

2.4 CONDICIONES DE EJECUCIÓN

Para el montaje del prototipo objeto de este proyecto la ejecución constará en asegurarse de que se dispone de los elementos materiales y herramientas necesarias y montarlos en el sistema siempre que cumplan los ensayos y verificaciones de los apartados anteriores.
Si se deseara realizar el prototipo descrito en este proyecto en serie se debería revisar el proceso de ejecución y montaje del mismo para mejorar la calidad y la productividad del mismo.

2.4.1 COMPRA DEL MATERIAL

Para comprar los materiales se deben analizar todos los proveedores de los que se disponga, analizando y comparando sus precios y tiempos de espera.

En el caso del prototipo realizado en el proyecto, al ser un proyecto académico se intentará gastar lo mínimo, por lo tanto, se busca comprar los elementos al mínimo precio.
En el caso de tener que comercializar el prototipo se deberá estudiar exhaustivamente el mercado, el precio, los tiempos de espera, la cantidad de los lotes de cada proveedor, las posibles ofertas por comprar en grandes cantidades, etc.
Cabe destacar la gran importancia que tienen los tiempos de espera y los retrasos en la recepción del material que puede hacer que un producto que sea más caro resulte más rentable si se recibe a tiempo, ya que un retraso haría paralizar el proceso de montaje. Para evitar estos problemas es imprescindible tener una buena estrategia comercial tanto interior como con los proveedores y disponer del stock necesario.

2.5 CONDICIONES ECONÓMICAS

2.5.1 MEJORAS DEL PROYECTO INICIAL

Como ya se ha dicho muchas veces, la finalidad de este proyecto es académica y por tanto no existirá ninguna necesidad de mejorar el proceso.
En el caso de su fabricación para la comercialización, cualquier intento de mejora o perfeccionamiento realizado por el fabricante deberá ser consultado con el proyectista antes de ser realizada. En el caso de que el proyectista estudiara la mejora y considerara que es técnica y legalmente valida y resultara oportuno introducirla en el sistema, se deberían renegociar las condiciones del contrato.

2.5.2 PAGOS DE LOS TRABAJOS

En este apartado del pliego de condiciones se establecen las condiciones a seguir por el contratista, el proyectista y el contratante.

- Si el pago se produce entre los cinco días naturales posteriores a la recepción definitiva del prototipo, inclusive con antelación a los mismos, el importe no sufrirá recargo alguno.
- Cuando el pago se efectúe entre seis y treinta días naturales después de la recepción del producto definitivo por parte del contratante, el producto sufrirá un recargo del 2% sobre el precio inicial previsto.
- Si se realiza el pago entre los treinta y uno y sesenta días posteriores a la recepción definitiva del producto, este sufrirá un recargo del 4% sobre el importe establecido.
- Para un aplazamiento del pago entre los sesenta y uno y noventa días tras la recepción del producto definitivo, éste, sufrirá un recargo del 6,5% sobre el precio final.

- Si se retrasa el pago entre los noventa y uno y trescientos días naturales después de la entrega del producto, el precio del producto sufrirá un recargo del 30% sobre el presupuesto inicial.
- En caso de que el contratante optara por pagar a plazos, debe comunicarlo con anterioridad y establecerse a priori el número de plazos, y el intervalo de tiempo entre cada pago fraccionado, sufriendo cada plazo un recargo en función del tiempo transcurrido y de acuerdo con los puntos anteriores.
- En caso del incumplimiento del pago en el término de tiempo escogido por el cliente, el fabricante tendrá derecho a demandarle ante los tribunales.

2.6 CONDICIONES LEGALES

2.6.1 CONTRATO

El contrato deberá realizarse por escrito, cumplir todos los requisitos legales, y estar firmado por todas las partes implicadas.
En este contrato se deberá especificar tanto el precio inicial de fabricación del producto como su coste unitario.
A este precio se le añadirá, si es necesario, las tarifas impuestas en el apartado anterior.
Deberán incluirse también todas aquellas cláusulas que deban cumplir cada una de las partes.

2.6.2 ADJUDICACIÓN DE LA CONTRATA

Los trabajos de fabricación, montaje, instalación, programación y comprobación de los dispositivos que componen el sistema se adjudicarán siguiendo los criterios que considere oportunos la empresa contratante.

2.6.3 ARBITRAJE Y JURISDICCIÓN

Si en el caso de producirse algún desentendimiento entre las dos partes y éstas no se pusieran por ellas mismo de acuerdo, se nombraría a un técnico por cada una de las partes para que llegaran a un acuerdo entre ellos.
Si este acuerdo no se lograra de este modo, se recurriría a los tribunales de justicia para que aplicaran la jurisdicción.

2.6.4 IMPUESTOS

Se exigirá a la contrata que esté al corriente de los pagos de impuestos, tasas y contribuciones necesarias para el desarrollo de sus actividades empresariales.
Para la comercialización del producto, se añadirá un impuesto sobre el precio del mismo correspondiente al impuesto sobre el valor añadido (I.V.A.) que está estipulado actualmente en un 21%.
En el caso de que este impuesto fuera modificado en un futuro por la administración del estado, se deberá adaptar este impuesto para cumplir con esa modificación.

2.6.5 RESCISIÓN DEL CONTRATO

En el supuesto caso de que se produjera un retraso excesivo en el tiempo de entrega del producto, sin previo aviso del contratista al contratante, el contratante podrá rescindir el contrato.

2.7 CONDICIONES FACULTATIVAS

El objetivo de este proyecto es controlar el funcionamiento de 3 servomotores mediante las funciones implementadas en el presente proyecto a través de un autómata.
En el caso de que el proyecto se implantara dentro de un sistema industrial para su comercialización se deberían recalcular y reforzar las estructuras, así como adaptarlas al resto del proceso.

2.8 DERECHOS Y DEBERES DEL CONTRATISTA

Debe conocer y cumplir todas las leyes referentes a su actividad empresarial.
Los permisos obligatorios para la realización de este proyecto se deberán obtener por parte del contratante y no del contratista.
Debe conocer las especificaciones técnicas y normas de seguridad que deben cumplir todos los elementos del proyecto. Para ello deberá cumplir con la normativa establecida en el Reglamento Electrotécnico de Baja Tensión.
Deberá realizar las pruebas necesarias para asegurarse de que todos los elementos y el sistema en general funcione correctamente ofreciendo una buena calidad del proceso finalizado.
El contratante no podrá reclamar por retrasos producidos en el proceso de fabricación por causas justificadas ajenas al contratista. En el caso de que estos retrasos no se

justifiquen, el contratante deberá abonar un importe del 10% del importe de fabricación.
Los elementos fabricados cumplirán con los requisitos especificados en el proyecto y solo se podrán modificar con la aprobación del proyectista.
El contratante deberá facilitar al contratista todos los elementos e información necesaria para una realización eficiente y rápida del proceso de fabricación. Además, deberá entregarle por escrito las especificaciones del proyecto y los planos de los elementos y esquemas que componen dicho proyecto.
El trabajo del contratista finaliza cuando después de comprobar que el elemento fabricado funciona correctamente y es puesto en marcha.
No será competencia del proyectista comprobar el cumplimiento de las comprobaciones especificadas por parte de la empresa instaladora.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

CONTROL DE POSICIÓN DE UN PARKING AUTOMÁTICO

3. PRESUPUESTO

SEPTIEMBRE DE 2019

GUILLEM REGO MÁÑEZ
RUBÉN PUCHE PANADERO Y ÁNGEL SAPENA BAÑÓ

Tabla de contenido

3.1 INTRODUCCIÓN

En el presente documento se van a detallar los gastos necesarios para la obtención de los materiales, software y mano de obra para poder realizar el montaje en el laboratorio. No se contemplan gastos de otras instalaciones del parking ni de la instalación de nuestro sistema en un parking real.

3.2 COSTE DE LOS MATERIALES

En la tabla 3.1 se muestran los elementos utilizados y su coste.

Materiales			
Concepto	**Unidades**	**Precio unitario (€)**	**Coste total (€)**
Autómata ABB AC500 PM571	1	774,00€	774,00€
Cable señal encoder	3	53,99€	161,97€
Cable potencia servomotor	3	34,59€	103,77€
Módulo de E/S DC541	1	241,00€	241,00€
Servodriver ABB BSD0200	3	489,90€	1.469,70€
Servomotor ABB BSM0200CN00	3	272,49€	816,57€
Fuente de alimentación CP-C 24/5.0	1	126,00€	126,00€
		TOTAL	3.690,01€

Tabla 3.1 Coste de materiales

3.3 COSTE DEL SOFTWARE

Para aplicar un precio justo de coste por licencia tendremos en cuenta que estas son anuales, con lo que dividiremos el precio anual por las horas de trabajo. Contando que se trabaja 5 días a la semana, de lunes a viernes, y descontando 30 días de vacaciones obtenemos 225 días de trabajo al año, de esta manera, con 8 horas de trabajo diarias, obtenemos 1800 horas de trabajo.

En la tabla 3.2 detallamos el precio del software del programa de control, de creación del Scada y de configuración del servodriver.

Software					
Concepto	**Precio unitario (€)**	**Horas trabajadas/año**	**Precio/hora (€)**	**Horas utilización**	**Coste total (€)**
CoDeSys	1.200€	1800	0,67€	120	80,40€
CX-Supervisor	954,99€		0,53€	40	21,20€
BSD	400€		0,22€	2	0,44€
				TOTAL	102,04€

Tabla 3.2 Coste del Software

3.4 COSTE DE LA MANO DE OBRA

En la tabla 3.3 se detallan los costes de programación, pruebas y redacción que es nuestro encargo en este proyecto. Se has estimado un precio de 18,50€/hora.

Mano de obra			
Concepto	**Horas**	**Precio unitario (€)**	**Coste total (€)**
Programación del autómata	150	18,50€	2.775,00€
Programación del SCADA	50		925,00€
Programación del servodriver	3		55,50€
Pruebas y verificaciones	30		555,00€
Redacción del proyecto	100		1.850,00€
		TOTAL	6.160,50€

Tabla 3.3 Coste de mano de obra

3.5 PRESUPUESTO DE EJECUCIÓN MATERIAL

El Presupuesto de Ejecución Material o PEM se trata de la suma de los 3 anteriores costes. Vemos su total en la tabla 3.4.

Presupuesto de Ejecución Material	
Concepto	**Coste (€)**
Software	102,04€
Materiales	3.690,01€
Mano de obra	6.160,50€
TOTAL PEM	9.952,55€

Tabla 3.4 Presupuesto de Ejecución Material

3.6 GASTOS GENERALES

Los gastos generales son aquellos gastos del negocio no relacionados directamente con el producto, luz, agua, personal administrativo, servicios, etc. Estos gastos son siempre necesarios independientemente del volumen de producción de la empresa. En este caso se aplicará el 14% en concepto de gastos generales.

Para poder calcular los gastos generales debemos obtener en primer lugar el coste de software, materiales y mano de obra para después poder calcular el porcentaje de gastos generales.

Gastos generales		
Concepto	**%**	**Coste (€)**
Software		102,04€
Materiales		3.690,01€
Mano de obra		6.160,50€
TOTAL PEM		9.952,55€
Gastos generales	14%	1.393,36€

Tabla 3.5 Gastos Generales

3.7 BENEFICIO INDUSTRIAL

A continuación, se aplicará el beneficio que obtendrá la empresa por realizar el presente proyecto. Se ha estimado un 25% del coste del proyecto en concepto de beneficio industrial.

Gastos generales		
Concepto	**%**	**Coste (€)**
Software		102,04€
Materiales		3.690,01€
Mano de obra		6.160,50€
TOTAL PEM		9.952,55€
Beneficio	6%	597,15€

Tabla 3.6 Beneficio Industrial

3.8 COSTE TOTAL

Para obtener el coste total sumaremos el total de los costes de realización del proyecto, los gastos generales y el beneficio industrial y aplicaremos el 21% en concepto de IVA y de esta manera conseguiremos el coste total del proyecto.

Gastos generales		
Concepto	**%**	**Coste (€)**
Presupuesto de Ejecución Material		9.952,55€
Gastos generales		1.393,36€
Beneficio industrial		597,15€
TOTAL SIN IVA		11.943,06€
IVA	21%	2.508,04€
TOTAL		14.451,10€

Tabla 3.7 Coste total

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

CONTROL DE POSICIÓN DE UN PARKING AUTOMÁTICO

4. ANEXOS

SEPTIEMBRE DE 2019

GUILLEM REGO MÁÑEZ

RUBÉN PUCHE PANADERO Y ÁNGEL SAPENA BAÑÓ

Tabla de contenido

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

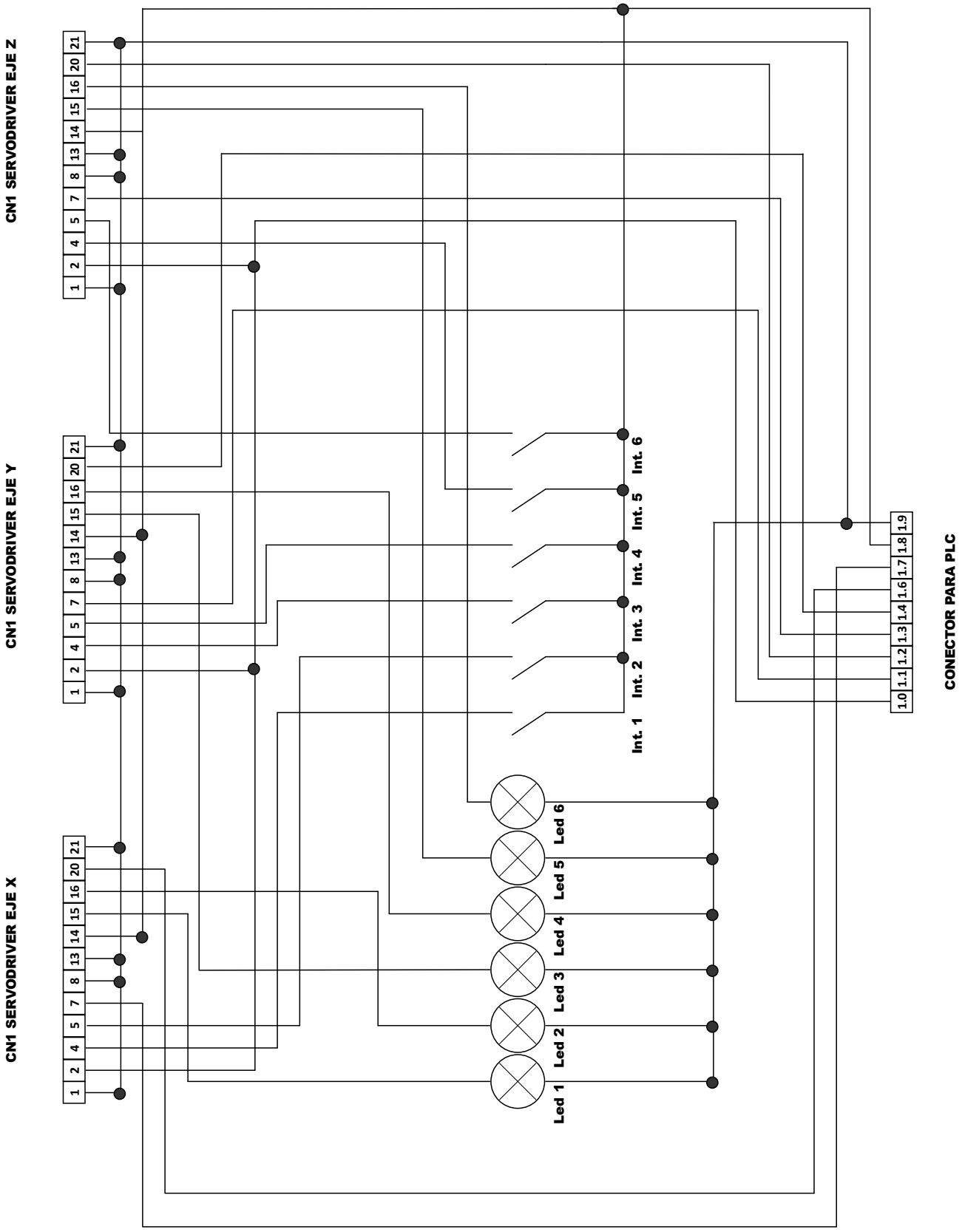
CONTROL DE POSICIÓN DE UN PARKING AUTOMÁTICO

ANEXO 1. ESQUEMAS DE CONEXIÓN

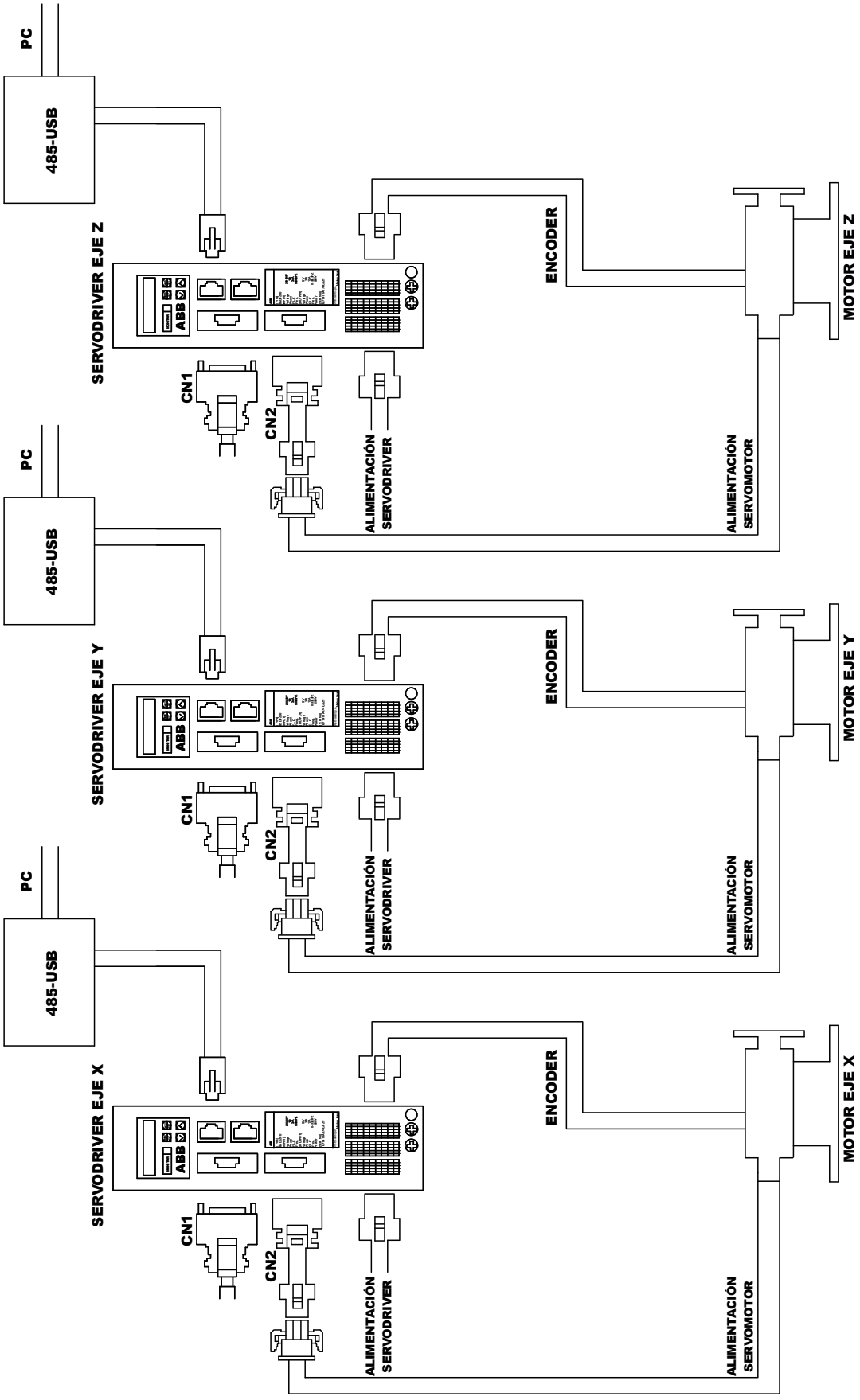
SEPTIEMBRE DE 2019

GUILLEM REGO MÁÑEZ

RUBÉN PUCHE PANADERO Y ÁNGEL SAPENA BAÑÓ



Universidad	Creado por	Tutor	
Universitat Politècnica de València	**Guillem Rego Máñez**	**Rubén Puche Panadero**	
Nombre del proyecto		Cotutor	
Control de posición de un parking automático		**Ángel Sapena Bañó**	
UNIVERSITAT POLITÈCNICA DE VALÈNCIA	Título	Fecha	Hoja
	Conexión servodrivers a PLC	**Julio 2019**	**1**



Universidad	Creado por	Tutor	
Universitat Politècnica de València	**Guillem Rego Máñez**	**Rubén Puche Panadero**	
Nombre del proyecto		Cotutor	
Control de posición de un parking automático		**Ángel Sapena Bañó**	
UNIVERSITAT POLITÈCNICA DE VALÈNCIA	Título: **Conexión servodrivers a servomotores y PC**	Fecha: **Julio 2019**	Hoja: **1**

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

CONTROL DE POSICIÓN DE UN PARKING AUTOMÁTICO

ANEXO 2. PROGRAMA DE CONTROL - CODESYS

SEPTIEMBRE DE 2019

GUILLEM REGO MÁÑEZ
RUBÉN PUCHE PANADERO Y ÁNGEL SAPENA BAÑÓ

Tabla de contenido

4.2.1 LÍNEAS DE CÓDIGO

```
1. M[Columnas*Filas + (Filas-1)*Columnas + (Columnas-1)/2]:=1;
2.
3. IF Inicializacion=1 OR PrimerArranque=0 THEN
4. miDC541_IO (EN   := TRUE, SLOT := 1, OUT0 := 1);
5. stopX:=1;
6. stopY:=1;
7. stopZ:=1;
8. startX:=0;
9. startY:=0;
10.      startZ:=0;
11.      miDC541_Freq_Y(SLOT := 1, CH:=1,EN_VISU :=TRUE, EN :=marcha
   , START :=startY, STOP := stopY, FREQ := frecY, PULSE := PulsY);
12.      miDC541_Freq_Z(SLOT := 1, CH:=7,EN_VISU :=TRUE, EN :=marcha
   , START :=startZ, STOP := stopZ, FREQ := frecZ, PULSE := PulsZ);
13.      miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha
   , START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
14.      startX:=1;
15.      startY:=1;
16.      startZ:=1;
17.      stopX:=0;
18.      stopY:=0;
19.      stopZ:=0;
20.      Inicializacion:=0;
21.      PrimerArranque:=1;
22.      movX:=0;
23.      END_IF
24.
25.      Wait(IN:=Temp , PT:=T#17s);
26.      Espera(IN:=Tempo , PT:=T#11s*Mult);
27.      Sleep(IN:=Tempor , PT:=T#14s*(Filas-o));
28.
29.      IF Pausa.Q=0 THEN
30.      Pausa(IN:=Temp2 , PT:=T#17s);
31.      END_IF
32.
33.
34.      IF Movimientoservo=1 THEN
35.      IF movX=0 THEN
36.      miDC541_IO (EN   := TRUE, SLOT := 1, OUT2 := 1);
37.      miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha
   , START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
38.      movX:=1;
39.      END_IF
40.      IF Wait.Q =1 THEN
41.      con:=1;
42.      END_IF
43.      IF Wait.Q=1 AND w<>((Columnas+1)/2) THEN
44.      Temp:=0;
45.      Tempo:=1;
46.      IF senY=1 THEN
47.      miDC541_IO (EN   := TRUE, SLOT := 1, OUT4 := 1);
48.      ELSE
49.      miDC541_IO (EN   := TRUE, SLOT := 1, OUT4 := 0);
```

```
50.       END_IF
51.       miDC541_Freq_Y(SLOT := 1, CH:=1,EN_VISU :=TRUE, EN :=marcha
   , START :=startY, STOP := stopY, FREQ := frecY, PULSE := PulsY);
52.       END_IF
53.       IF o<Filas AND Espera.Q=1 OR (o<Filas AND w=(Columnas+1/2)
   AND Wait.Q=1) THEN
54.       Tempo:=0;
55.       Tempor:=1;
56.       miDC541_IO (EN    := TRUE, SLOT := 1, OUT6 := 0);
57.       miDC541_Freq_Z(SLOT := 1, CH:=7,EN_VISU :=TRUE, EN :=marcha
   , START :=startZ, STOP := stopZ, FREQ := frecZ, PULSE := PulsZ);
58.       END_IF
59.       IF Sleep.Q=1 OR (Espera.Q=1 AND o=Filas) OR (Wait.Q=1 AND o
   =Filas AND w=((Columnas+1)/2) )THEN
60.       IF con=1 THEN
61.       stopX:=1;
62.       startX:=0;
63.       miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha
   , START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
64.       stopX:=0;
65.       startX:=1;
66.       con:=0;
67.       END_IF
68.       Tempo:=0;
69.       Tempor:=0;
70.       Temp2:=1;
71.       IF senX=1 THEN
72.       miDC541_IO (EN    := TRUE, SLOT := 1, OUT2 := 1);
73.       ELSE
74.       miDC541_IO (EN    := TRUE, SLOT := 1, OUT2 := 0);
75.       END_IF
76.       miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha
   , START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
77.       END_IF
78.       IF Pausa.Q=1 THEN
79.       Temp2:=0;
80.       Pausa(IN:=Temp2 , PT:=T#17s);
81.       Retorno:=1;
82.       Inicializacion:=1;
83.       Movimientoservo:=0;
84.       END_IF
85.       END_IF
86.
87.       IF Retorno=1 THEN
88.       IF cuenta=0 THEN
89.       Temp:=1;
90.       cuenta:=1;
91.       END_IF
92.       IF movX=0 THEN
93.       IF senX=1 THEN
94.       miDC541_IO (EN    := TRUE, SLOT := 1, OUT2 := 0);
95.       ELSE
96.       miDC541_IO (EN    := TRUE, SLOT := 1, OUT2 := 1);
97.       END_IF
98.       miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha
   , START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
```

```
99.      movX:=1;
100.     END_IF
101.     IF Wait.Q =1 THEN
102.     con:=1;
103.     END_IF
104.     IF Wait.Q=1 AND w<>((Columnas+1)/2) THEN
105.     Temp:=0;
106.     Tempo:=1;
107.     IF senY=0 THEN
108.     miDC541_IO (EN    := TRUE, SLOT := 1, OUT4 := 1);
109.     ELSE
110.     miDC541_IO (EN    := TRUE, SLOT := 1, OUT4 := 0);
111.     END_IF
112.     miDC541_Freq_Y(SLOT := 1, CH:=1,EN_VISU :=TRUE, EN :=marcha
   , START :=startY, STOP := stopY, FREQ := frecY, PULSE := PulsY);
113.     END_IF
114.     IF o<FIlas AND Espera.Q=1 OR (o<FIlas AND w=((Columnas+1)/2
   ) AND Wait.Q=1) THEN
115.     miDC541_IO (EN    := TRUE, SLOT := 1, OUT6 := 1);
116.     miDC541_Freq_Z(SLOT := 1, CH:=7,EN_VISU :=TRUE, EN :=marcha
   , START :=startZ, STOP := stopZ, FREQ := frecZ, PULSE := PulsZ);
117.     Tempo:=0;
118.     Tempor:=1;
119.     END_IF
120.     IF Sleep.Q=1 OR (Espera.Q=1 AND o=Filas) OR (Wait.Q=1 AND o
   =Filas AND w=((Columnas+1)/2)) THEN
121.     IF con=1 THEN
122.     stopX:=1;
123.     startX:=0;
124.     miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha
   , START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
125.     stopX:=0;
126.     startX:=1;
127.     con:=0;
128.     END_IF
129.     Tempo:=0;
130.     Tempor:=0;
131.     Temp2:=1;
132.     miDC541_IO (EN    := TRUE, SLOT := 1, OUT2 := 0);
133.     miDC541_Freq_X(SLOT := 1, CH:=3,EN_VISU :=TRUE, EN :=marcha
   , START :=startX, STOP := stopX, FREQ := frecX, PULSE := pulsX);
134.     END_IF
135.     IF Pausa.Q=1 THEN
136.     Temp2:=0;
137.     Pausa(IN:=Temp2 , PT:=T#17s);
138.     Retorno:=0;
139.     PosicionInicial:=1;
140.     Inicializacion:=1;
141.     cuenta:=0;
142.     END_IF
143.     END_IF
144.
145.     IF PeticionAparcamiento=1 AND PosicionInicial=1 AND marcha=
   1 AND Movimientoservo=0THEN
146.     PeticionAparcamiento:=0;
147.     Aparcado:=0;
```

```
148.     FOR i:=Filas TO 1 BY -1 DO
149.     FOR j:=1 TO Columnas DO
150.     FOR k:=1 TO 2 DO
151.     IF M[(k-1)*Columnas*Filas + (i-1)*Columnas + (j-
   1)]=0 AND Aparcado=0 THEN
152.     senZ:=0;
153.     PosicionInicial:=0;
154.     Movimientoservo:=1;
155.     Temp:=1;
156.     Posicion:=(k-1)*11*5 + (i-1)*11 + (j-1);
157.     o:=i;
158.     w:=j;
159.     IF w<((Columnas+1)/2) THEN
160.     Mult:=((Columnas+1)/2)-w;
161.     ELSIF w>((Columnas+1)/2) THEN
162.     Mult:=w-((Columnas+1)/2);
163.     END_IF
164.     pulsZ:=pulsZ1*(Filas-i);
165.     pulsY:=pulsY1*(Mult);
166.     IF j<((Columnas/2)+1) THEN
167.     senY:=0;
168.     ELSIF ((Columnas/2)+1)>6 THEN
169.     senY:=1;
170.     END_IF;
171.     IF k=1 THEN
172.     senX:=0;
173.     ELSIF k=2 THEN
174.     senX:=1;
175.     END_IF;
176.     M[Posicion]:=1;
177.     Aparcado:=1;
178.     END_IF;
179.     END_FOR;
180.     END_FOR;
181.     END_FOR;
182.     END_IF;
183.
184.     IF PeticionRetirada=1 AND PeticionAparcamiento=0 AND Posici
   onInicial=1 AND Movimientoservo=0  THEN
185.     PeticionRetirada:=0;
186.     IF M[(p-1)*Columnas*Filas + (f-1)*Columnas + (c-
   1)]=1 AND (f=5 AND c=6 AND p=1)=FALSE THEN
187.     pulsZ:=pulsZ1*(Filas-f);
188.     o:=f;
189.     w:=c;
190.     IF w<6 THEN
191.     Mult:=6-w;
192.     ELSIF w>6 THEN
193.     Mult:=w-6;
194.     END_IF
195.     pulsY:=pulsY1*(Mult);
196.     senZ:=0;
197.     Movimientoservo:=1;
198.     Temp:=1;
199.     PosicionInicial:=0;
200.     IF c<6 THEN
```

```
201.      senY:=0;
202.      ELSIF c>6 THEN
203.      senY:=1;
204.      END_IF;
205.      IF p=1 THEN
206.      senX:=0;
207.      ELSIF p=2 THEN
208.      senX:=1;
209.      END_IF;
210.      M[(p-1)*Columnas*Filas + (f-1)*Columnas + (c-1)]:=0;
211.              END_IF;
212.      END_IF;
```

4.2.2 LISTA DE VARIABLES UTILIZADAS

En este apartado recogeremos las distintas variables utilizadas en el programa, el tipo de variable y una breve descripción:

Variable	Tipo	Descripción
i	INT	Variable que cambiará de valor por el primer bucle FOR y equivaldrá al número de la columna.
j	INT	Variable que cambiará de valor por el segundo bucle FOR y equivaldrá al número de la fila.
k	INT	Variable que cambiará de valor por el tercer bucle FOR y equivaldrá a la profundidad.
f	INT	Variable que vendrá determinada por el código del usuario a la hora de retirar su vehículo y equivaldrá al número de la fila.
c	INT	Variable que vendrá determinada por el código del usuario a la hora de retirar su vehículo y equivaldrá al número de la columna.
p	INT	Variable que vendrá determinada por el código del usuario a la hora de retirar su vehículo y equivaldrá al número de la profundidad.
o	INT	Variable que almacenará el valor de i o de f y que nos servirá para enviar el número de pulsos adecuado en el eje Y
w	INT	Variable que almacenará el valor de j o de c y que nos servirá para enviar el número de pulsos adecuado en el eje Z
M	ARRAY [1..11, 1..5, 1..2] OF BOOL	Matriz cuyos elementos equivalen a cada una de las plazas y en la que se almacenarán valores que determinarán si están ocupadas o no
PosicionInicial	BOOL	Variable con la que controlaremos sí la plataforma está en la posición inicial.
PeticionAparacamiento	BOOL	Variable cuyo valor será 1 cuando un usuario solicite el estacionamiento del vehículo

PeticionRetirada	BOOL	Variable cuyo valor será 1 cuando un usuario solicite la retirada del vehículo
Aparcado	BOOL	Variable que nos indicará si el vehículo ya se ha estacionado
Marcha	BOOL	Variable que determinará la marcha de los 3 servomotores
senX	BOOL	Variable que determinará el sentido del movimiento del servomotor del eje X. 0 = sentido positivo. 1= sentido negativo
senY	BOOL	Variable que determinará el sentido del movimiento del servomotor del eje Y. 0 = sentido positivo. 1= sentido negativo
senZ	BOOL	Variable que determinará el sentido del movimiento del servomotor del eje Z. 0 = sentido positivo. 1= sentido negativo
FreqX	LREAL	Frecuencia de funcionamiento del servomotor del eje X. Equivale a la frecuencia de nuestra corriente 50Hz.
FreqY	LREAL	Frecuencia de funcionamiento del servomotor del eje Y. Equivale a la frecuencia de nuestra corriente 50Hz.
FreqZ	LREAL	Frecuencia de funcionamiento del servomotor del eje Z. Equivale a la frecuencia de nuestra corriente 50Hz.
PulsX	DWORD	Número de pulsos que le enviaremos al servo del eje X para mover la plataforma a donde se requiera.
PulsY	DWORD	Número de pulsos que le enviaremos al servo del eje Y para mover la plataforma a donde se requiera.
PulsZ	DWORD	Número de pulsos que le enviaremos al servo del eje Z para mover la plataforma a donde se requiera.
PulsY1	DWORD	Número de pulsos que se han de envíar para mover la plataforma en el eje Y la distancia equivalente a una plaza de estacionamiento.
PulsZ1	DWORD	Número de pulsos que se han de envíar para mover la plataforma en el eje Z la distancia equivalente a una plaza de estacionamiento.

Mult	UINT	Variable sin signo que equivaldrá al número de plazas que se tiene que mover la plataforma en el eje Y.
startX	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje X.
stopY	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje Y.
StartZ	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje Z.
stopX	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje X.
startY	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje Y.
stopZ	BOOL	Variable que sirve para empezar el movimiento del servomotor del eje Z.
Temp	BOOL	Variable que sirve para activar el temporizador Wait.
Tempo	BOOL	Variable que sirve para activar el temporizador Espera.
Tempor	BOOL	Variable que sirve para activar el temporizador Sleep.
Temp2	BOOL	Variable que sirve para activar el temporizador Pausa.
Posición	INT	Variable que sirve para almacenar el valor de la plaza requerida.
Filas	INT	Variable con la que podemos variar el número de filas del parking
Columnas	INT	Variable con la que podemos variar el número de columnas del parking
Inicialización	BOOL	Variable que sirve para ejecutar la inicialización de los servomotores.
PrimerArranque	BOOL	Variable que sirve para ejecutar la inicialización de los servomotores la primera vez que ejecutamos el programa
movX	BOOL	Variable que evitará la repetición del primer movimiento en el eje X.
con	BOOL	Variable que inicializará los servomotores entre el primer y el segundo movimiento en el eje X.

Tabla 4.2.1 Lista de variables utilizadas

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

CONTROL DE POSICIÓN DE UN PARKING AUTOMÁTICO

ANEXO 3. MANUAL DEL USUARIO DE SCADA

SEPTIEMBRE DE 2019
GUILLEM REGO MÁÑEZ
RUBÉN PUCHE PANADERO Y ÁNGEL SAPENA BAÑÓ

Tabla de contenido

4.3.1 INTRODUCCIÓN Y PRIMEROS PASOS

Nuestro sistema SCADA va a tratar de un visualizador de nuestro parking donde encontraremos los coches aparcados en las plazas ocupadas. También tendremos una página para realizar nuestra simulación con botones para cambiar las plazas ocupadas, activar la marcha, solicitar estacionamiento o retirada del vehículo e introducir la plaza donde se encuentra el coche que queremos retirar.

Separaremos la estructura de nuestro proyecto en 2 páginas y 2 scripts.

4.3.1.1 PÁGINA 1

En esta página tendremos los botones para controlar y realizar nuestra simulación.

Empezamos añadiendo un botón de activar y desactivar con un estilo de conmutador rotatorio con el que cambiaremos el valor de marcha a 0 o 1. Añadiremos además un texto que indique “Marcha” para visualizar rápidamente de lo que se trata.

Añadiremos un botón para solicitar aparcamiento que al hacer click sobre él se visualizará la página 2 y se cambiará el valor de la variable PeticionAparcamiento a 1 con la siguiente línea de comando:

```
PeticionAparcamiento=1
```

Tras añadir este añadiremos un botón similar pero para solicitar retirada que al igual que el anterior al hacer click sobre él se visualizará la página 2 y esta vez en lugar de PeticionAparcamiento cambiaremos el valor de la variable PeticionRetirada con la siguiente línea de comando:

```
PeticionRetirada=1
```

Por último, respecto a la función de Retirar introduciremos un botón que servirá para introducir la plaza de nuestro vehículo antes de solicitar la retirada del mismo. Con esto simularemos la lectura de un ticket mediante introducción del número de fila, columna y profundidad.

Lo que hará este botón será mostrar 3 cuadros de texto en el que se pedirá el número de fila, columna y profundidad con la apariencia de la figura 4.3.1.

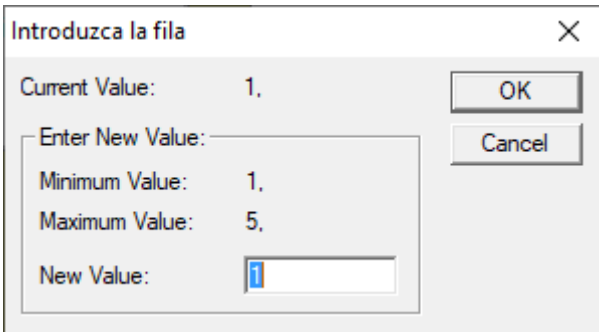


Figura 4.3.1 Ventana de introducción de fila

Para conseguir esto hemos programado estas líneas de comando para cuando se hace click sobre el botón:

```
EditPoint(f, "Introduzca la fila", 1.000000, 5.000000, FALSE )
EditPoint(c, "Introduzca la columna", 1.000000, 11.000000, FALSE )
EditPoint(p, "Introduzca la profundidad", 1.000000, 2.000000, FALSE )
```

Tras esto añadiremos 2 botones, el primero para establecer cada valor de la matriz M, donde almacenamos si las plazas están ocupadas o no, igual a 1, y el segundo para establecerlo en 0. Ambos seguirán esta estructura únicamente cambiando el valor de M[o].

```
FOR o=0 TO 109 STEP 1
M[o]=1
NEXT
```

Por último añadiremos 3 botones que lo que harán al hacerse click sobre ellos es editar tres puntos, 1 cada botón, dentro de la matriz M correspondientes a M[44] , M[92] y M[31]. A la hora de hacer click en uno de estos 3 botones nos encontraremos con una ventana de texto como la de la figura 4.3.2

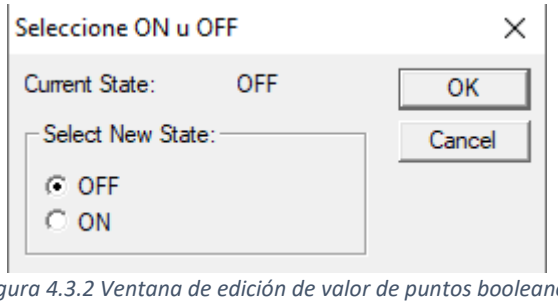


Figura 4.3.2 Ventana de edición de valor de puntos booleanos

Finalmente, la página nos queda con la siguiente apariencia:

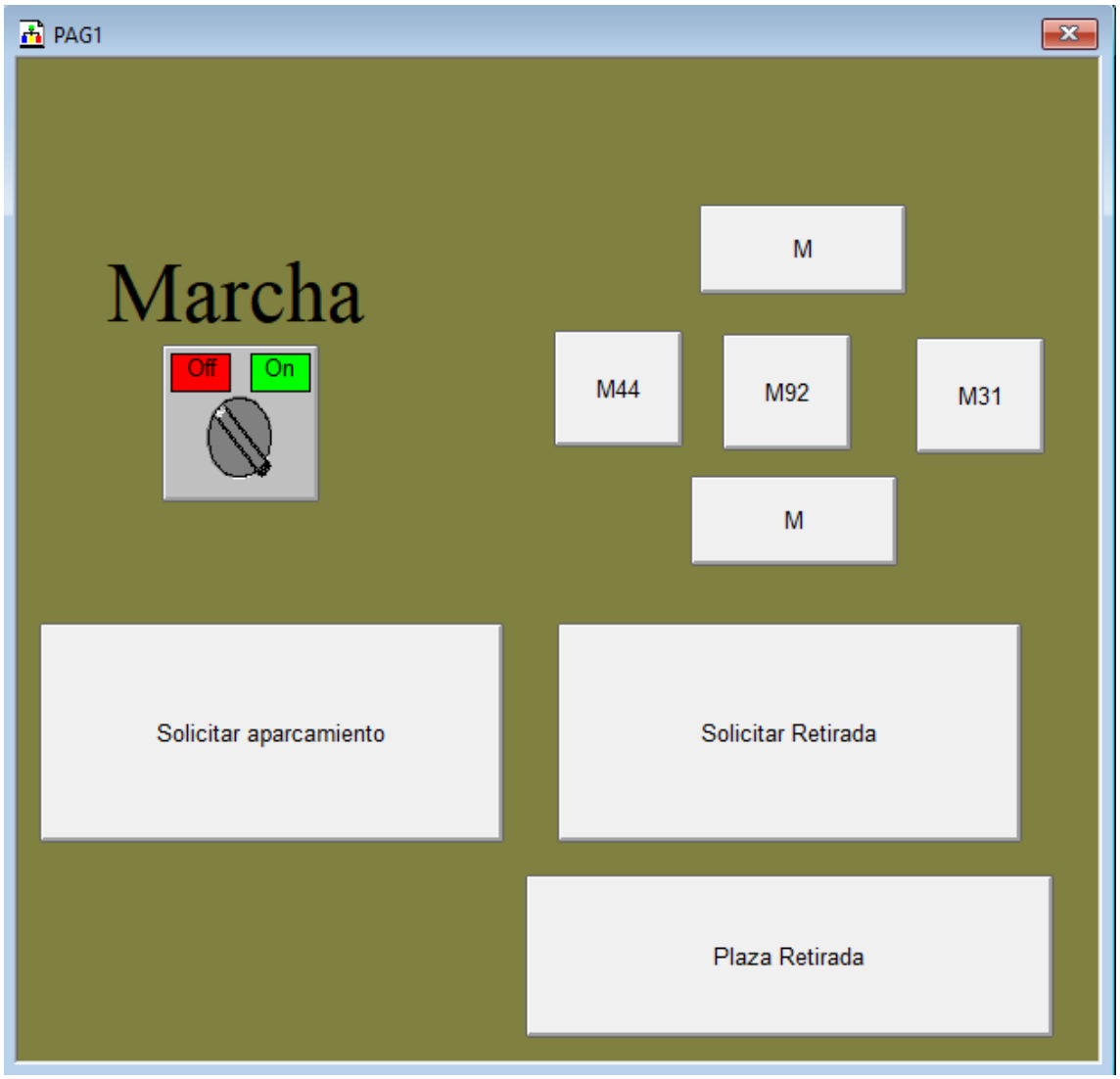


Figura 4.3.3 Página 1

4.3.1.2 PÁGINA 2

En esta página representaremos, de forma bidimensional donde nuestros ejes X y Z del programa de CoDeSys compartirán eje, nuestro parking con dos imágenes separadas por un espacio que representará el espacio intermedio entre las dos hileras de coches de la forma de la figura de la figura 4.3.4

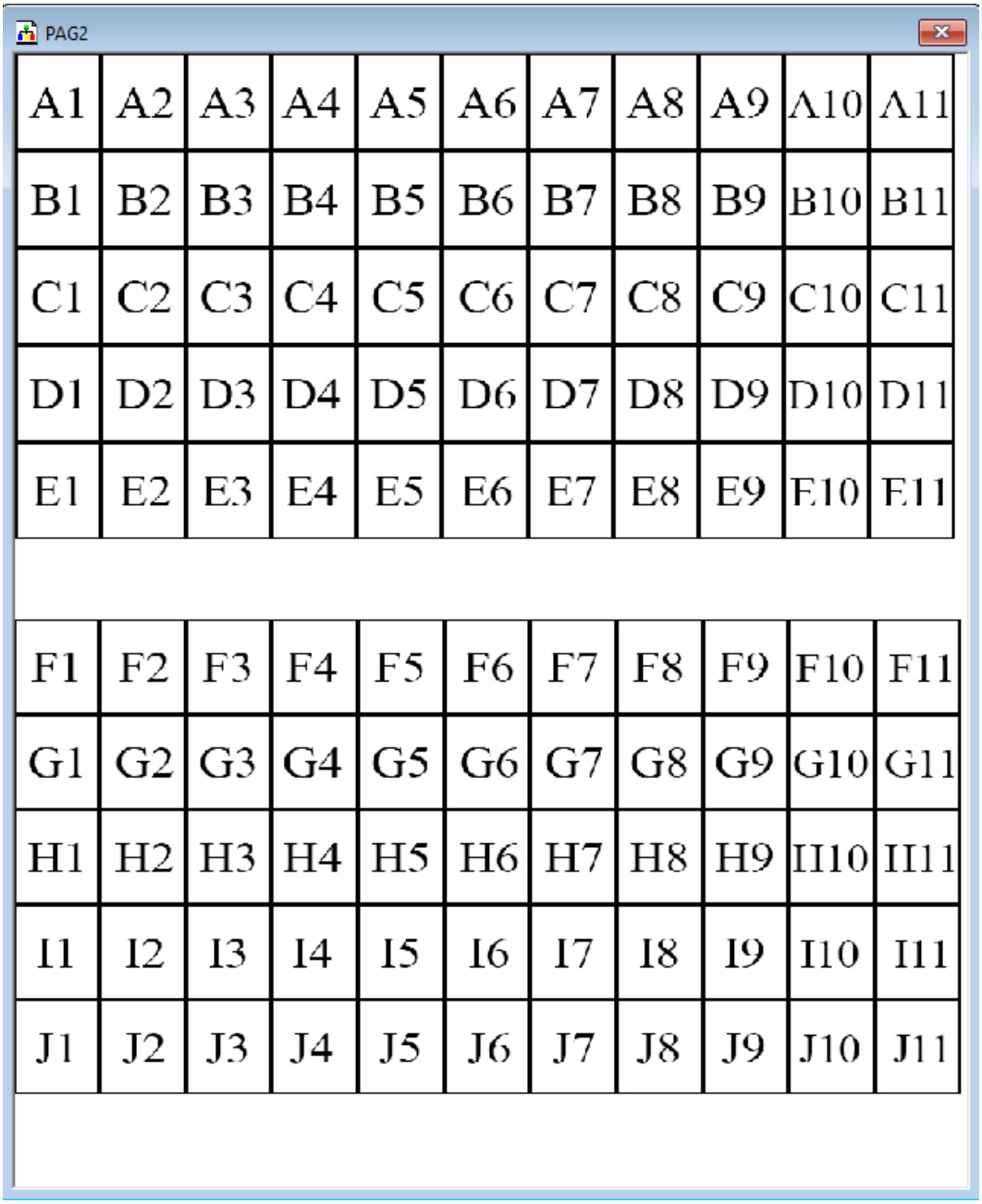


Figura 4.3.4 Plazas de aparcamiento

Tras tener nuestra estructura del parking representada, añadiremos 1 objeto de la librería gráfica que representará a 1 coche en cada plaza de aparcamiento, con la condición de visibilidad de que el valor de su plaza sea igual a 1, obteniendo lo mostrado en la figura 4.3.5

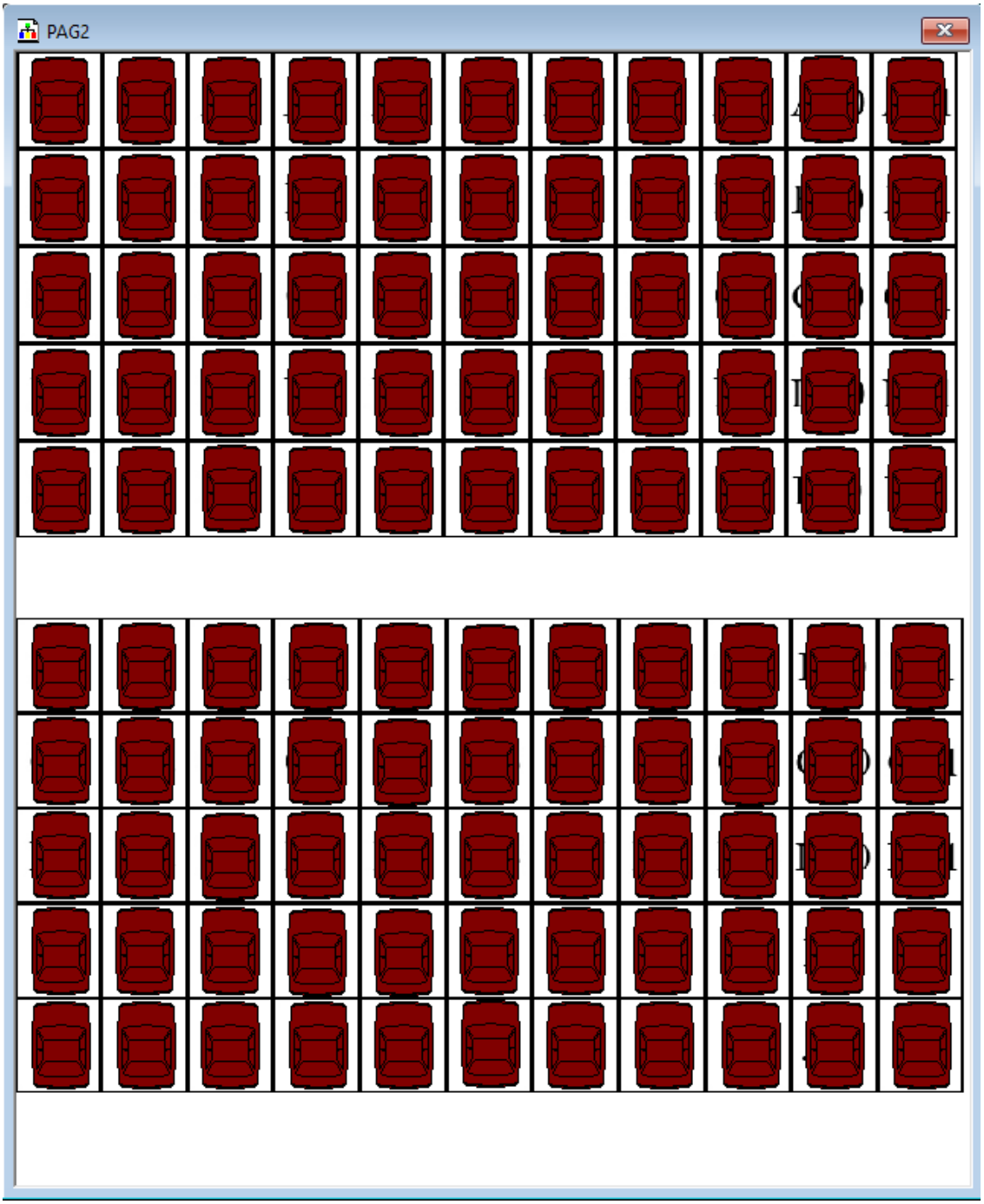


Figura 4.3.5 Plazas ocupadas

Por último, añadiremos 1 rectángulo rojo que representará a nuestra plataforma.

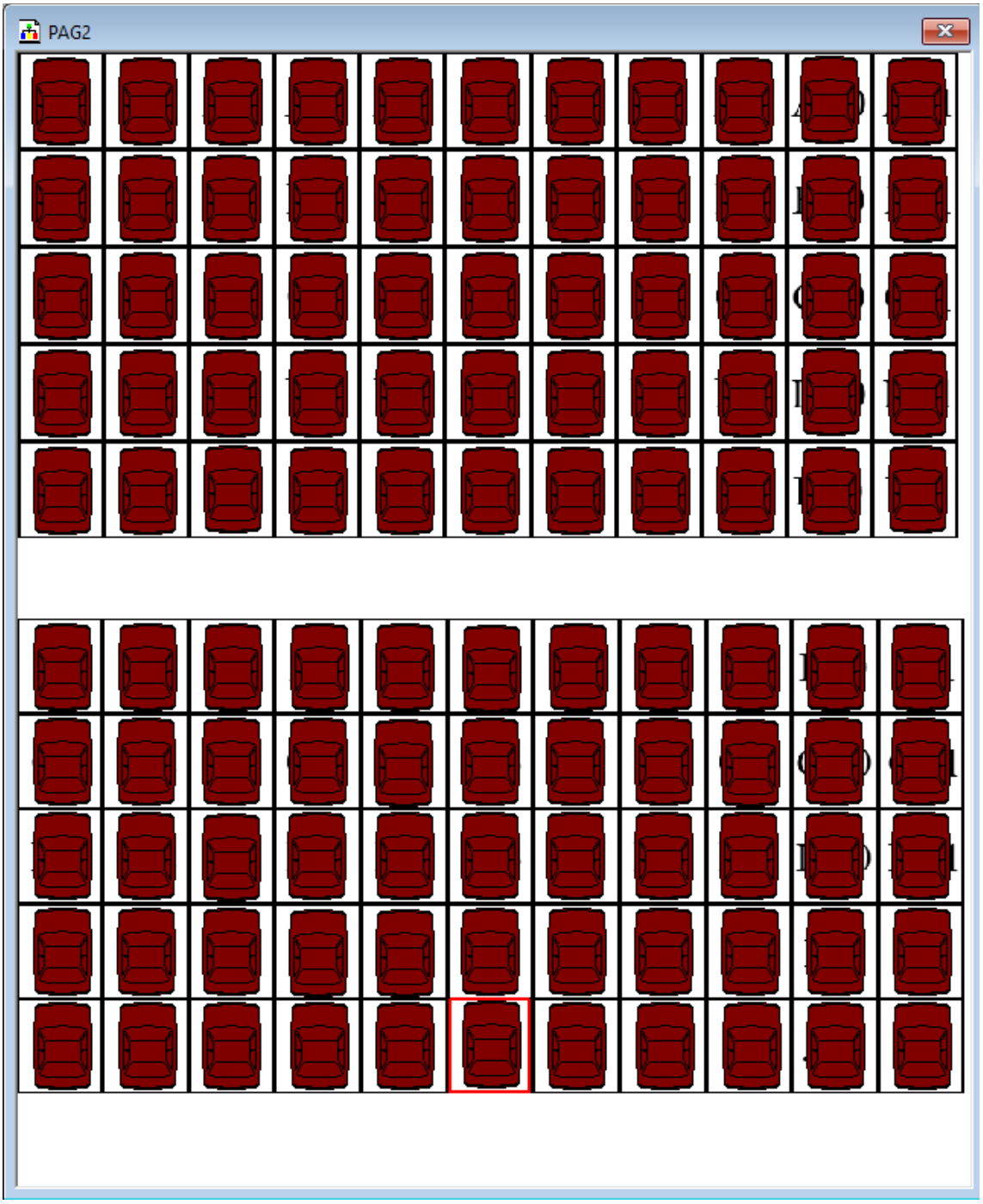


Figura 4.3.6 Página 2

4.3.1.3 SCRIPT 1: APARCAR

El script Aparcar como su nombre indica nos servirá para realizar la representación gráfica del proceso de estacionamiento del vehículo. Para ello haremos, antes que nada, que la visibilidad de nuestro coche en la plaza inicial M[104] dependa de una variable llamada vis que usaremos en el script.

Las condiciones del Script serán las de la figura 4.3.7.

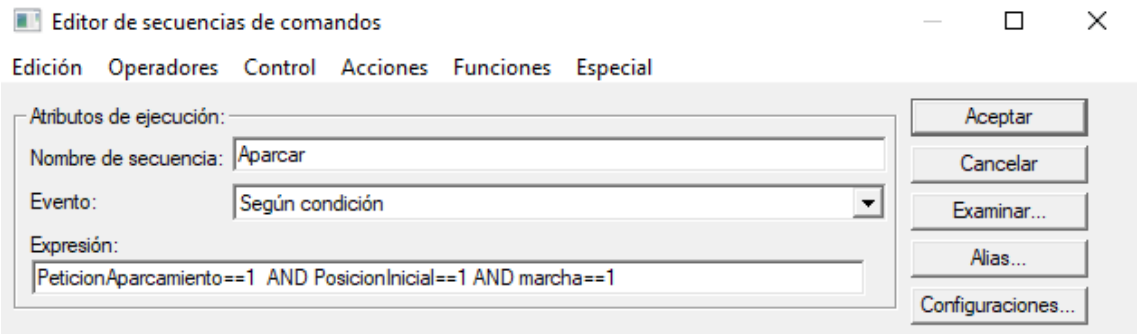


Figura 4.3.7 Condiciones del script Aparcar

En este script lo que haremos será asignar el valor de M[104] y de vis igual a 1 para que no se pueda estacionar en la plaza inicial y para visualizar el coche de la animación.

Tras esto, como en CoDeSys tendremos un bucle FOR y un condicional IF para comprobar qué plaza está vacía para estacionar en ese lugar el vehículo.

Una vez dentro del condicional IF desplazaremos el coche de la plaza 104 y la plataforma con otro bucle FOR y un comando de reposo Sleep para que no sea un movimiento instantáneo.

También retornaremos la plataforma mediante un movimiento no instantáneo, pero el coche de la plaza 104 sí lo retornaremos mediante movimiento instantáneo ya que lo haremos invisible cambiando el valor de vis a 0 ya que el coche en la plaza a la que se había desplazado ya es visible ya que cuando termine el primer conjunto de movimientos, ir de la plaza 104 a la correspondiente, del coche de M[104] se asignará como valor de M[plaza correspondiente] = 1.

Teniendo la siguiente estructura:

```
1. M[104]=1
2. vis=1
3. PeticionAparcamiento=0
4. MovX=0
5. FOR i=5 TO 1 STEP -1
6. FOR j=1 TO 11
7. FOR k=1 TO 2
8. Posicion=((k-1)*5*11)+((i-1)*11)+(j-1)
9. IF M[Posicion]==0 AND Aparcado==0 THEN
10.      PosicionInicial=0
11.      FOR y=0 TO 310
12.      PAG2.CAR68.move( 0 , -y )
13.      PAG2.Rectangle_1.move( 0 , -y )
14.      Sleep(50)
15.      NEXT
16.      Sleep(100)
17.      FOR MovimientoX=0 TO 300
18.      IF MovimientoX<=(6-j)*56 AND j<6 THEN
19.      PAG2.CAR68.move( -MovimientoX , -y )
20.      PAG2.Rectangle_1.move( -MovimientoX , -y )
21.      MovX=MovimientoX
22.      Sleep(50)
23.      ENDIF
24.      IF MovimientoX<=(j-6)*56 AND j>6 THEN
25.      PAG2.CAR68.move( MovimientoX , -y )
26.      PAG2.Rectangle_1.move( MovimientoX , -y )
27.      MovX=-MovimientoX
28.      Sleep(50)
29.      ENDIF
30.      NEXT
31.      FOR MovimientoY=0 TO 310
32.      IF MovimientoY<=(6-i)*60 AND k==1 THEN
33.      PAG2.CAR68.move( -MovX , -y-MovimientoY )
34.      PAG2.Rectangle_1.move( -MovX , -y-MovimientoY )
35.      Sleep(50)
36.      ENDIF
37.      IF MovimientoY<=i*62 AND k==2 THEN
38.      PAG2.CAR68.move( -MovX , -y+MovimientoY )
39.      PAG2.Rectangle_1.move( -MovX , -y+MovimientoY )
40.      Sleep(50)
41.      ENDIF
42.      NEXT
43.      Sleep(500)
44.      IF k==1 THEN
45.      FOR MovimientoY=-(11-i)*60 TO -310
46.      PAG2.Rectangle_1.move( (j-6)*56 , MovimientoY )
47.      Sleep(50)
48.      NEXT
49.      ENDIF
50.      IF k==2 THEN
51.      PAG2.Rectangle_1.move((j-6)*56 , (i-5)*62 )
52.      FOR MovimientoY=(5-i)*62 TO 310
53.      PAG2.Rectangle_1.move( (j-6)*56 , -MovimientoY )
```

```
54.       Sleep(50)
55.       NEXT
56.       ENDIF
57.       Sleep(100)
58.       IF j<6 THEN
59.       FOR MovimientoX=(6-j)*56 TO 0 STEP -1
60.       PAG2.Rectangle_1.move( -MovimientoX , -310 )
61.       Sleep(50)
62.       NEXT
63.       ENDIF
64.       IF j>6 THEN
65.       FOR MovimientoX=(j-6)*56 TO 0 STEP -1
66.       PAG2.Rectangle_1.move( MovimientoX , -310 )
67.       Sleep(50)
68.       NEXT
69.       ENDIF
70.       Sleep(100)
71.       FOR y=310 TO 0 STEP -1
72.       PAG2.Rectangle_1.move( 0 , -y )
73.       Sleep(50)
74.       NEXT
75.       Sleep(100)
76.       PAG2.CAR68.move( 0 , 0 )
77.       vis=0
78.       M[Posicion]=1
79.       Aparcado=1
80.       ENDIF
81.       NEXT
82.       NEXT
83.       NEXT
84.       PosicionInicial=1
85.       Aparcado=0
```

4.3.1.4 SCRIPT 2: RETIRADA

En este punto comenzaremos hablando sobre las condiciones del script ya que el funcionamiento propio del script es muy similar al anterior. Las condiciones del script son la de la figura 4.3.8.

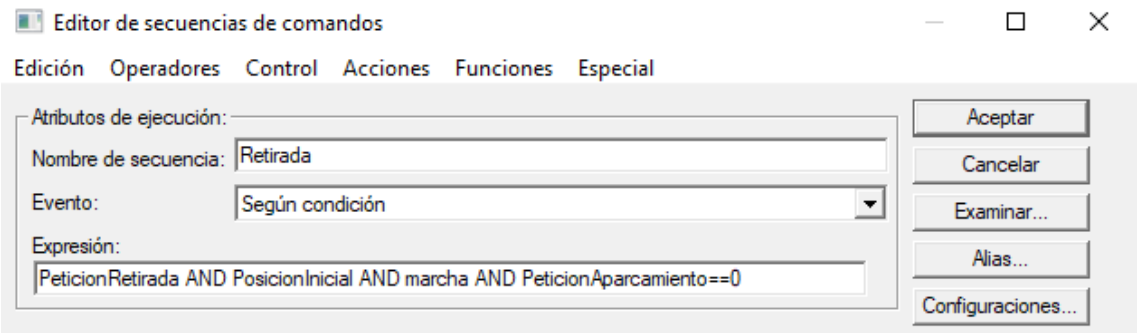


Figura 4.3.8 Condiciones del script Retirada

Como en CoDeSys, aquí también, la función de Retirada tendrá características similares al script Aparcar.

Se comprobará que la plaza que hemos pedido está ocupada y si es así, se hará un movimiento instantáneo del coche de la plaza inicial a la requerida mientras desplazamos la plataforma a la plaza requerida usando de nuevo bucle FOR y función Sleep.

Una vez llegue el coche de la plaza se hará invisible debido a que el valor de la plaza dentro de la matriz M será 0 y haremos visible el coche que hemos desplazado a esa plaza para desplazarlo, al igual que la plataforma, a la posición inicial.

Una vez en la posición inicial cambiamos el valor de vis a 0 para devolver a su situación inicial el coche de la plaza 104 que es la plaza inicial.

Tras lo dicho anteriormente nos queda la siguiente estructura:

```
1. PeticionRetirada=0
2. Posicion=((p-1)*5*11)+((f-1)*11)+(c-1)
3.
4. IF Posicion==104 THEN
5. Message("Posición Incorrecta. Vuelve a introducir posición.")
6. ENDIF
7.
8. IF M[Posicion]==0 AND Posicion!=104 THEN
9. Message("Plaza vacía. Vuelve a introducir posición.")
10.      ENDIF
11.
12.      IF M[Posicion]==1 AND Posicion!=104 THEN
13.      PosicionInicial=0
14.      PeticionRetirada=0
15.      vis=0
16.      FOR y=0 TO 310
17.      PAG2.Rectangle_1.move( 0 , -y )
18.      Sleep(50)
19.      NEXT
20.      Sleep(100)
21.      FOR MovimientoX=0 TO 300
22.      IF MovimientoX<=(6-c)*56 AND c<6 THEN
23.      PAG2.Rectangle_1.move( -MovimientoX , -y )
24.      MovX=MovimientoX
25.      Sleep(50)
26.      ENDIF
27.      IF MovimientoX<=(c-6)*56 AND c>6 THEN
28.      PAG2.Rectangle_1.move( MovimientoX , -y )
29.      MovX=-MovimientoX
30.      Sleep(50)
31.      ENDIF
32.      NEXT
33.      FOR MovimientoY=0 TO 310
34.      IF MovimientoY<=(6-f)*60 AND p==1 THEN
35.      PAG2.Rectangle_1.move( -MovX , -y-MovimientoY )
36.      Sleep(50)
37.      ENDIF
38.      IF MovimientoY<=f*62 AND p==2 THEN
39.      PAG2.Rectangle_1.move( -MovX , -y+MovimientoY )
40.      Sleep(50)
41.      ENDIF
42.      NEXT
43.      Sleep(500)
44.      IF p==1 THEN
45.      PAG2.CAR68.move((c-6)*56 , (11-f)*(-60) )
46.      PAG2.Rectangle_1.move((c-6)*56 , (11-f)*(-60) )
47.      vis=1
48.      M[Posicion]=0
49.      FOR MovimientoY=-(11-f)*60 TO -310
50.      PAG2.CAR68.move( (c-6)*56 , MovimientoY )
51.      PAG2.Rectangle_1.move( (c-6)*56 , MovimientoY )
52.      Sleep(50)
53.      NEXT
```

```
54.       ENDIF
55.       IF p==2 THEN
56.       PAG2.CAR68.move((c-6)*56 , (f-5)*62 )
57.       PAG2.Rectangle_1.move((c-6)*56 , (f-5)*62 )
58.       vis=1
59.       M[Posicion]=0
60.       FOR MovimientoY=(5-f)*62 TO 310
61.       PAG2.CAR68.move( (c-6)*56 , -MovimientoY )
62.       PAG2.Rectangle_1.move( (c-6)*56 , -MovimientoY )
63.       Sleep(50)
64.       NEXT
65.       ENDIF
66.       Sleep(100)
67.       IF c<6 THEN
68.       FOR MovimientoX=(6-c)*56 TO 0 STEP -1
69.       PAG2.CAR68.move( -MovimientoX , -310 )
70.       PAG2.Rectangle_1.move( -MovimientoX , -310 )
71.       Sleep(50)
72.       NEXT
73.       ENDIF
74.       IF c>6 THEN
75.       FOR MovimientoX=(c-6)*56 TO 0 STEP -1
76.       PAG2.CAR68.move( MovimientoX , -310 )
77.       PAG2.Rectangle_1.move( MovimientoX , -310 )
78.       Sleep(50)
79.       NEXT
80.       ENDIF
81.       Sleep(100)
82.       FOR y=310 TO 0 STEP -1
83.       PAG2.CAR68.move( 0 , -y )
84.       PAG2.Rectangle_1.move( 0 , -y )
85.       Sleep(50)
86.       NEXT
87.       Sleep(100)
88.       PosicionInicial=1
89.       ENDIF
90.       vis=0
91.
```

4.3.2 LISTA DE PUNTOS UTILIZADOS

En este apartado recogeremos los distintos puntos utilizados en el programa, el tipo de variable y una breve descripción:

Punto	Tipo	Descripción
i	INT	Variable que cambiará de valor por el primer bucle FOR y equivaldrá al número de la columna.
j	INT	Variable que cambiará de valor por el segundo bucle FOR y equivaldrá al número de la fila.
k	INT	Variable que cambiará de valor por el tercer bucle FOR y equivaldrá a la profundidad.
f	INT	Variable que vendrá determinada por el código del usuario a la hora de retirar su vehículo y equivaldrá al número de la fila.
c	INT	Variable que vendrá determinada por el código del usuario a la hora de retirar su vehículo y equivaldrá al número de la columna.
p	INT	Variable que vendrá determinada por el código del usuario a la hora de retirar su vehículo y equivaldrá al número de la profundidad.
M	ARRAY [109] OF BOOL	Matriz cuyos elementos equivalen a cada una de las plazas y en la que se almacenarán valores que determinarán si están ocupadas o no
PosicionInicial	BOOL	Variable con la que controlaremos sí la plataforma está en la posición inicial.
PeticionAparacamiento	BOOL	Variable cuyo valor será 1 cuando un usuario solicite el estacionamiento del vehículo
PeticionRetirada	BOOL	Variable cuyo valor será 1 cuando un usuario solicite la retirada del vehículo
Aparcado	BOOL	Variable que nos indicará si el vehículo ya se ha estacionado
Marcha	BOOL	Variable que determinará la marcha de los 3 servomotores

MovX	INT	Variable que nos servirá para almacenar el valor de Movimiento X y así poder tener la posición en X para continuar el movimiento de nuestros objetos
MovimientoX	INT	Variable que determinará el movimiento en el eje X que requiera nuestro programa
MovimientoY	INT	Variable que determinará el movimiento en el eje Y que requiera nuestro programa sumado a la variable y
o	INT	Variable que nos servirá para asignar mediante un botón y un bucle FOR el valor 1 o 0 a todos los elementos de la matriz M
y	INT	Variable que nos permitirá mediante un bucle FOR trasladar nuestra plataforma y vehículo al espacio intermedio entre las 2 hileras de plazas.
vis	BOOL	Variable que determinará la visibilidad del coche que se encuentra en la plaza inicial (104).

Tabla 4.3.1 Lista de puntos utilizados

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

CONTROL DE POSICIÓN DE UN PARKING AUTOMÁTICO

ANEXO 4. CÓMO CREAR UN PROYECTO EN CODESYS

SEPTIEMBRE DE 2019
GUILLEM REGO MÁÑEZ
RUBÉN PUCHE PANADERO Y ÁNGEL SAPENA BAÑÓ

Tabla de contenido

4.4.1 CONFIGURACIÓN Y CREACIÓN DE UN NUEVO PROYECTO

Para crear un nuevo proyecto debemos dirigirnos a la pestaña *File* y seleccionar *New*.

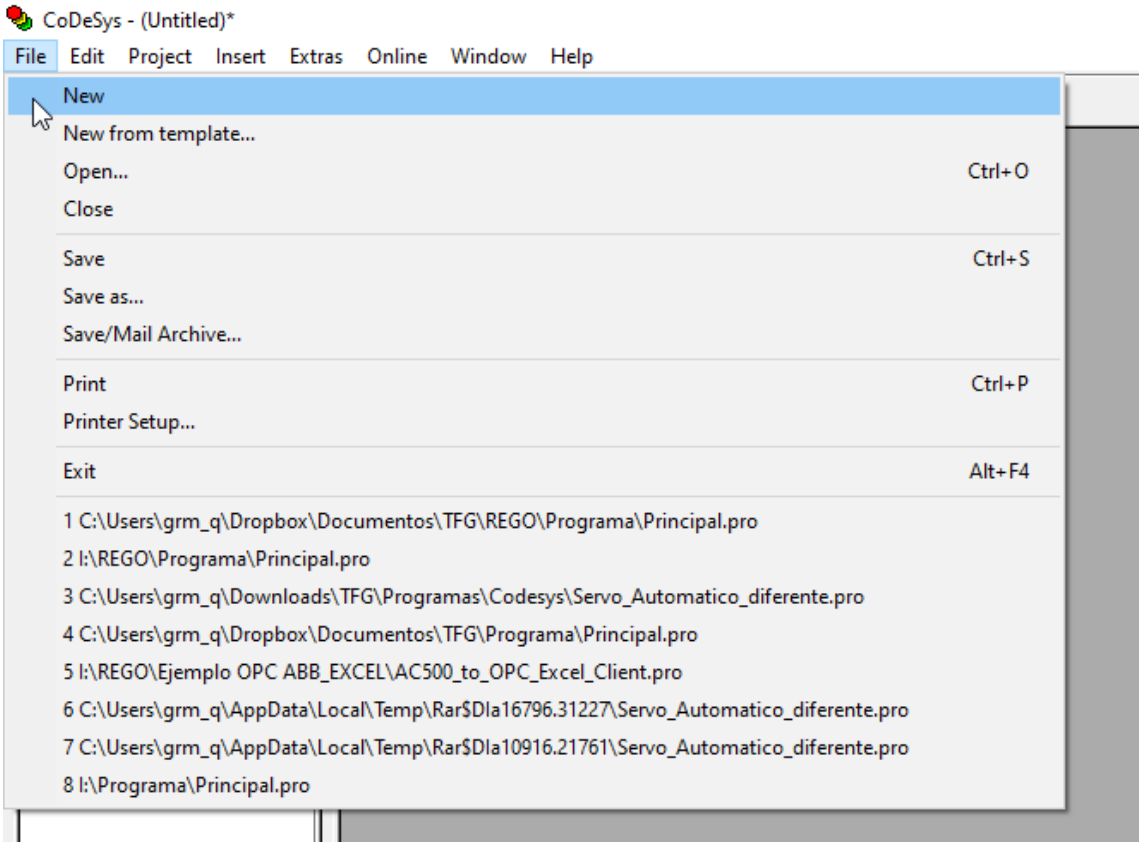


Figura 4.4.1 Crear nuevo proyecto

Tras hacer este paso nos saldrá una ventana donde deberemos seleccionar nuestro autómata, en este caso, el AC500 PM571.

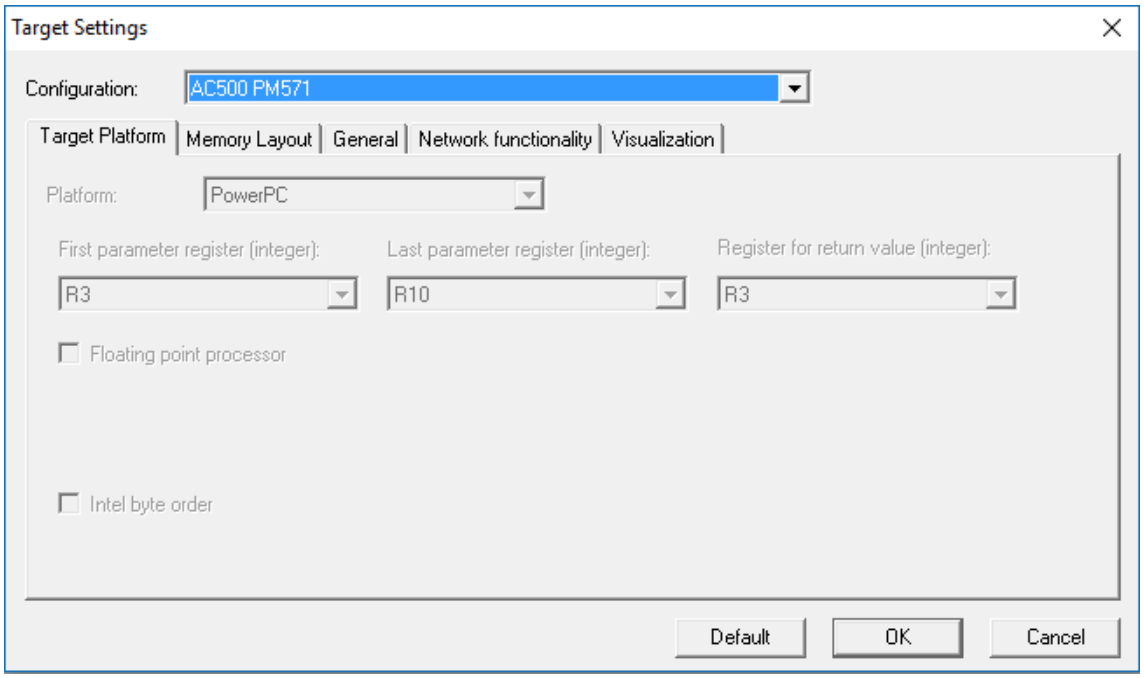


Figura 4.4.2 Elección de autómata

Tras seleccionar nuestro autómata deberemos seleccionar el lenguaje de programación y asignar el nombre del programa que deseemos.

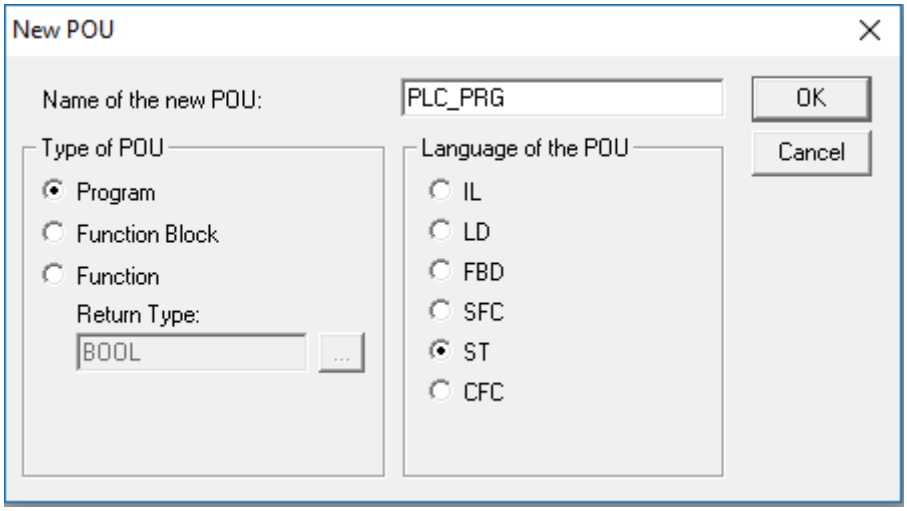


Figura 4.4.3 Elección de lenguaje de programación

Tras elegir el lenguaje de programación estaremos ante el entorno de programación, que dependiendo del lenguaje es de una forma u otra. Explicaremos los entornos de programación en LD y ST.

4.4.1.1 ENTORNO DE PROGRAMACIÓN EN ST

En el lenguaje ST nos encontramos el entorno de programación de la figura 4.1.4.

Figura 4.4.4 Entorno de programación en ST

Como se puede observar en la imagen ahora nos aparecen 4 apartados, el de la izquierda donde sale nuestro programa y podemos cambiar entre 4 pestañas, donde entre otras cosas podremos programar un SCADA interno de codesys o cambiar los recursos que usemos para el programa. Arriba tenemos las variables que hemos declarado y su tipo, que o bien se pueden añadir en la misma ventana o cuando programemos nos saldrá una ventana para declararlas y escoger de que tipo de variable se trata. El espacio blanco del centro de la pantalla, el más grande, es donde escribiremos las líneas de código de nuestro programa. Y el de abajo, es para una vez compilemos el programa, mostrarnos los errores y las alertas que encuentre codesys en nuestro código.

4.4.1.2 ENTORNO DE PROGRAMACIÓN EN LD

En el lenguaje LD nos encontramos el entorno de la figura 4.1.5

Figura 4.4.5 Entorno de programación en LD

Como observamos, el entorno es muy parecido al de ST simplemente nos cambia el apartado del medio ya que en LD programamos con contactos y bobinas. Para introducir estos elementos en nuestro programa tenemos los botones de la figura 4.1.6 para elegir lo que queramos introducir en nuestro programa.

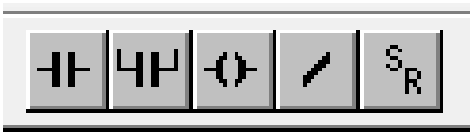


Figura 4.4.6 Botones en programación en LD

Para introducir elementos como temporizadores o contadores debemos irnos a la pestaña *Insert* y seleccionar *Function Block*.

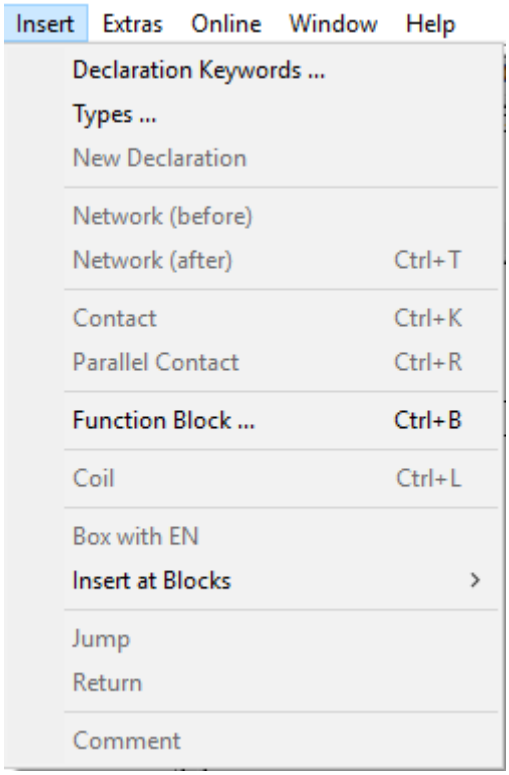


Figura 4.4.7 Añadir elementos

4.4.1.3 ENTORNO SCADA

En CoDeSys además podremos realizar una representación visual mediante SCADA. Para ello deberemos entrar en la pestaña *Visualizations.* Para empezar, deberemos crear un nuevo objeto haciendo click derecho sobre *Visualizations* y eligiendo *Add Object.*

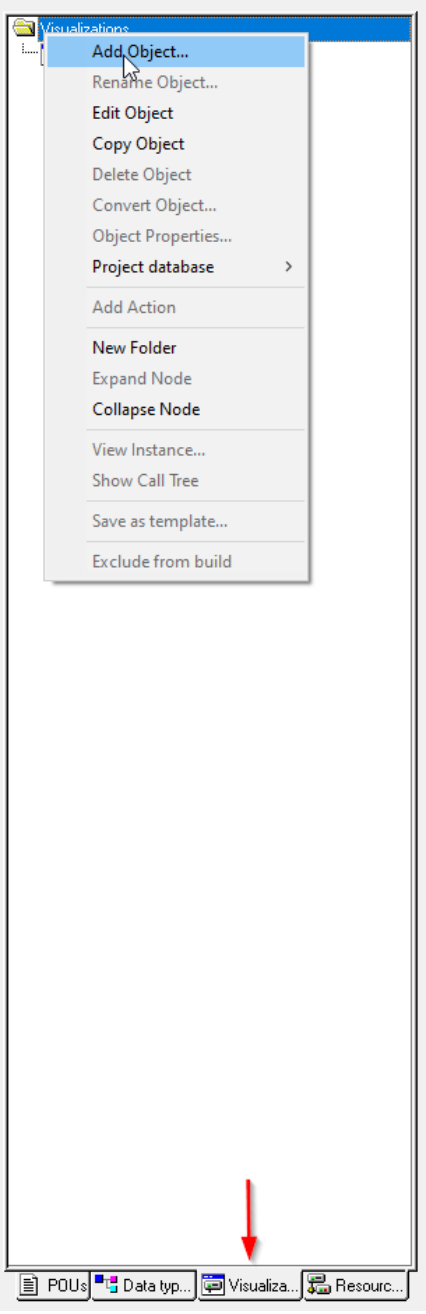


Figura 4.4.8 Añadir SCADA

Una vez hechos los pasos anteriores estaremos en el entorno de nuestro SCADA.

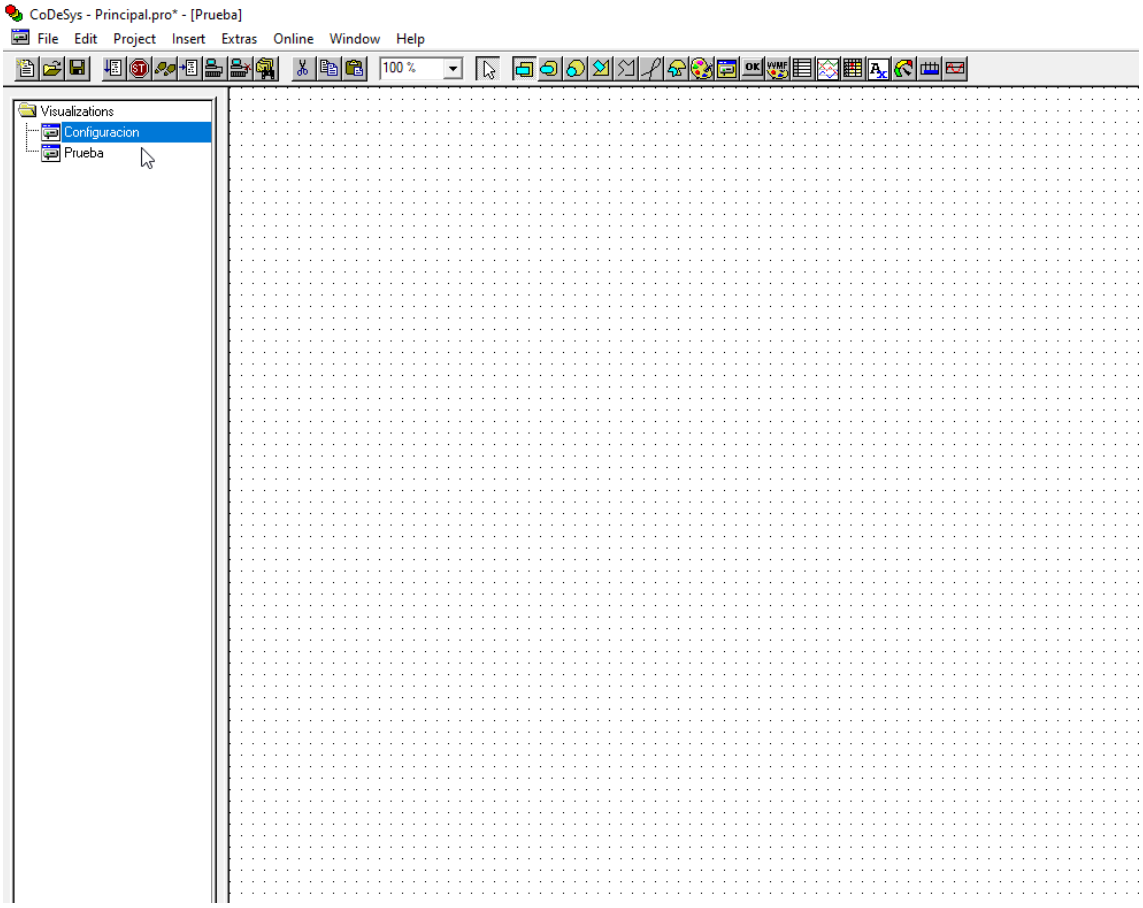


Figura 4.4.9 Entorno SCADA

Podremos añadir objetos para nuestra representación tales como polígonos, botones, tablas, gráficos, imágenes, medidores y elementos de ActiveX a través de los iconos de la figura 4.1.10.

Figura 4.4.10 Iconos para añadir elementos en SCADA

4.4.1.4 CONFIGURACIÓN DEL PLC

Para configurar nuestro PLC y añadir los módulos del PLC que va a usar nuestro programa deberemos irnos a la pestaña *Resources*, entrar en la opción *PLC Configuration* y hacer click derecho en el lugar correspondiente al que queramos añadirle un módulo o encima del elemento en el que queramos realizar una acción.

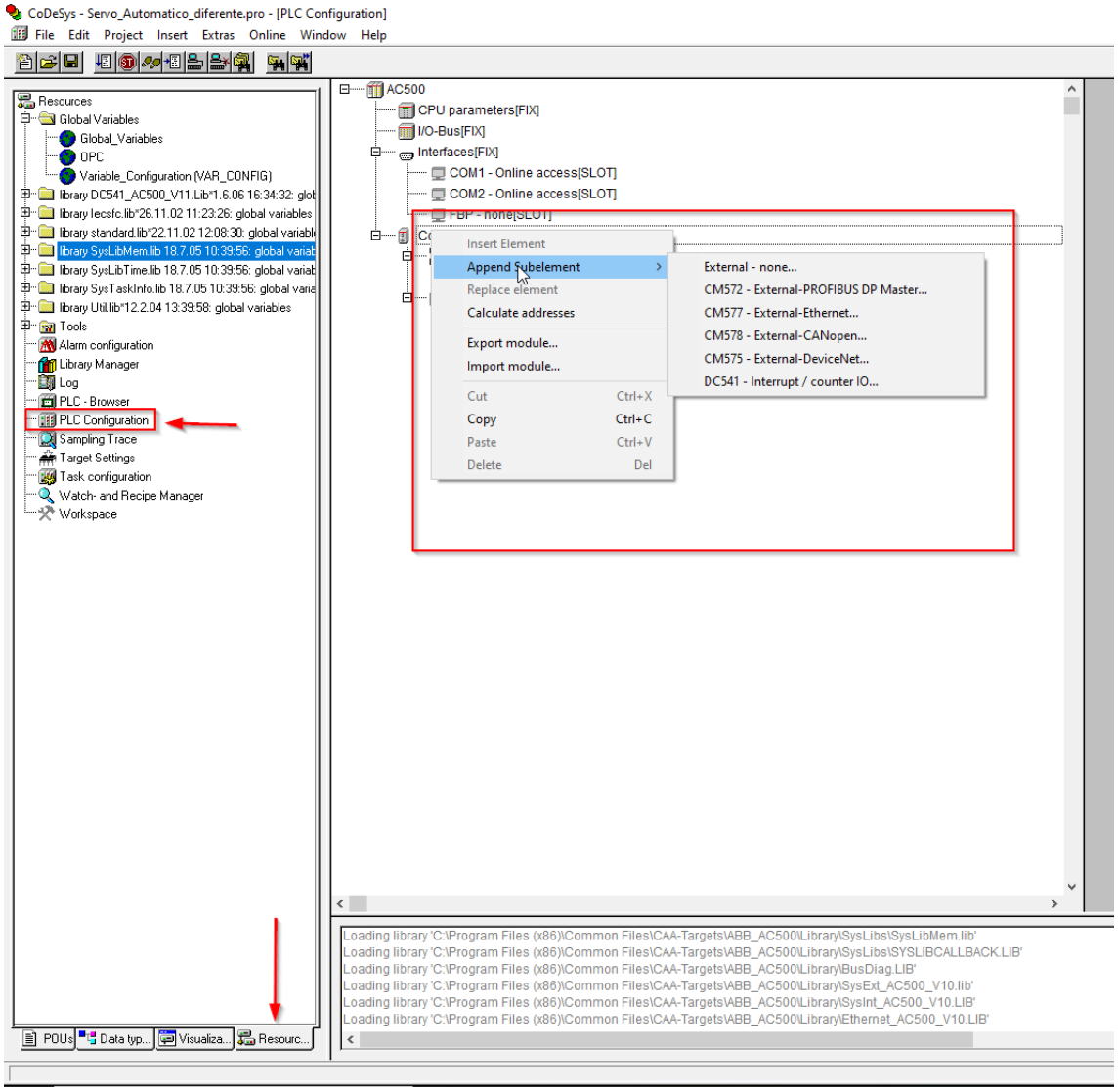


Figura 4.4.11 Añadir módulos

4.4.2 SIMULACIÓN Y PRUEBA DE PROGRAMAS

Para realizar una simulación de nuestro autómata, primero le damos a la pestaña *Online* y seleccionamos *Simulation mode*.

Figura 4.4.12 Entrar en Modo Simulación

Tras esto, en la misma pestaña seleccionamos *Login*.

Ahora tendremos un entorno distinto, uno como el de la figura 4.4.13

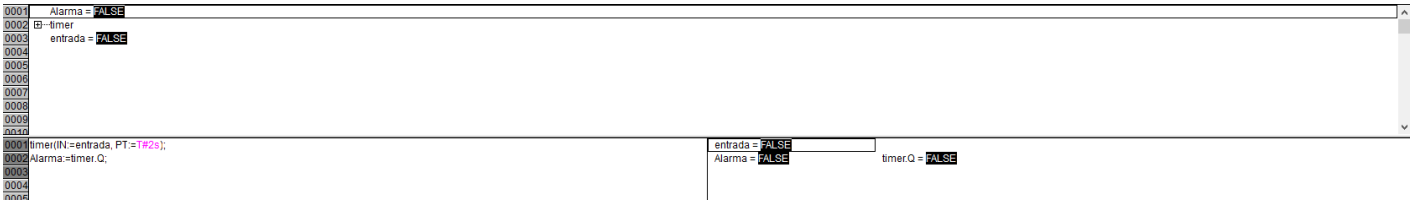


Figura 4.4.13 Entorno de Simulación

Le damos a F5, o a *Run* también en la pestaña *Online*. Luego de realizar este paso podremos darles valores a las variables, por ejemplo, en una variable BOOL si le hacemos doble click y pulsamos F7 le cambiamos el valor a TRUE o FALSE. Si alguna variable está condicionada a otros valores, cuando la condición se cumpla automáticamente cambiará el valor de la variable y así podremos asegurarnos de que el programa funciona correctamente, ya que observaremos si los valores cambian cuando deben, etc.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

CONTROL DE POSICIÓN DE UN PARKING AUTOMÁTICO

ANEXO 5. CÓMO CREAR UN PROYECTO EN CX-SUPERVISOR

SEPTIEMBRE DE 2019

GUILLEM REGO MÁÑEZ

RUBÉN PUCHE PANADERO Y ÁNGEL SAPENA BAÑÓ

Tabla de contenido

4.5.1 CONFIGURACIÓN Y CREACIÓN DE UN NUEVO PROYECTO

Para crear un nuevo proyecto debemos dirigirnos a la pestaña *Archivo,* seleccionar *Nuevo Proyecto* y seleccionar *Proyecto CX-Supervisor Plus.*

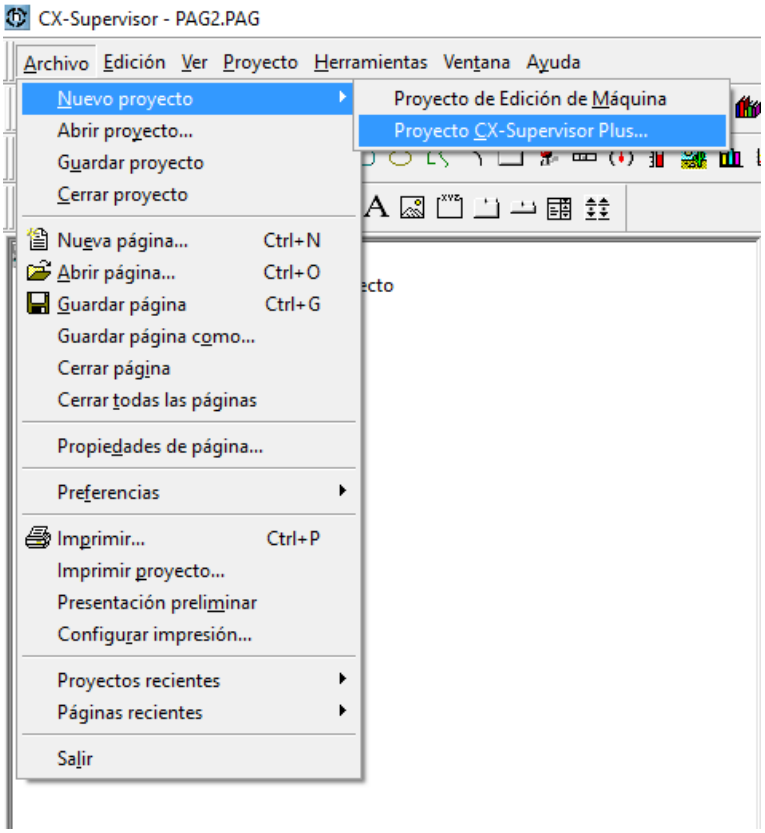


Figura 4.5.1 Creación Nuevo Proyecto

Tras hacer este paso nos saldrá una ventana donde deberemos seleccionar la ubicación en la que queremos almacenar nuestro archivo y asignarle un nombre.

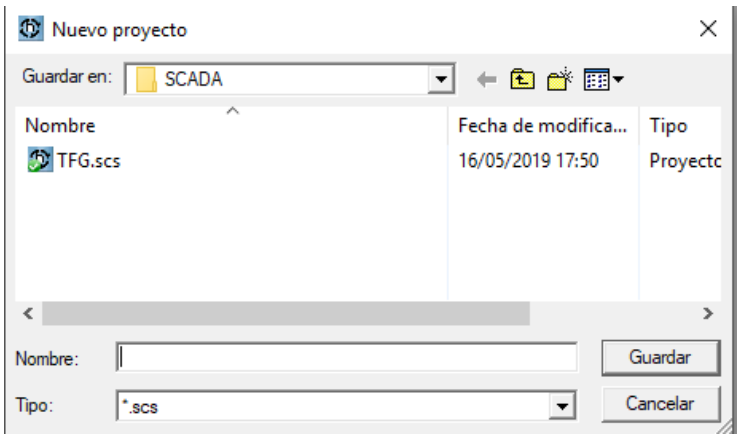


Figura 4.5.2 Asignación de nombre

4.5.1.1 ENTORNO DEL PROGRAMA

Tras seleccionar la ubicación nos encontraremos ante el entorno donde haremos nuestro SCADA.

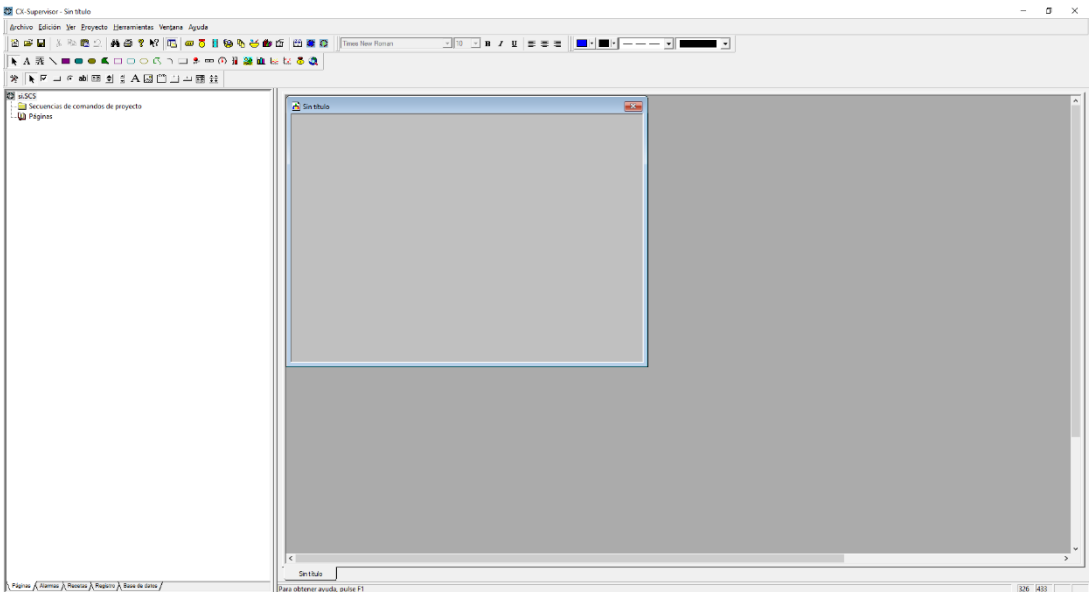


Figura 4.5.3 Entorno CX-Supervisor

Como se puede observar en la imagen podemos diferenciar 2 áreas principales, el área izquierda, donde tenemos una lista con fondo blanco nos servirá para agregar páginas y movernos entre estas. La zona con fondo gris será donde veamos esas páginas y editemos tanto esas páginas como sus contenidos u objetos.

También encontramos una barra de herramientas donde tendremos múltiples opciones para personalizar nuestro proyecto, algunas de estas son por ejemplo: añadir un botón, añadir un objeto o abrir el editor de animaciones. Estas opciones serán detalladas en este mismo anexo en apartados posteriores.

4.5.1.2 CREACIÓN Y EDICIÓN DE PÁGINAS

Para agregar una página a nuestro SCADA deberemos hacer click derecho en el área izquierda de nuestro entorno y seleccionar *Nueva página.*

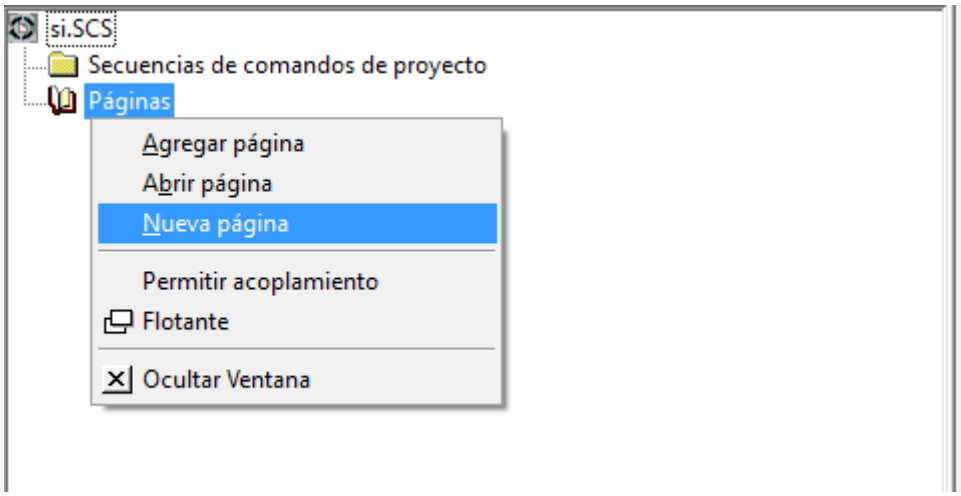


Figura 4.5.4 Crear página

Al hacer click nos creará instantáneamente una página en nuestro proyecto.

4.5.1.2.1 PERSONALIZACIÓN DE PÁGINAS

Para personalizar nuestras páginas deberemos hacer doble click en el fondo de cada una de ellas y se nos abrirá la ventana de la figura 4.5.5.

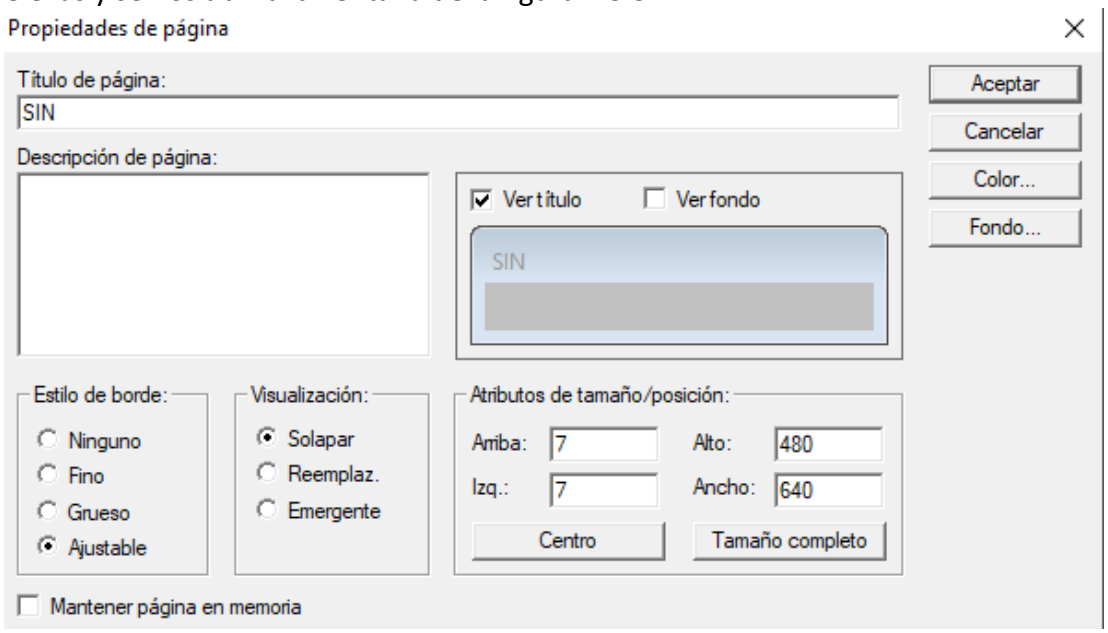


Figura 4.5.5 Personalización de páginas

Como podemos observar, en la ventana anterior podremos cambiar el título de la página, añadirle una descripción, cambiarle el color de fondo y seleccionar la visibilidad tanto del título como del fondo. Además, podremos seleccionar el estilo de borde, la visualización de la página y el tamaño y posición de la misma.

4.5.1.2.2 ADICIÓN DE OBJETOS

A la hora de añadir objetos tenemos principalmente 6 opciones a parte de añadir texto que lo podremos hacer haciendo click en el icono de la figura 4.5.6.

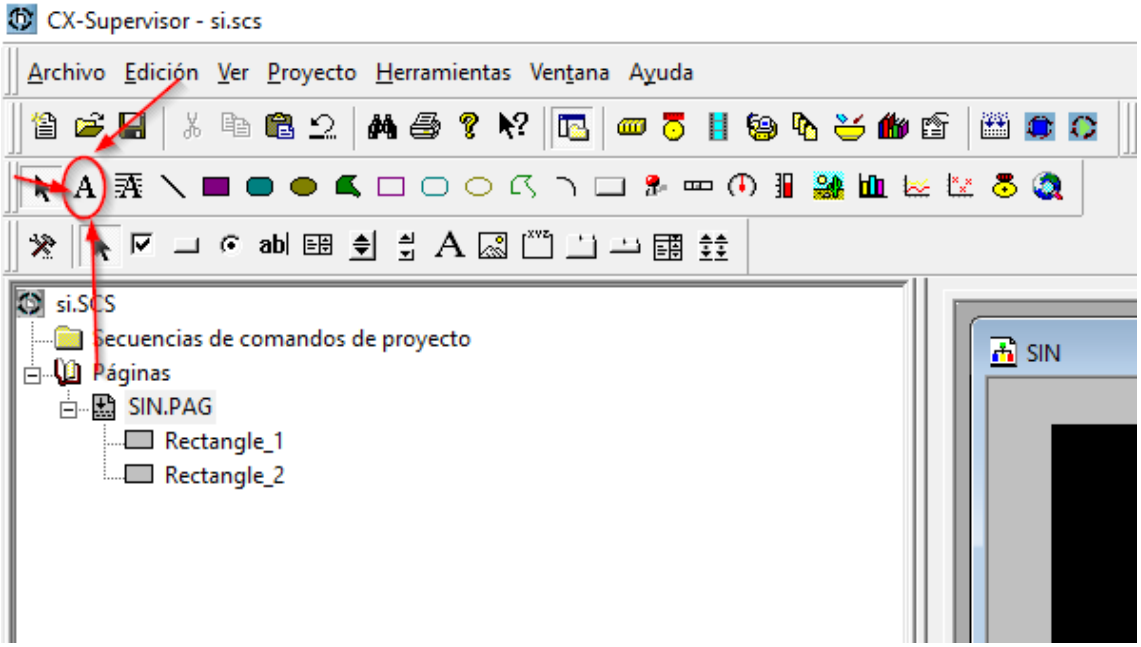


Figura 4.5.6 Añadir texto

La primera opción es añadir una forma con fondo relleno o vacía. Las opciones son: rectángulo, rectángulo redondeado, elipse, polígono y arco. Para añadir una de estas formas tendremos que hacer click en alguno de los iconos de la figura 4.5.7.

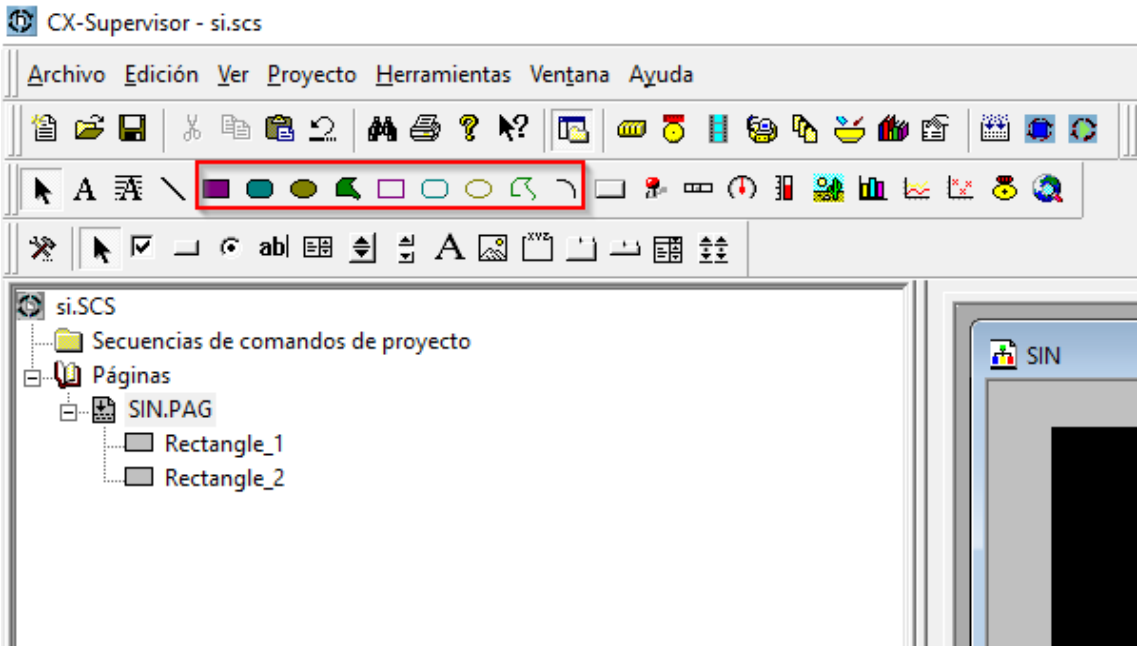


Figura 4.5.7 Añadir polígonos

En la figura 4.5.8 podemos observar une ejemplo de este tipo de objetos.

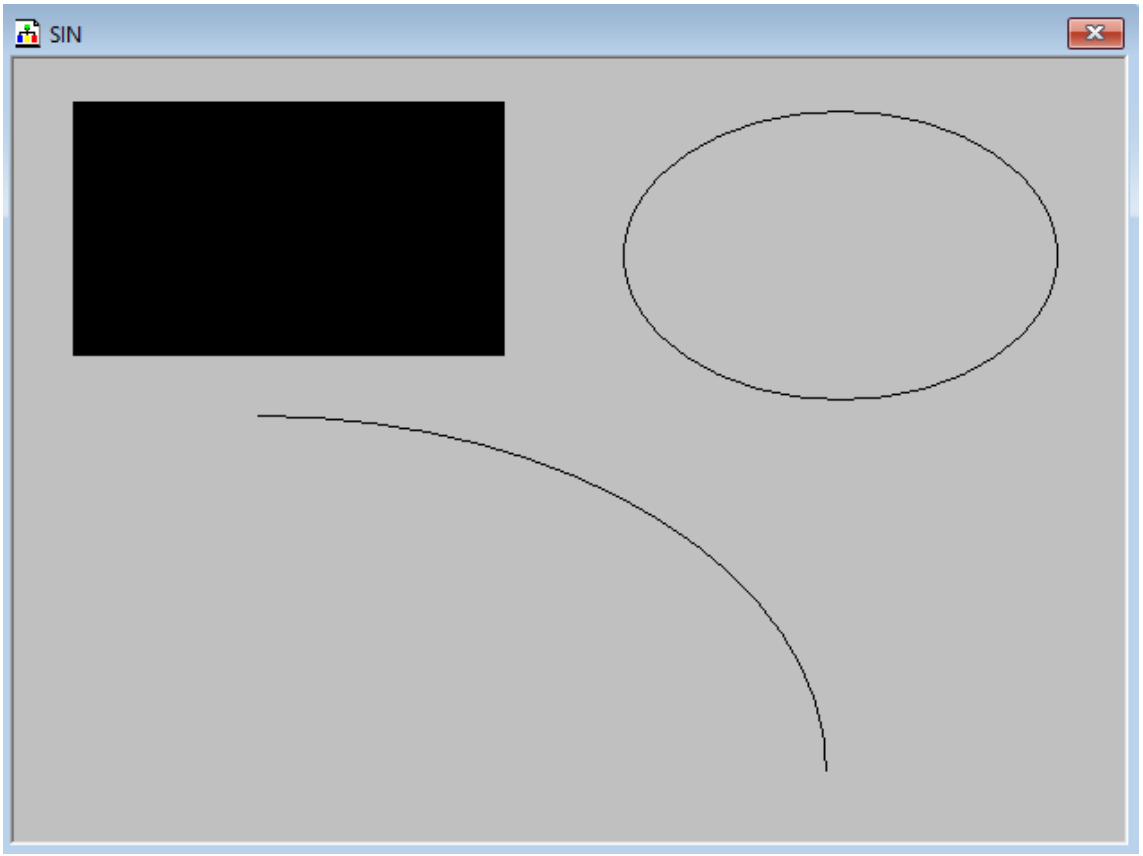


Figura 4.5.8 Ejemplo polígonos

La segunda opción es añadir botones o controles deslizantes para principalmente editar o variar el valor de un punto, o para ejecutar una secuencia de comandos. Para añadirlos deberemos hacer click en alguno de los iconos de la figura 4.5.9.

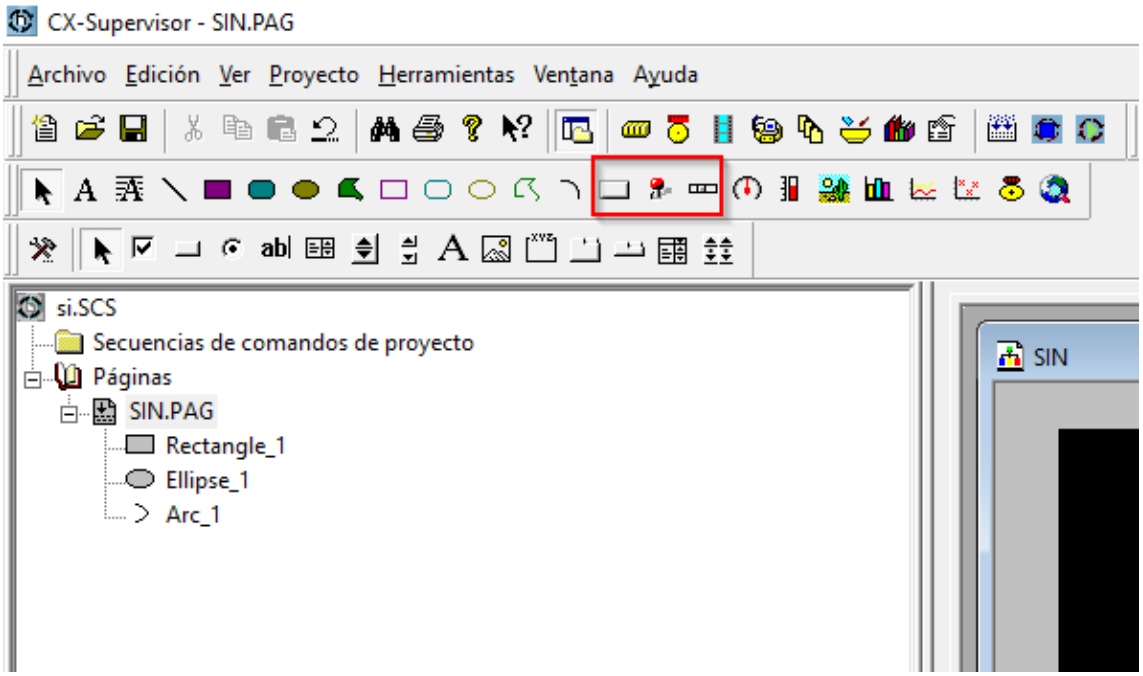


Figura 4.5.9 Añadir botones o controles deslizantes

En la figura 4.5.10 podemos observar une ejemplo de este tipo de objetos.

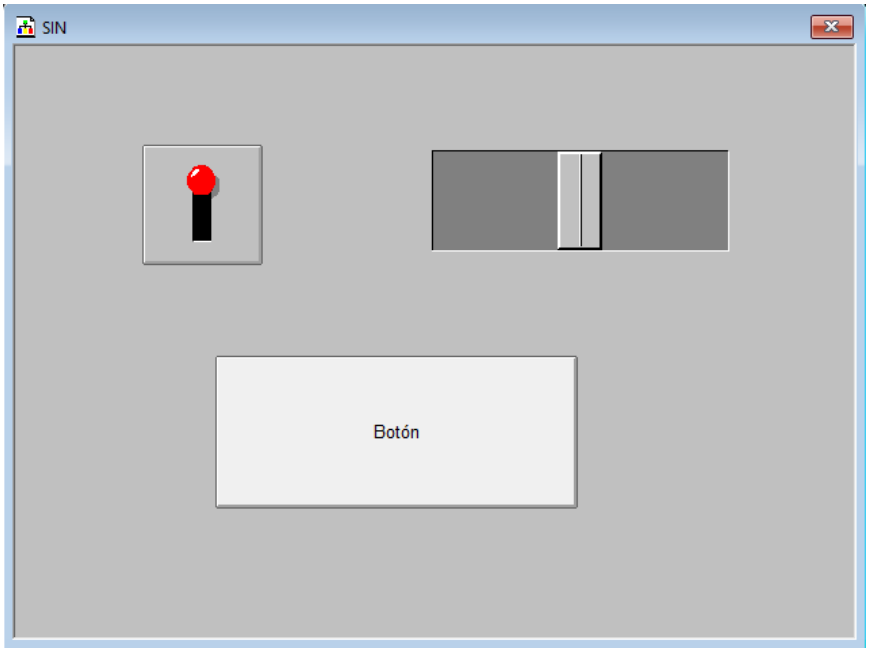


Figura 4.5.10 Ejemplo botones y control deslizante

La tercera opción que tenemos es añadir medidores que nos servirá como dice su nombre para medir y visualizar el valor del punto deseado, ya sea del sistema o propio del proyecto. Para añadir un medidor deberemos seleccionar uno de los dos iconos que vemos en la figura 4.5.11.

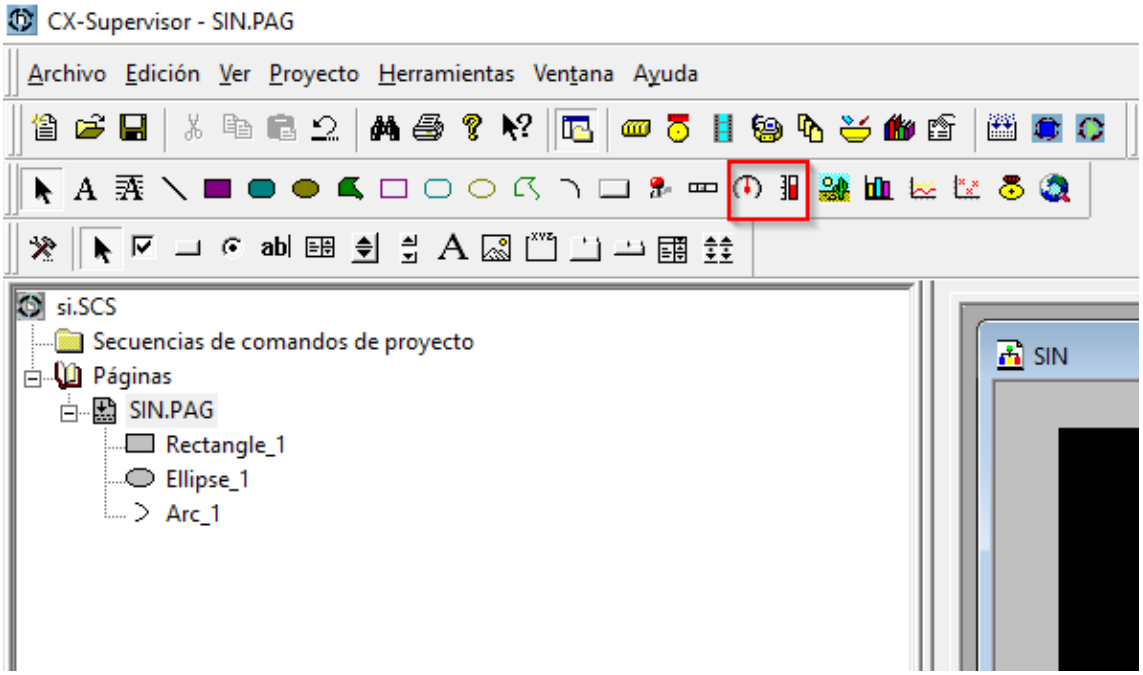


Figura 4.5.11 Añadir medidores

Podemos observar un ejemplo de estos objetos en la figura 4.5.12.

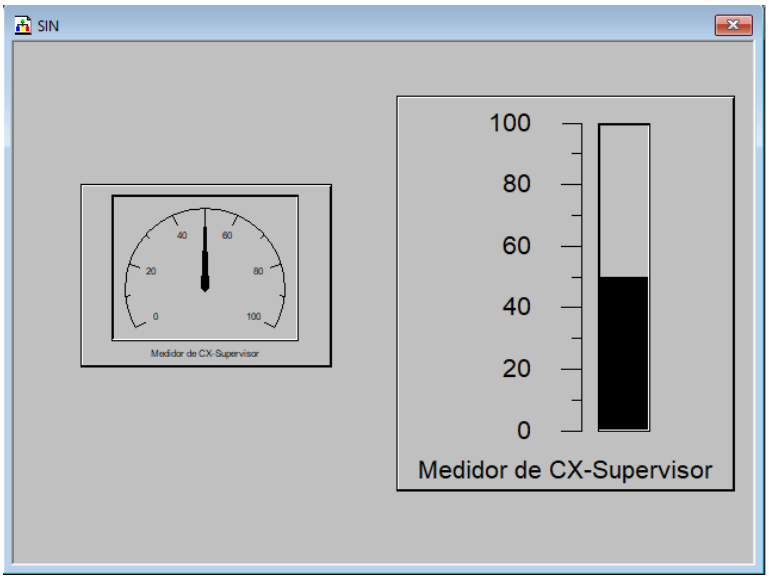


Figura 4.5.12 Ejemplo de medidores

La siguiente opción se trata de añadir una imagen. Se puede hacer mediante dos vías, mediante el botón resaltado en la figura 4.5.13 en el que podremos introducir una imagen propia o bien mediante la biblioteca de gráficos pulsando las teclas CTRL y L.

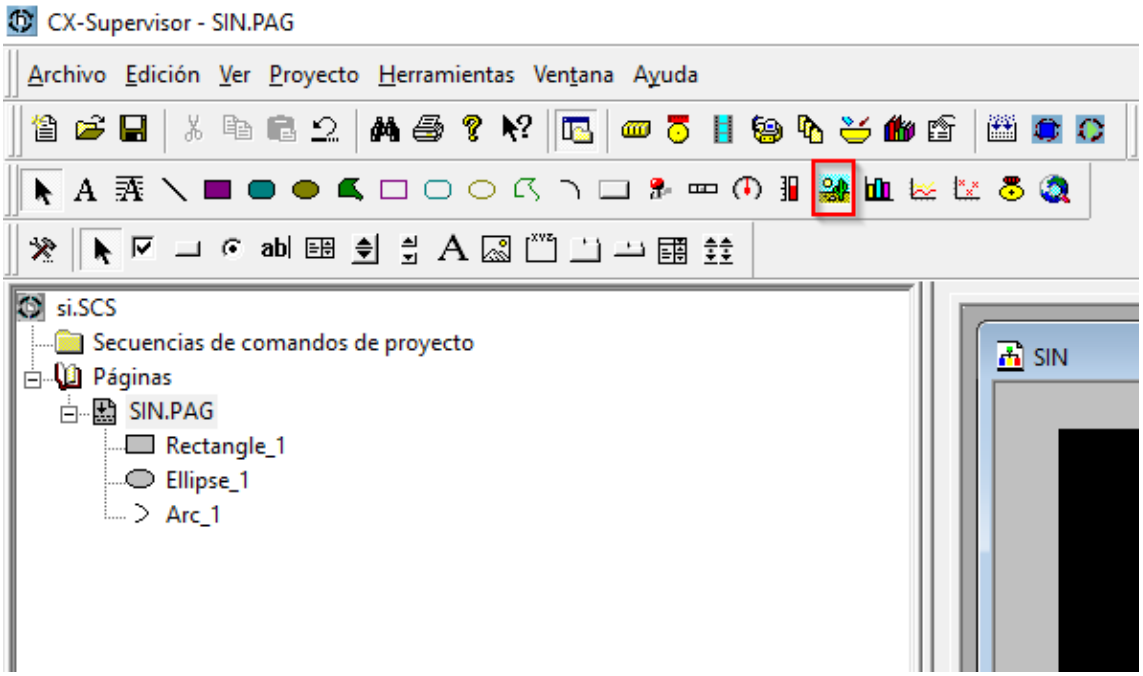


Figura 4.5.13 Añadir imagen

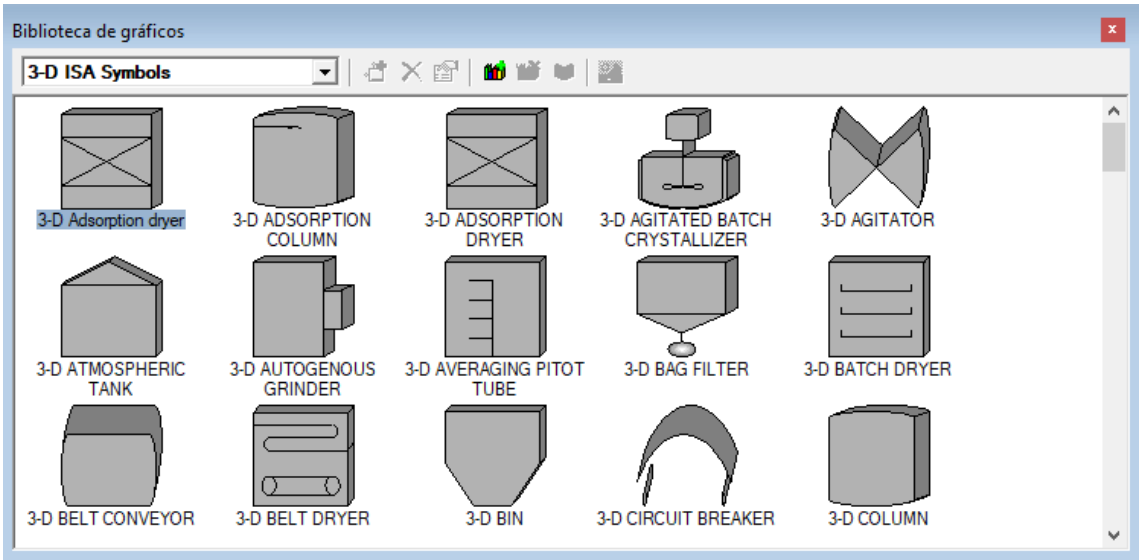


Figura 4.5.14 Biblioteca de gráficos

En la figura 4.5.15 podemos observar un ejemplo de estos objetos.

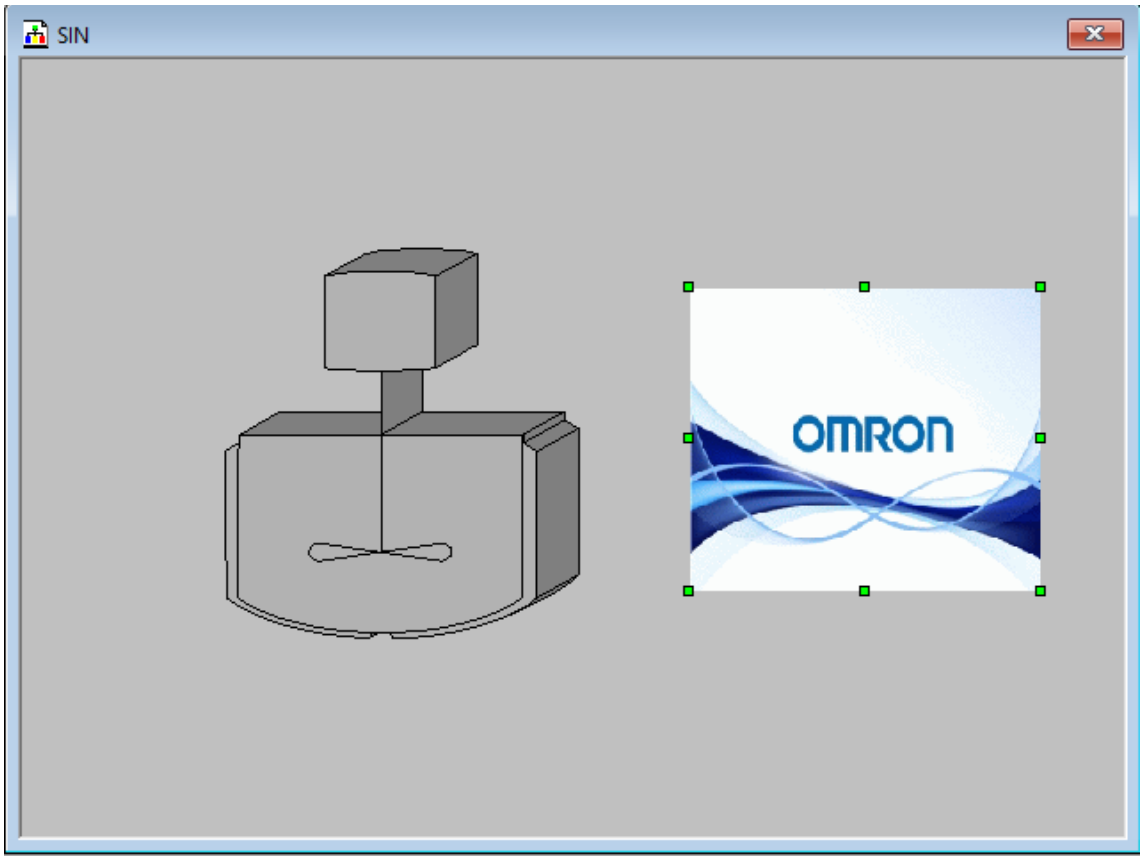


Figura 4.5.15 Ejemplo imagen y objeto de la biblioteca de gráficos

La quinta opción se trata de añadir un gráfico en la que tenemos 3 variedades: un diagrama, un gráfico de tendencia y un gráfico de dispersión. Para añadir este tipo de objetos tenemos que hacer click en uno de los 3 iconos que vemos resaltados en la figura 4.5.16.

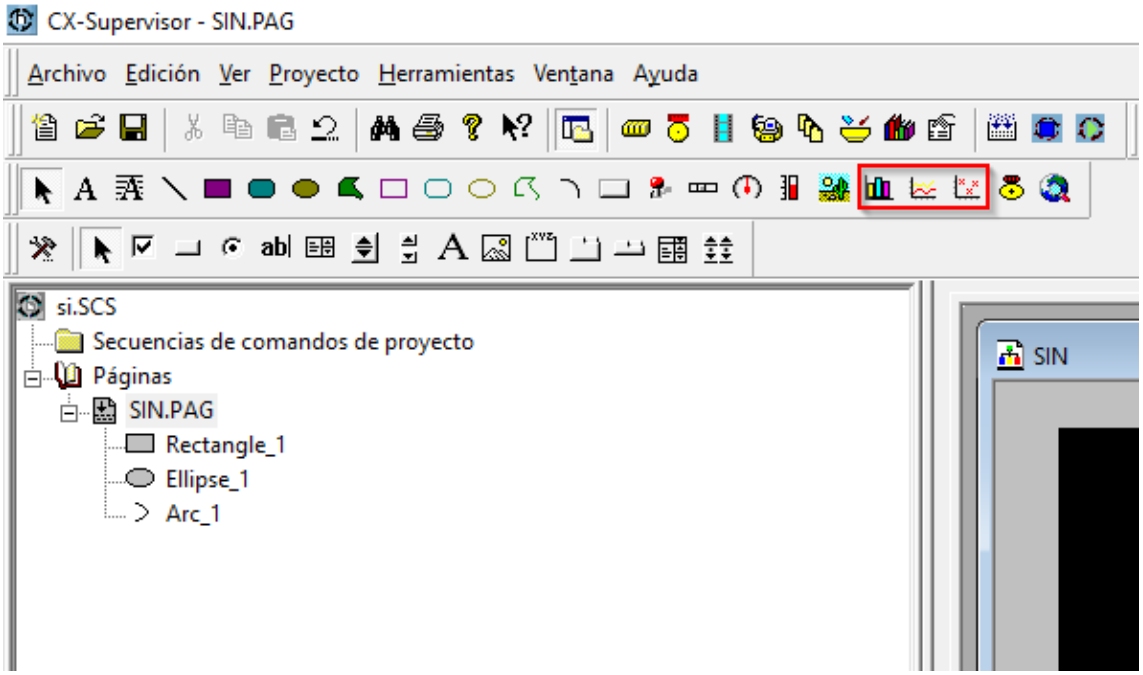


Figura 4.5.16 Añadir gráficos

Podemos observar estos elementos en la figura 4.5.17.

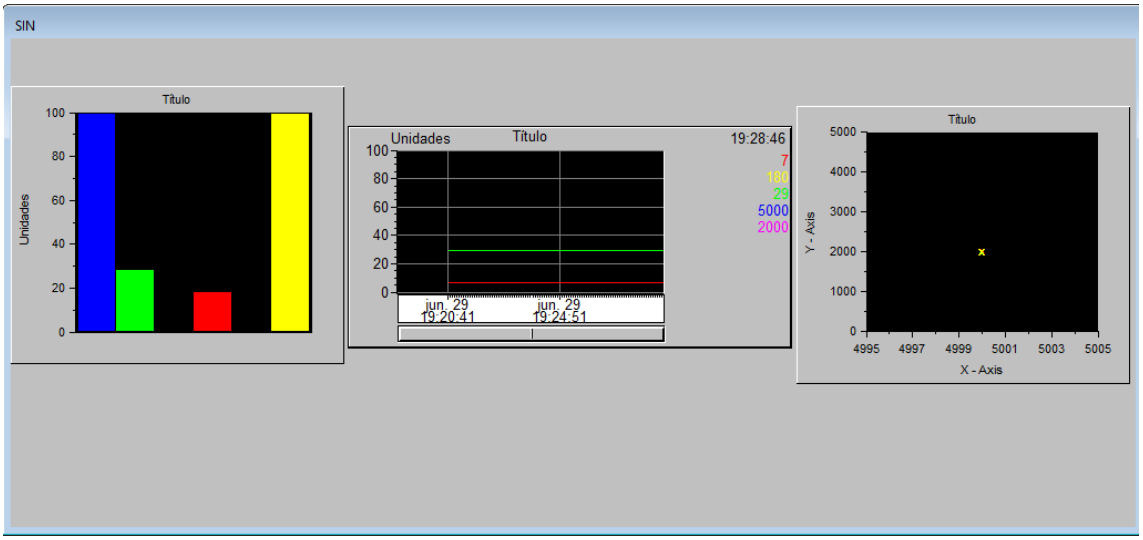


Figura 4.5.17 Ejemplo de gráficos

La última opción realmente se trata de 2 que hemos englobado en una única. Esta opción nos permite visualizar una alarma o una página web, y se puede añadir haciendo click en uno de los iconos señalados en la figura 4.5.18.

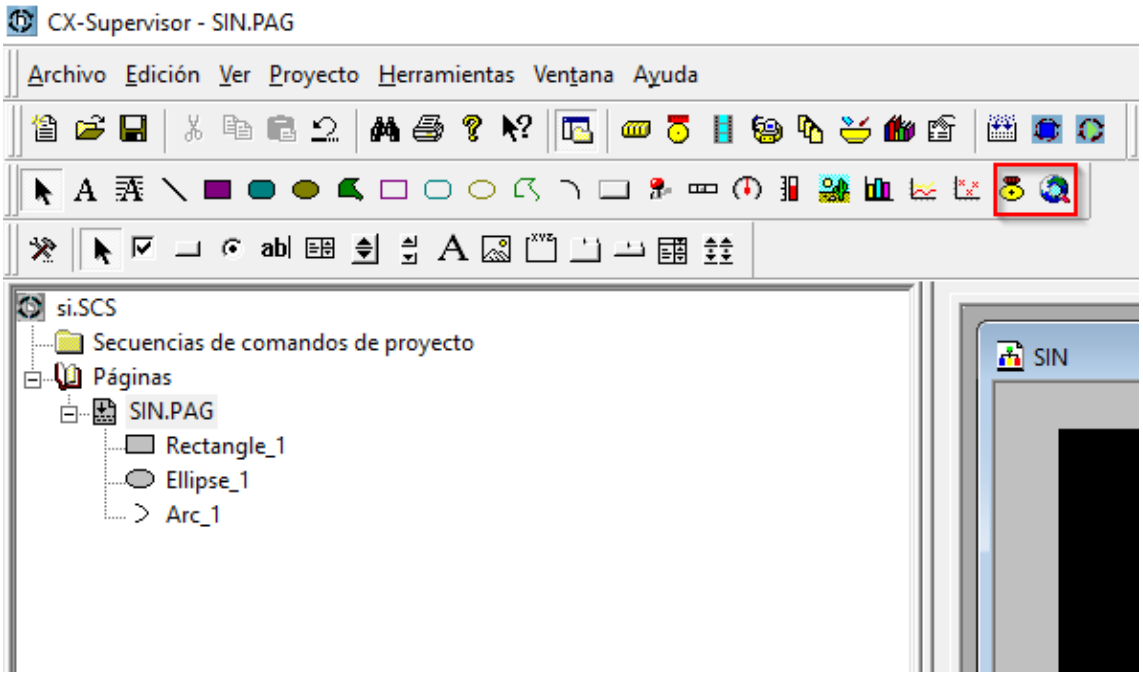


Figura 4.5.18 Añadir alarma o página web

Podemos observar estos elementos en la figura 4.5.19.

Figura 4.5.19 Ejemplo de alarma y página web

4.5.1.2.3 EDICIÓN DE ANIMACIONES

Para editar la animación de un objeto deberemos hacer doble click sobre el objeto deseado y luego hacer click en *Editor de animaciones.* También se puede acceder realizando doble click sobre un objeto.

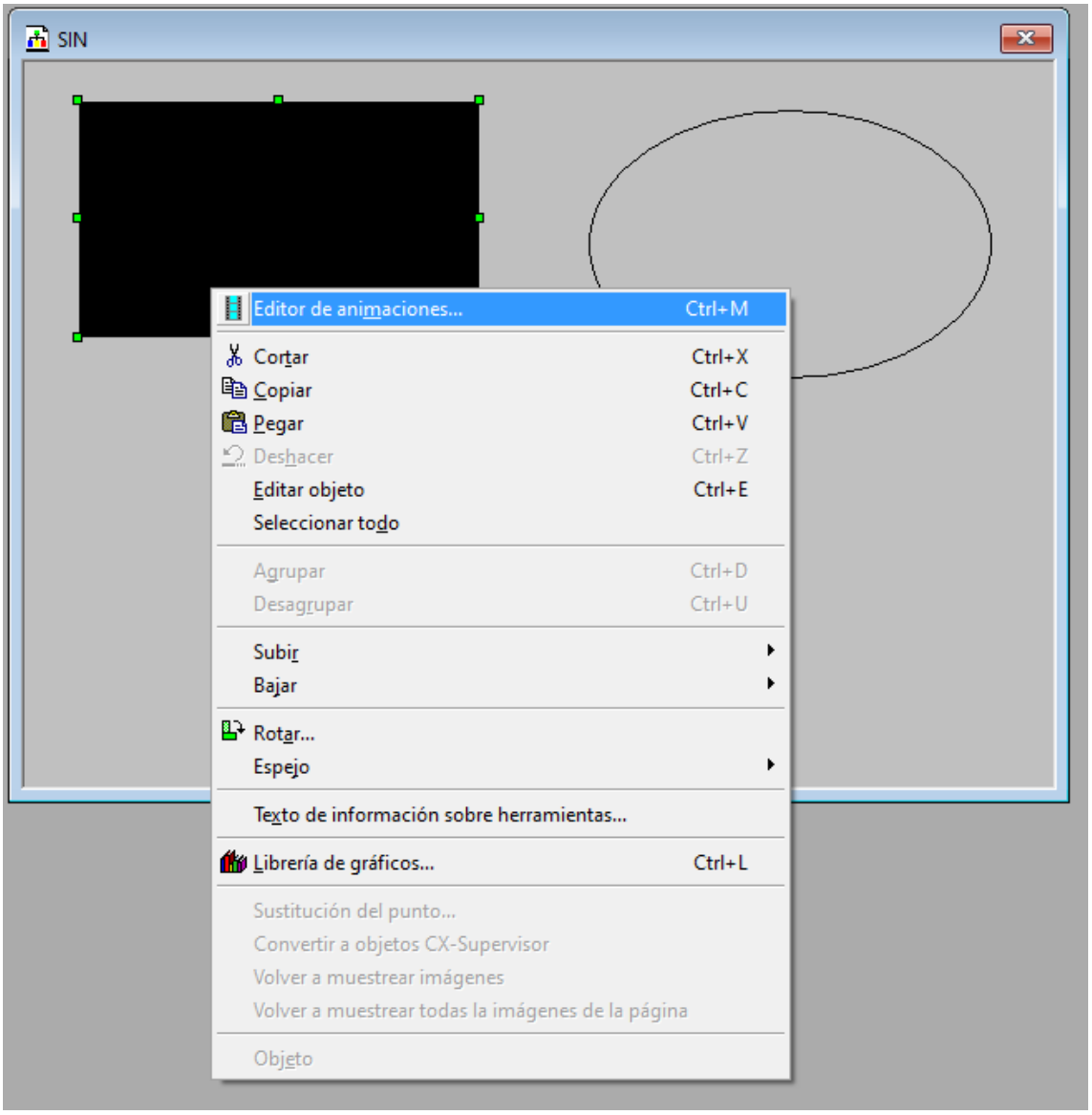


Figura 4.5.20 Entrar en editor de animaciones

Tras haber realizado la acción anterior veremos la ventana de la figura 4.5.21, en la que podremos cambiar el tamaño, color, posición, visibilidad, etc del objeto además de poder cambiar el valor de los puntos, que son nuestras variables, ejecutar una secuencia de comandos y visualizar o cerrar una página.

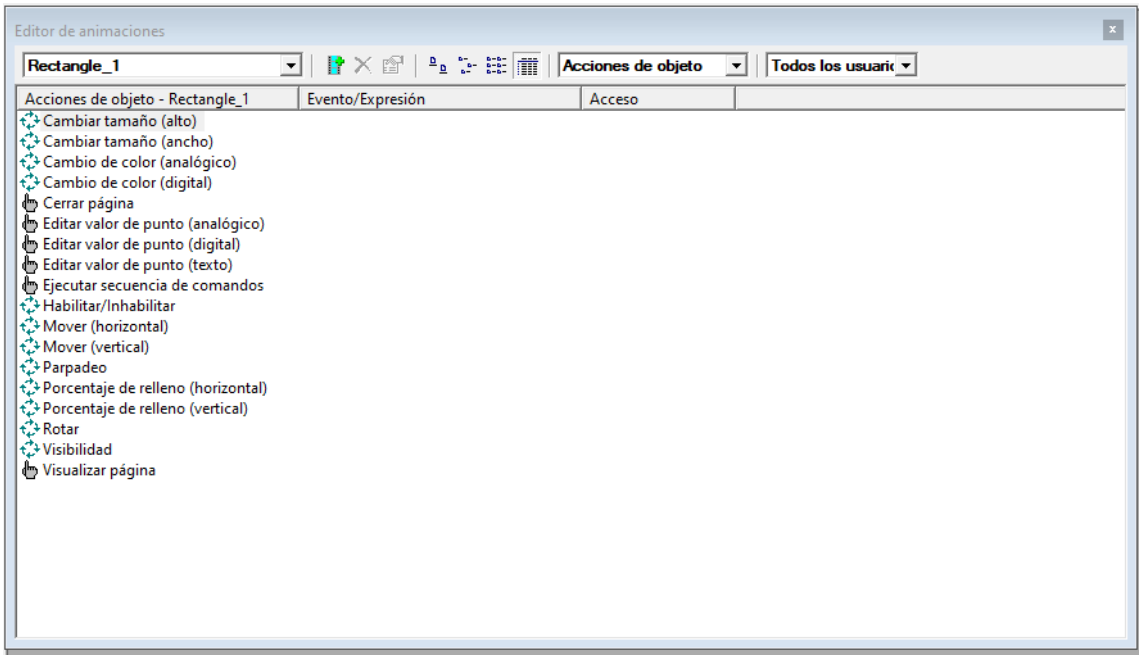


Figura 4.5.21 Editor de animaciones

Desde esa misma ventana podemos acceder también a editar acciones de páginas o del proyecto. Para hacerlo debemos desplegar la lista haciendo click en la flecha a la derecha de donde vemos *Acciones de objeto*.

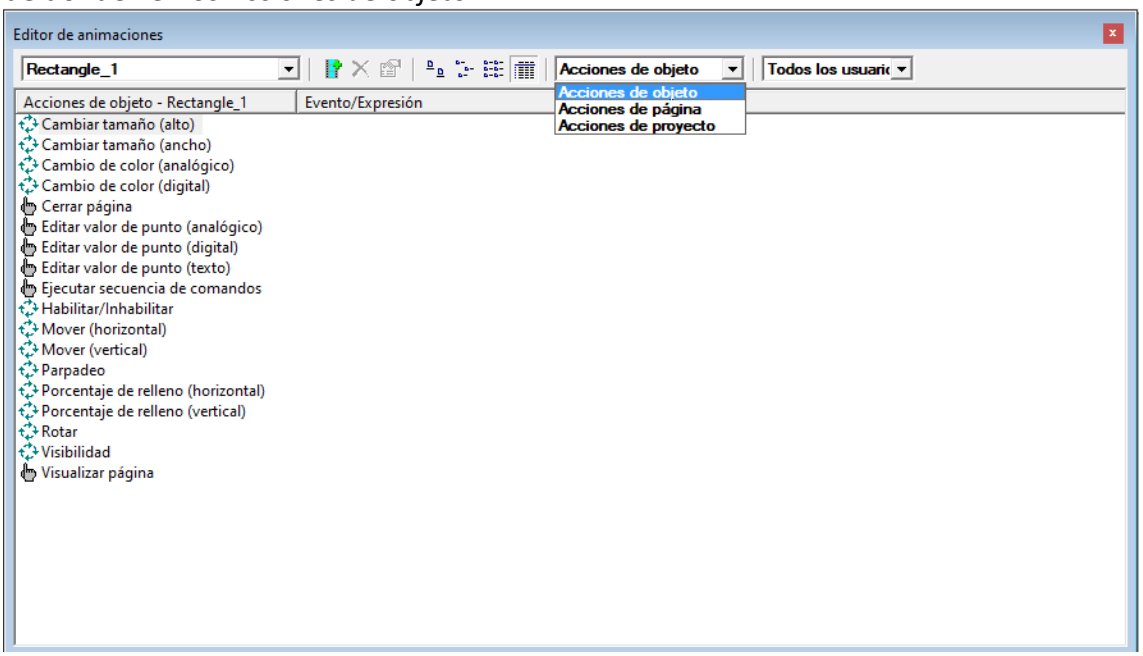


Figura 4.5.22 Cambiar tipo de acciones

Una vez estamos en esa ventana hacemos click en el ícono de al lado del nombre de la página o proyecto y podremos realizar un script para que se ejecute con determinadas condiciones.

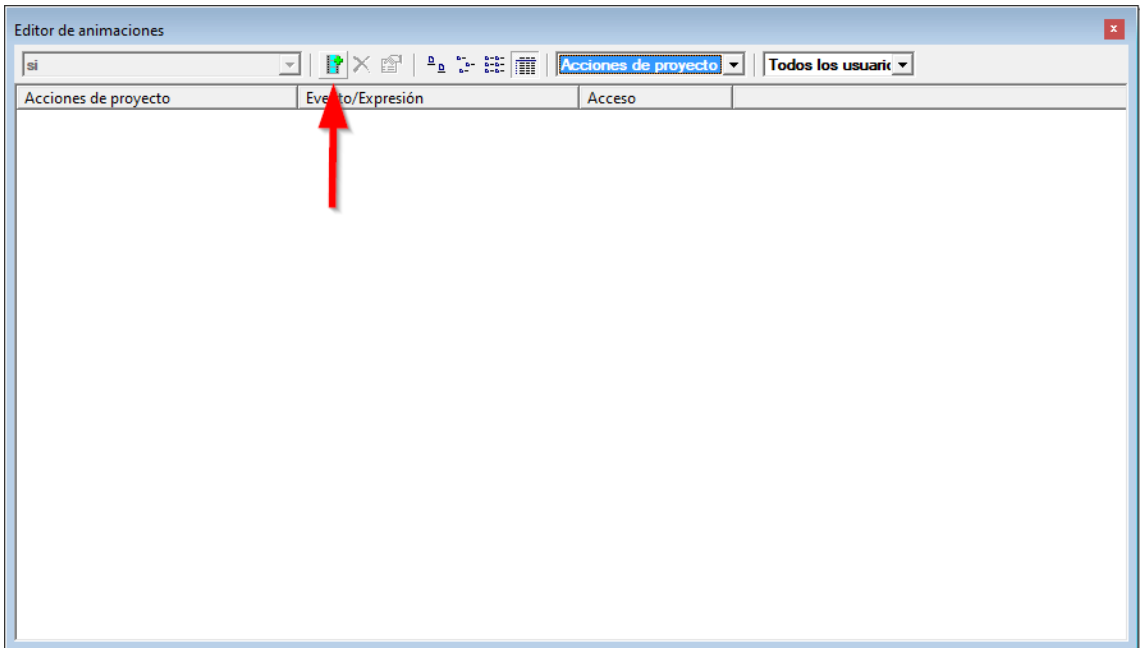


Figura 4.5.23 Añadir Scripts

4.5.1.2.4 EDICIÓN DE SCRIPTS

Mediante las acciones anteriores entramos en la edición de scripts con el entorno de la figura 4.5.24.

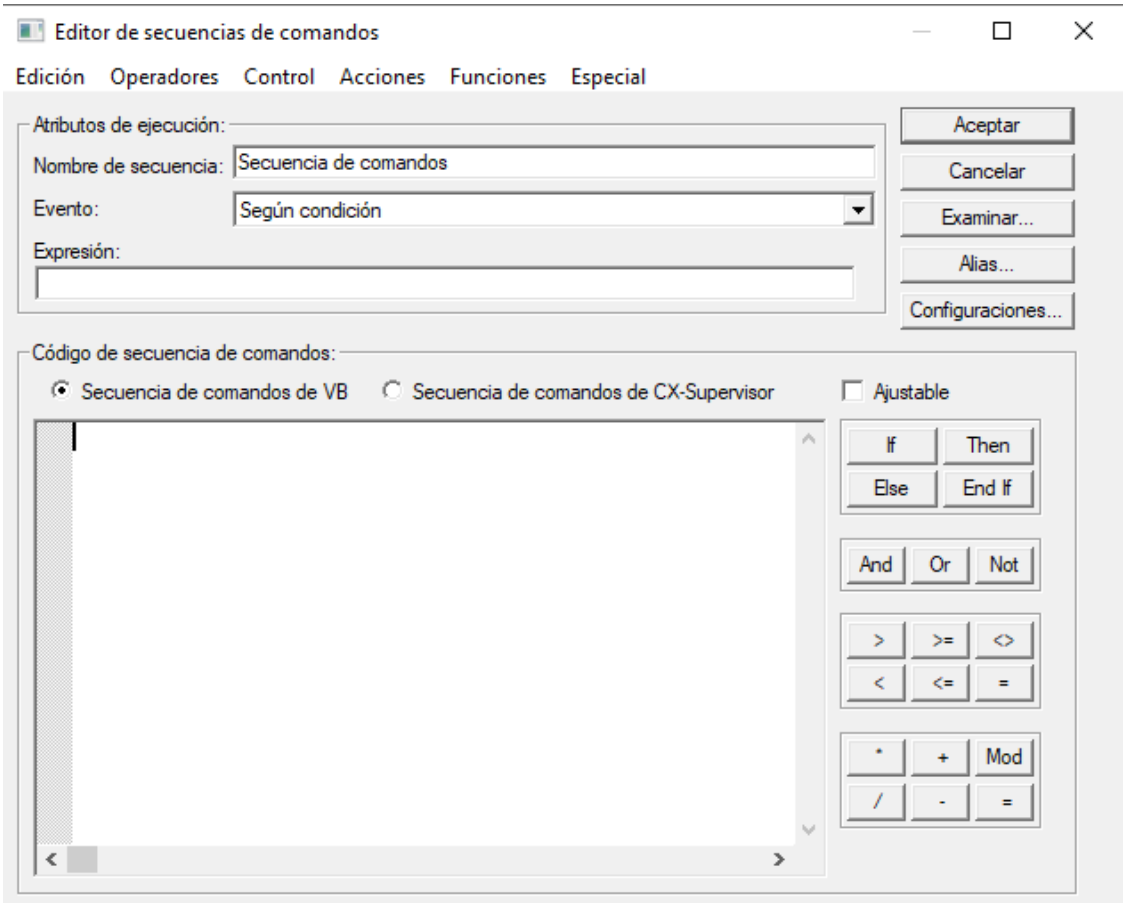


Figura 4.5.24 Entorno de edición de scripts

Como observamos en el entorno podemos escribir en dos lenguajes de programación distintos, *Secuencia de comandos de VB* y *Secuencia de comandos de CX-Supervisor*. Además, por si no conocemos los lenguajes, tenemos varios botones con el condicional IF y su funcionamiento, con las condiciones y operadores. También contamos con pestañas donde tenemos prácticamente todo lo que podemos hacer con los scripts, desde los bucles a acciones, funciones o incluso a acceder a base de datos.

4.5.2 RUNTIME Y PRUEBA DE PROGRAMA

El *Runtime* es el entorno donde ejecutamos nuestro programa y podemos ver sus diferentes funciones y realizar pruebas para comprobar que el programa funciona correctamente.

Para acceder al *Runtime* y ejecutar nuestro programa deberemos hacer click en el icono resaltado de la figura 4.5.25.

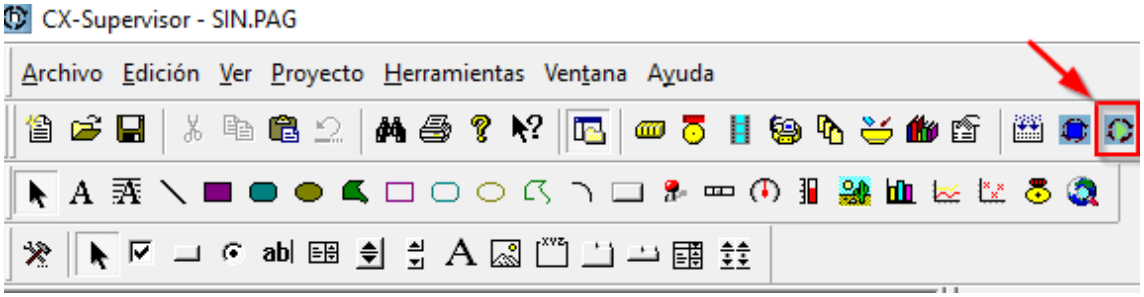


Figura 4.5.25 Ejecutar Runtime

Tras hacerlo nos encontraremos en el *Runtime* que tendrá una apariencia similar a la de la figura 4.5.26, pero con el contenido de nuestro proyecto.

Figura 4.5.26 Runtime

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

CONTROL DE POSICIÓN DE UN PARKING AUTOMÁTICO

ANEXO 6. CONEXIÓN PROGRAMA – SCADA VÍA OPC

SEPTIEMBRE DE 2019

GUILLEM REGO MÁÑEZ

RUBÉN PUCHE PANADERO Y ÁNGEL SAPENA BAÑÓ

Tabla de contenido

4.6.1 CODESYS OPC CONFIGURATOR

El primer paso para conectar nuestro programa con la representación gráfica del SCADA es abrir el *CoDeSys OPC Configurator*. Una vez hecho esto nos encontraremos con el entorno de la figura 4.6.1.

Figura 4.6.1 Entorno CoDeSys

Una vez en el entorno, lo primero que deberemos hacer es añadir nuestro PLC, para ello hacemos click derecho en *Server* y seleccionamos *Append PLC*.

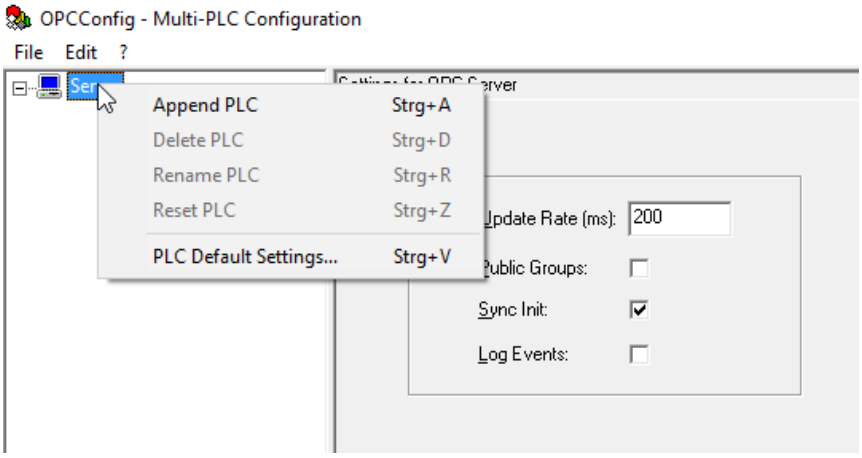


Figura 4.6.2 Añadir PLC

Tras esto deberemos introducir el nombre de nuestro proyecto de CoDeSys en *Project Name* y marcar la opción *Motorola Byteorder.*

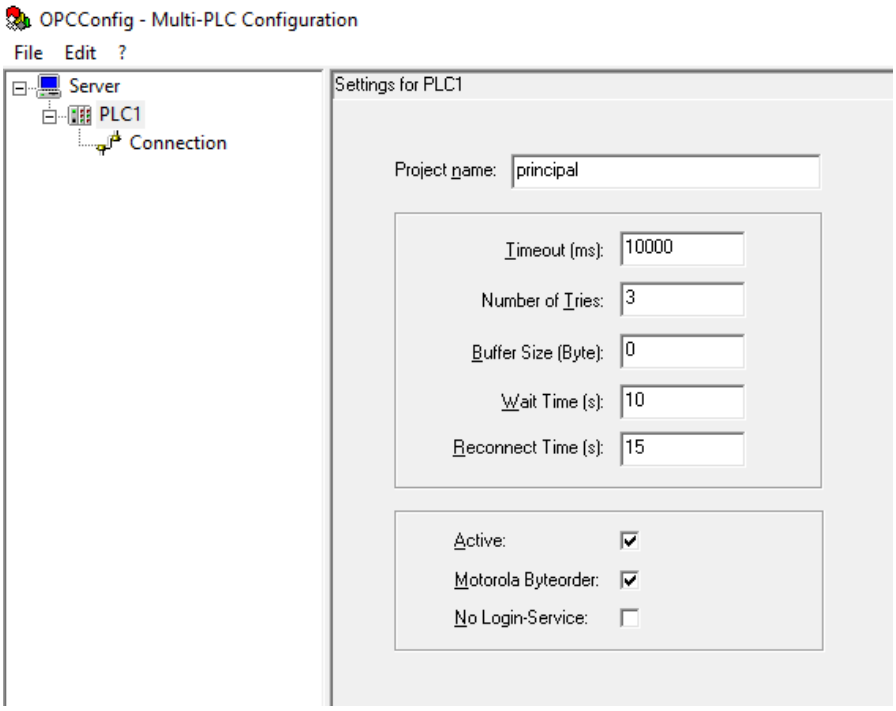


Figura 4.6.3 Opciones PLC

Tras esto seleccionaremos *Connection* y pulsaremos el botón Edit para añadir la conexión al PLC.

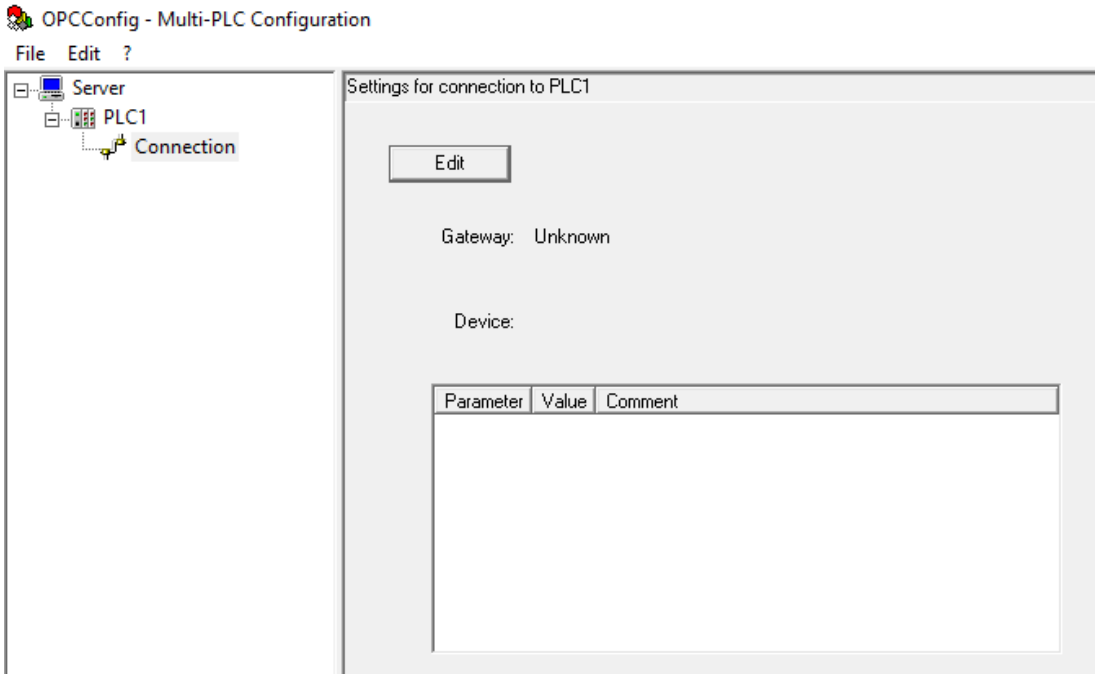


Figura 4.6.4 Opciones conexión

Ahora nos encontraremos ante una ventana nueva.

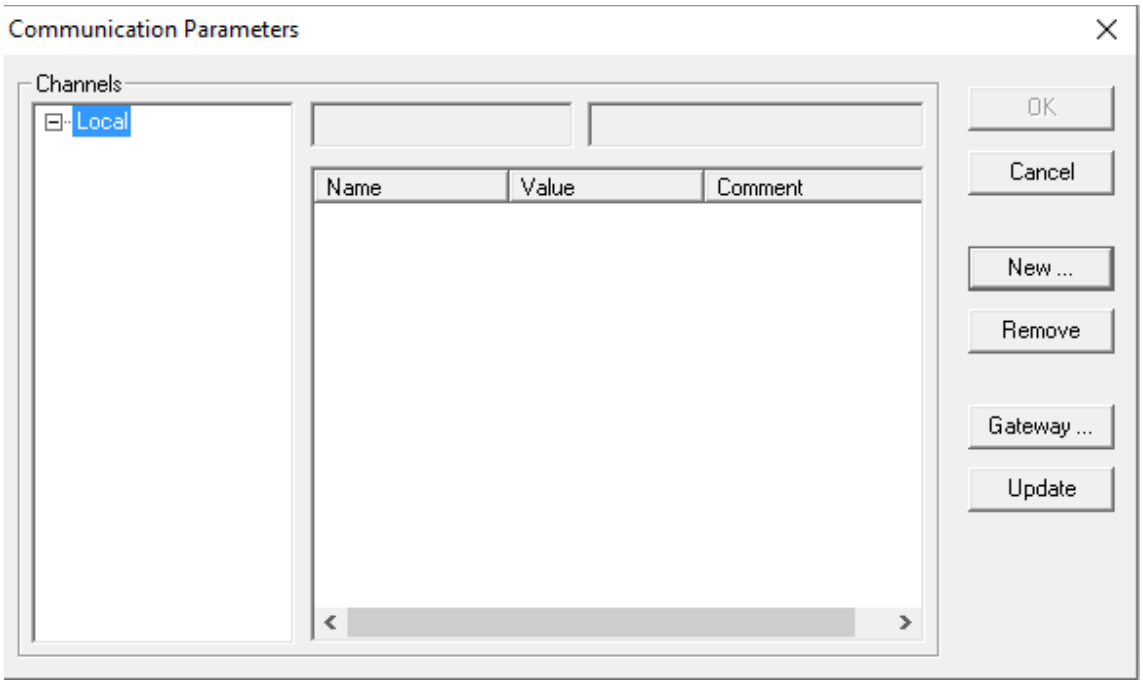


Figura 4.6.5 Paráetros de comunicación

Para añadir una nueva conexión deberemos darle a *New* y añadir nuestra conexión

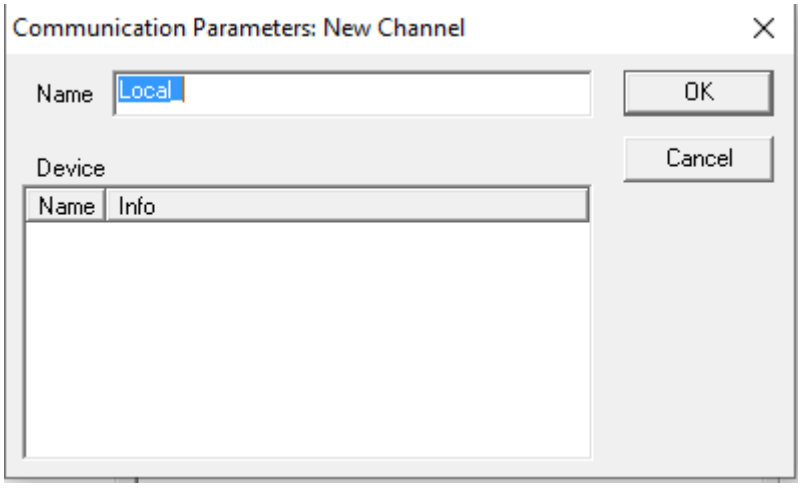


Figura 4.6.6 Añadir nueva conexión

También deberemos hacer click en *Gateway* e introducir parámetros como IP y puerto para cambiar los parámetros de nuestra conexión.

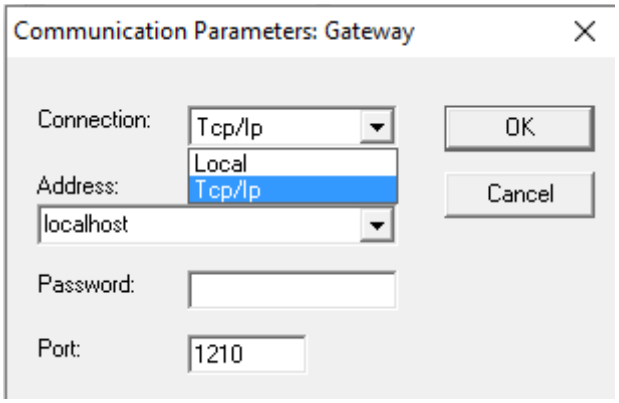


Figura 4.6.7 Añadir conexión mediante Gateway

4.6.2 VARIABLES Y SÍMBOLOS EN CODESYS

Para poner en común las variables de CoDeSys con el SCADA, lo primero que deberemos hacer es declarar las variables de forma global en la pestaña *Resources* y seleccionar *Global_Variables* dentro de la carpeta *Global Variables.*

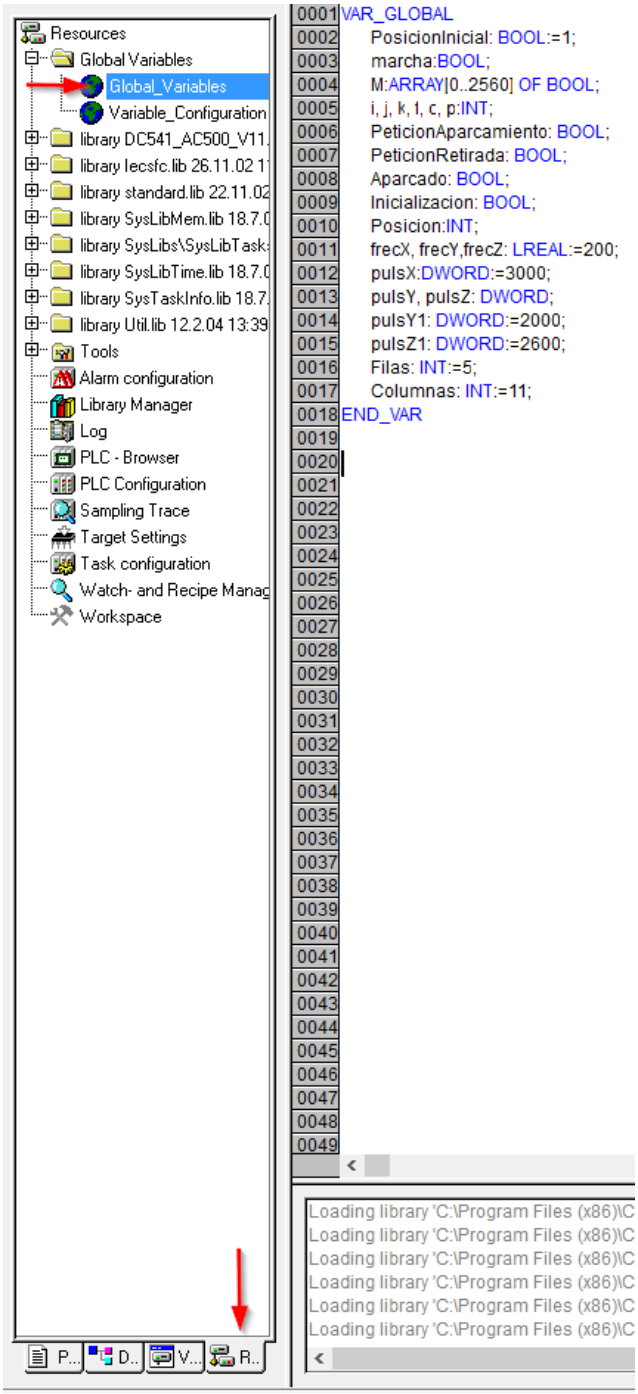


Figura 4.6.8 Variables globales

Para realizar la configuración de los símbolos en CoDeSys deberemos ir a la pestaña superior *Project* y seleccionar la opción *Options*

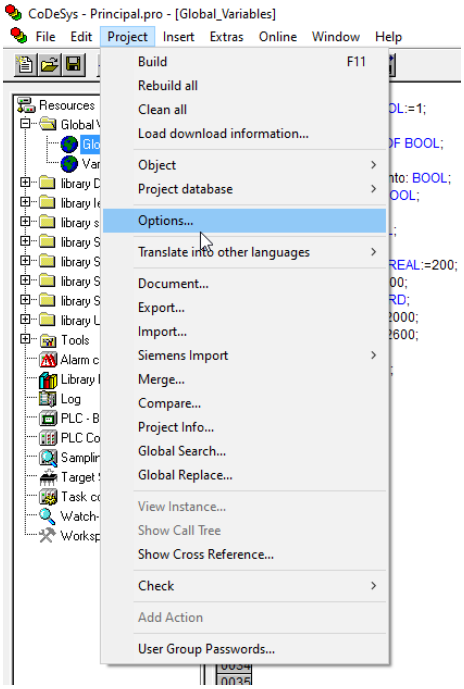


Figura 4.6.9 Opciones del proyecto

Una vez en *Options* deberemos ir a *Symbol Configuration* y hacer click en *Configure symbol file*

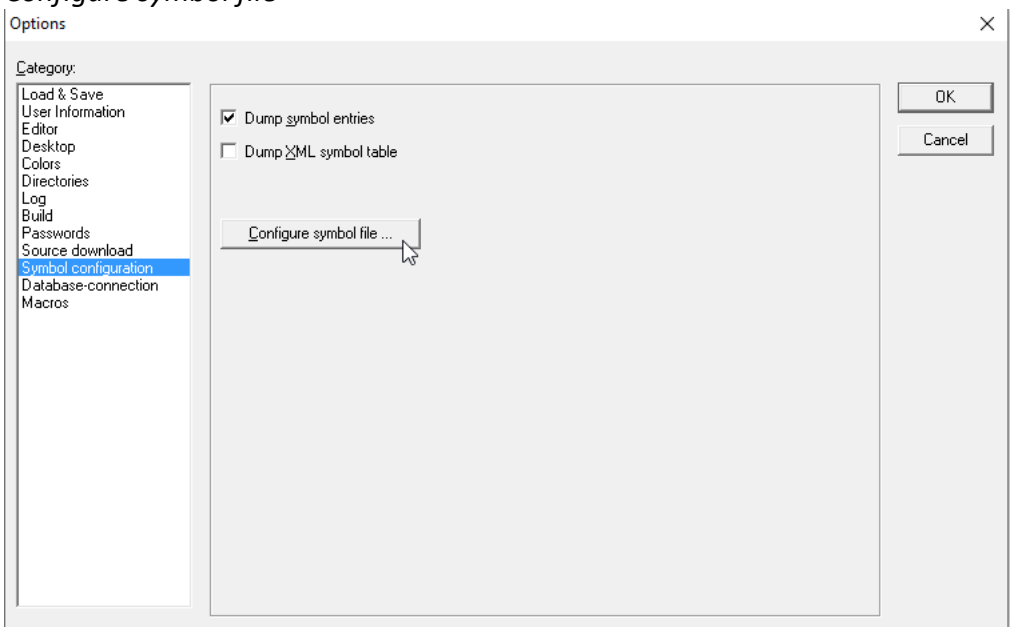


Figura 4.6.10 Configuración de símbolos

Ahora tendremos que seleccionar el grupo de variables que queremos tener en nuestro SCADA, que equivale al grupo *Global_Variables*

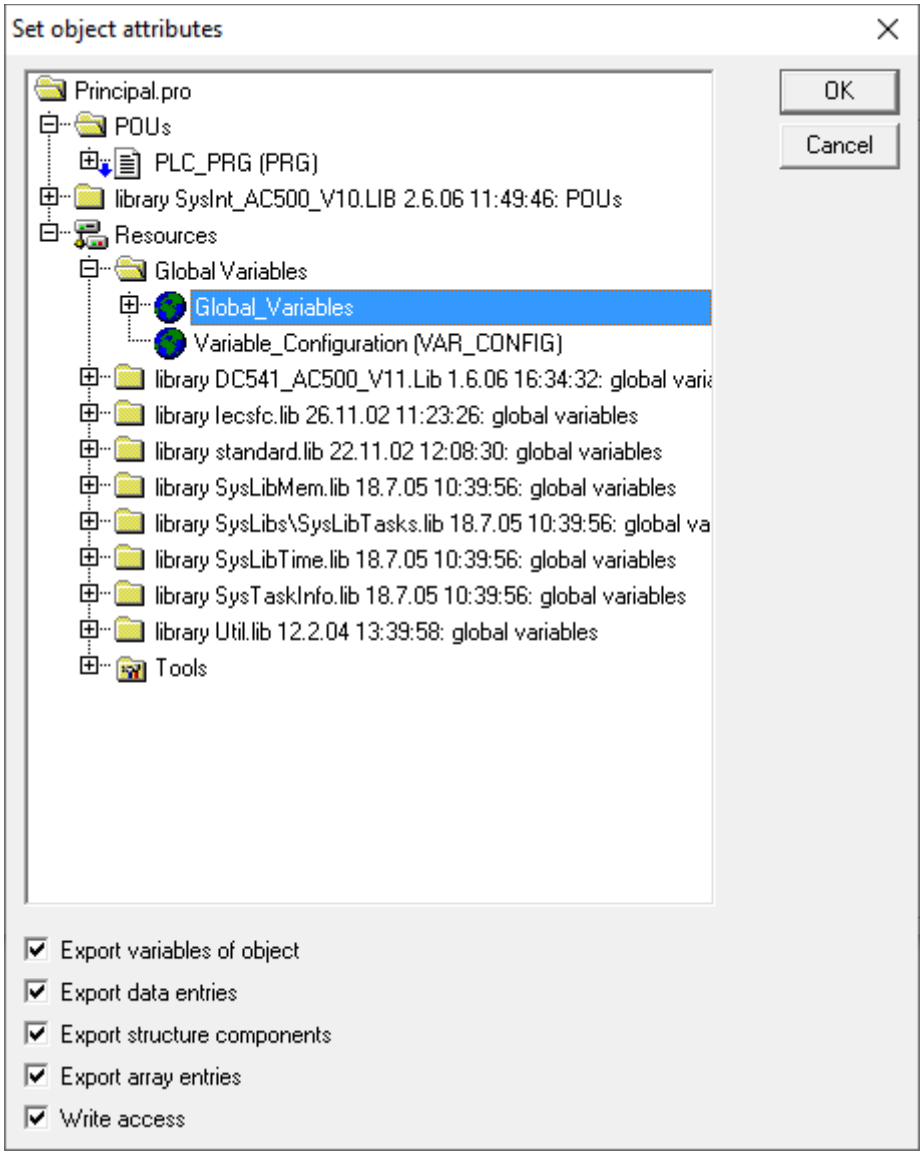


Figura 4.6.11 Exportar símbolos

Por último, para rehacer los archivos del programa y se apliquen correctamente los cambios deberemos irnos a la pestaña superior *Project* y hacer click en *Rebuild all*

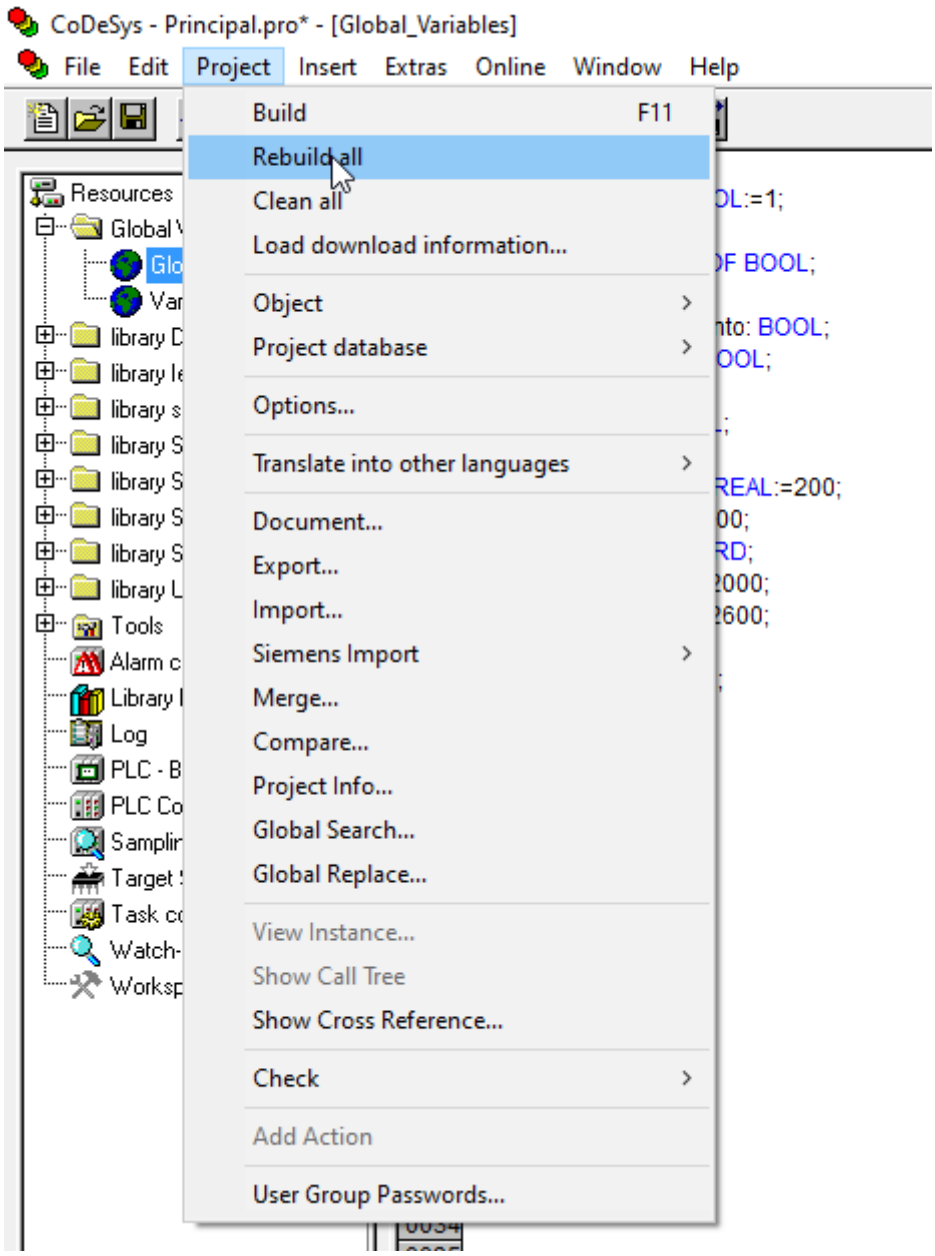


Figura 4.6.12 Reconstruir archivos del proyecto

4.6.3 PUNTOS Y ATRIBUTOS DE CONTROL DE COMUNICACIONES EN CX – SUPERVISOR

Para añadir los puntos en CX-Supervisor que vamos a tener sincronizados con el programa de CoDeSys deberemos seleccionar a la hora de agregar un punto como *Tipo de E/S – Entrada, Salida* o *Entrada/Salida* y en *Atributos de E/S* seleccionaremos *OPC/Otros*.

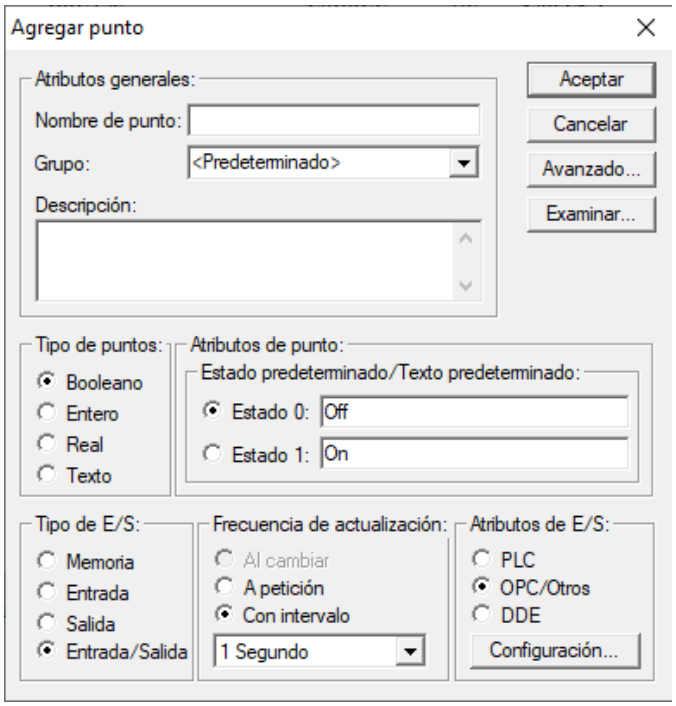


Figura 4.6.13 Agregar punto

Tras esto hacemos click en *Configuración*

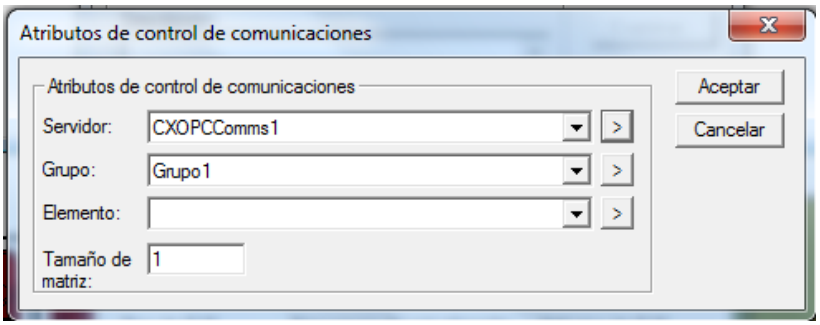


Figura 4.6.14 Configurar comunicaciones

En *Servidor* le damos a la flecha de la derecha y seleccionamos *Agregar* y seleccionamos *OMRON CX OPC Communications Control*

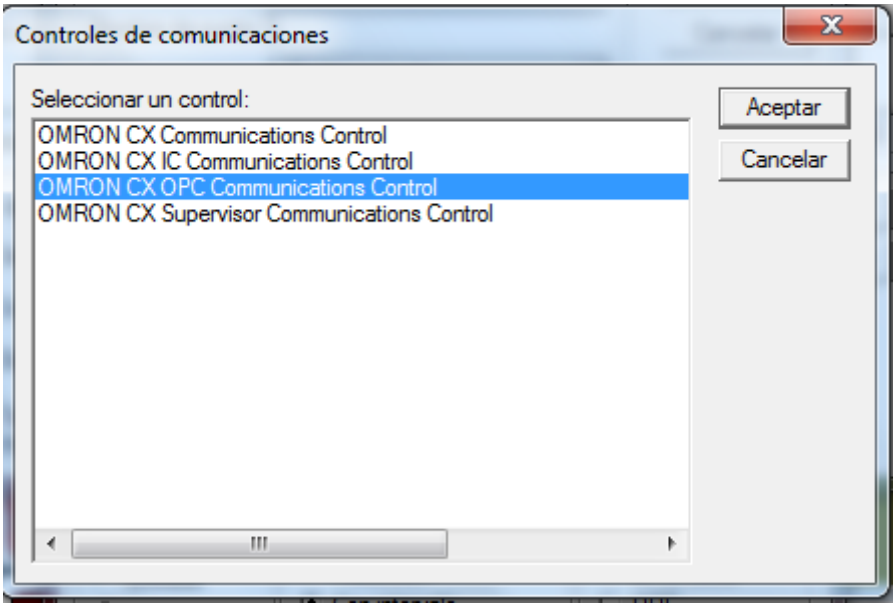


Figura 4.6.15 Controles de comunicación

Tras esto, seleccionaremos el *Nombre del ordenador, Nombre del servidor* y crearemos un nuevo archivo de proyecto o abriremos uno existente.

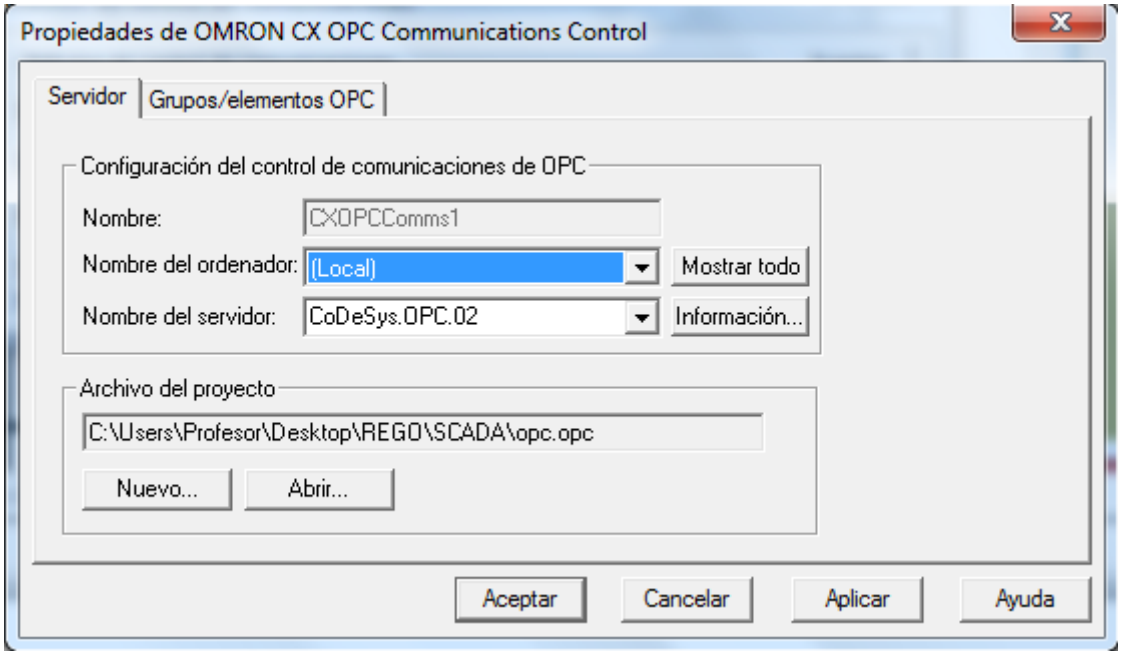


Figura 4.6.16 Propiedades de OPC

En la misma ventana, en la pestaña *Grupos/elementos OPC* podremos añadir grupos haciendo click derecho en el apartado *Groups* y ya dentro de un grupo podremos añadir elementos haciendo click derecho en el recuadro blanco a la derecha y seleccionando *Añadir elemento(s)*

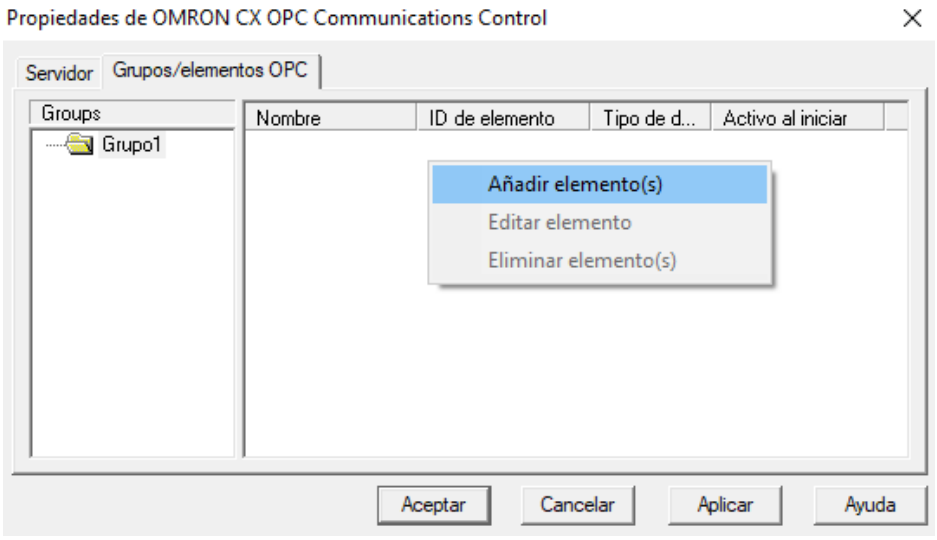


Figura 4.6.17 Añadir elementos OPC

Una vez estemos en la opción de añadir elementos tendremos una ventana donde nos saldrán todas las variables que en CoDeSys hemos puesto en *Global_Variables.* Seleccionaremos las que vayamos a usar y le damos al botón *Añadir elementos.*

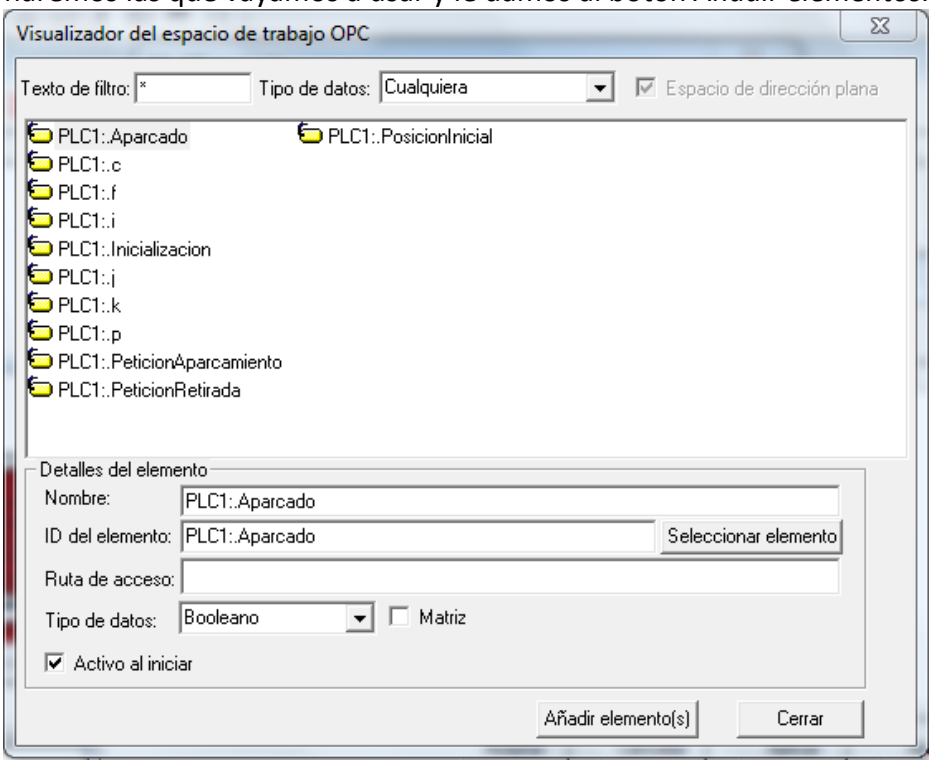


Figura 4.6.18 Visualizador del espacio de trabajo OPC

Una vez hayamos añadido los elementos que vamos a usar podremos seleccionar qué elemento corresponde a nuestro punto haciendo click sobre él y luego dándole al botón de *Aceptar*

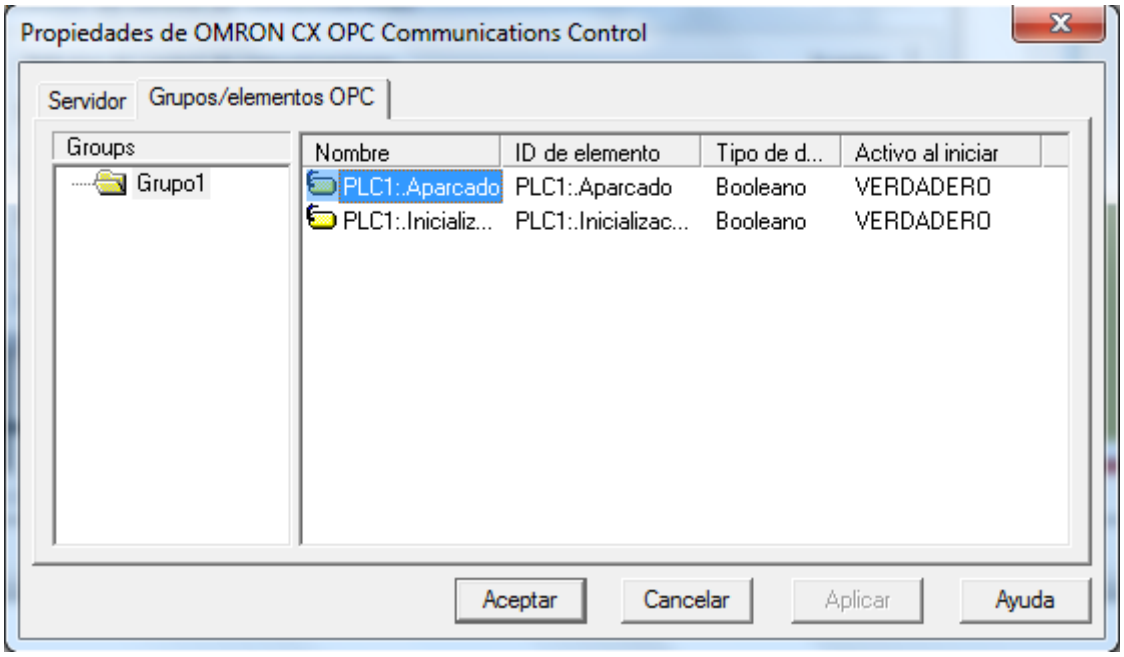


Figura 4.6.19 Elementos añadidos

Realizando los pasos anteriores llegamos otra vez a la ventana de la figura 4.6.20 donde podremos comprobar que está correctamente el *Servidor*, *Grupo* y *Elemento* además de variar el tamaño de la matriz según lo requiera nuestro punto.

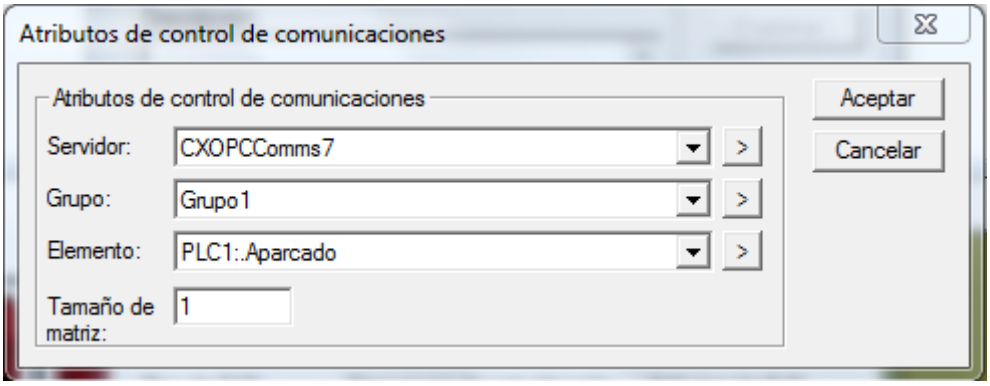


Figura 4.6.20 Atributos de control de comunicaciones