Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Nombre del Centro

Trabajo Fin de Grado

ESTUDIO DE CODESYS, ENTORNO DE EDICIÓN DE PROGRAMAS PARA PLCS Y SIMULACIÓN DE ESTACIONES AUTOMATIZADAS

Alumno: Antonio Garcia Rodriguez

Tutor: Prof. D. Ángel Gaspar González Rodríguez
Dpto: Departamento de Ingeniería Electrónica y Automática

Mayo, 2019

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Departamento de Informática

Don ÁNGEL G. GONZÁLEZ RODRÍGUEZ, tutor del Proyecto Fin de Carrera titulado: ESTUDIO DE CODESYS, ENTORNO DE EDICIÓN DE PROGRAMAS PARA PLCs Y SIMULACIÓN DE ESTACIONES AUTOMATIZADAS, que presenta ANTONIO GARCÍA RODRÍGUEZ, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, MAYO de 2019

El alumno: El tutor:

ANTONIO GARCÍA RODRÍGUEZ ANGEL G. GONZÁLEZ RODRÍGUEZ

Índice

Índice de Ilustraciones

Índice de Tablas

0. Objetivos y Motivación del TFG

El objetivo de este trabajo será la implementación y simulación de los distintos lenguajes existentes en la automatización de sistemas. Para poder llevarlo a cabo, se utilizará CODESYS, siendo éste un entorno de desarrollo para la programación de controladores. Se presentarán diversos ejercicios didácticos para poder aprender paso a paso el uso de esta herramienta.

A nivel motivacional, la elección de este trabajo final de grado es debido a que se trata de un software de reciente creación, el cual se está expandiendo, y cada vez más fabricantes están apostando por él. Quedan reflejados los conceptos usados/aprendidos en las asignaturas cómo Automática industrial, Informática industrial o Ingeniería de Control.

Utilizando los distintos lenguajes de programación y una serie de funciones avanzadas incorporadas por el software se han desarrollado diferentes ejercicios. Gracias a la opción de simulación virtual, ha permitido simular desde la consola actuaciones externas sin necesidad de tener conectado un PLC.

Por otro lado, se tratará de mostrar el potencial que tiene el programa gracias a la diversidad de opciones que muestra, como es la creación de una aplicación visual dónde poder ver en tiempo real como está funcionando los distintos programas.

1. Introducción al Programa de Edición y Simulación CODESYS

CODESYS (Sistema de Desarrollo Controlado) es una plataforma de software utilizada en la tecnología de automatización industrial. La plataforma se basa en el sistema de desarrollo CODESYS, una herramienta de programación IEC 61131-3. Proporciona a los usuarios finales soluciones integradas para la ingeniería de proyectos en aplicaciones de automatización. Esta herramienta cubre la ingeniería de proyectos, la programación, el funcionamiento en estaciones de trabajo, así como la ejecución y la depuración del código de la aplicación en el controlador. En la Ilustración 1. Se muestra la introducción al programa CODESYS.

(3S-Smart Software Solutions GmbH CODESYS, 2018).

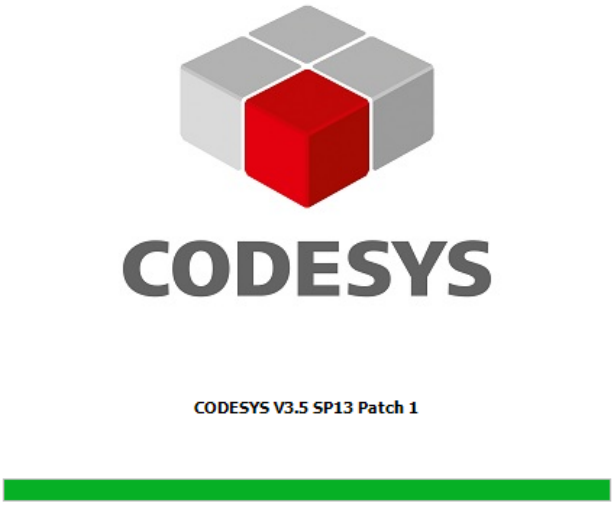

Ilustración 1. Logo e información del programa CODESYS

Además, los fabricantes de dispositivos usan CODESYS para crear sus propios componentes de automatización configurables y programables. Los dispositivos integrados para los diferentes niveles de un sistema de automatización son la base del éxito de CODESYS. Existen millones de dispositivos individuales compatibles con CODESYS, más de 1.000 tipos de dispositivos diferentes de más de 400 fabricantes y decenas de miles de usuarios finales de CODESYS en todo el mundo. Estos prueban que CODESYS es la herramienta de programación IEC 61131-3 independiente del fabricante líder en el mercado, algo que también permite

CODESYS es que se puedan usar al momento sin costo adicional los buses de campo más comunes.

Los usuarios finales emplean CODESYS para la satisfactoria creación de aplicaciones simples y/o complejas para controladores industriales. Que se utilizan en sectores como la fabricación, maquinaria móvil, edificación y energía, así como en otros muchos sectores industriales.

Las principales ventajas son:

- Sistema de programación IEC 61131-3 completo – desde la programación clásica de PLC, hasta la programación de controladores orientados a objetos.
- Extensas funciones para la ingeniería de proyectos adecuada y la puesta en servicio de aplicaciones de automatización, esto incluye monitorización de datos, búsqueda de errores de aplicación y cambio de la aplicación durante el funcionamiento.
- Módulos adicionales opcionales para el desarrollo metódico de aplicaciones.
- Configuración y puesta en marcha de los buses de campo industriales más importantes. Así como de sistemas de E/S específicos del fabricante, como CANopen, EtherCAT, PROFIBUS, Modbus o PROFINET.
- Integración perfecta de componentes adicionales opcionales para la ingeniería de proyectos: Visualización, control de movimiento, CNC, robótica o módulos de seguridad.
- Editores fáciles de usar como FBD o LD.

(Larraioz, 2018)

1.1. Cómo conseguir el programa

CODESYS es una herramienta abierta de software para simular, visualizar y programar equipos. Se puede encontrar a través de su página web en la sección de *store.* La versión utilizada de CODESYS ha sido la *V3.5 SP13 Patch 1.*

(CODESYS, Store CODESYS, 2018)

1.2. Por qué utilizar CODESYS

- Más de 500 PLC´s se programan en CODESYS, Bosch Rexroth, FESTO, Beckhoff, Scneider Electric, IFM o Mitshubishi son algunas de las marcas de fabricantes que ya han apostado por CODESYS, todas ellas se pueden encontrar en la lista de fabricantes en la web de CODESYS.

  (CODESYS, Lista fabricantes CODESYS, 2018)

- La versatilidad a la hora de programar tu PLC con 5 lenguajes diferentes. Simulador y HMI integrado. Trae consigo un simulador integrado, cosa que facilita muchísimo la vida al programador.

- El entorno de programación es totalmente gratuito, lo que permite aprender a programar sin ningún coste, a excepción de los drivers que debemos subir a cada PLC donde si tiene un coste de licencia.
- Librerías. El entorno de CODESYS está ligado a *CODESYS STORE*, que tiene múltiples librerías para ampliar funcionalidades de las soluciones de automatización, como veremos en el caso de la *Raspberry Pi.*

(Opiron Satoshi, 2017)

1.3. Tipos de lenguajes existentes

CODESYS está compuesto de hasta 5 lenguajes diferentes con los que programar:

1. ST: Texto estructurado (*Structured text).* Es un lenguaje de alto nivel similar a C, lo cual necesita un conocimiento previo de programación.
2. LD: Diagrama de contactos (*Ladder diagram)*. Son conexiones gráficas entre variables de tipo *booleana*, dónde representamos los diagramas de conexiones de circuitos eléctricos, nos brinda la posibilidad de utilizar contactos y bobinas, este lenguaje tiene una estructura simple.
3. FBD: Bloques funcionales (*Function Block Diagram).* Es un lenguaje de alto nivel que permite resumir funciones o programas de una manera visual para el usuario, ya que sólo se preocuparía de la programación funcional de su rutina.
4. SFC: Diagrama de funciones secuencial (*Sequential function chart).* Es un método gráfico de modelado y descripción de automatismos secuénciales, en el cual los estados por los que va pasando el sistema depende de algún cambio en algunas de las entradas, cuyos elementos básicos son las etapas, las transiciones y las acciones, cada etapa representa un estado del sistema. Las transiciones están asociadas a una condición de verdadero/falso, los cuales desactivan la etapa anterior y habilitan la etapa siguiente.
5. Lista de instrucciones *IL-Instruction List.* Consta de una secuencia de instrucciones, cada instrucción se inicia en una línea nueva y contiene un operador y, según el tipo de operación, uno o varios operandos por comas.

Delante de una instrucción puede encontrarse un identificador llamado *marca,* seguido de dos puntos. Sirve para identificar la instrucción y puede utilizarse por ejemplo como destino de salto.

El último elemento en una línea debe ser un comentario. Pueden insertarse líneas vacías entre instrucciones.

(DBR Automation, 2015)

1.4. Estructura de un proyecto

CODESYS está basado en el protocolo IEC 61131-3, el cual la forma que tiene de organizar las funciones y bloques son las unidades de Organización de Programa (*program organization units o POUs*). Éstas permiten crear programas, funciones y funciones con memoria. Por lo tanto, un bloque de programa puede ser de tipo:

Existen los siguientes tipos de POUs:

- Programa PLC*_PROG*. Como el programa principal. En los PLCs multitareas, pueden ejecutarse simultáneamente un elevado número de programas principales.
- Función *FUN*. Es aquella que puede tener parámetros fijados también llamados argumentos, pero no tiene variables estáticas. Es decir, no tiene memoria de modo que para los mismos valores de entrada se obtienen siempre los mismos valores de salida. Un ejemplo son las funciones matemáticas: *MUL, GT, etc.*
- Bloque de funciones *FB*, es el que tiene variables estáticas. Sus salidas dependerán de las condiciones de sus variables, tanto internamente como externamente, sus valores permanecen constantes entre las ejecuciones individuales del bloque de funciones. Se trata también de bloques principales para generar un programa de PLC.

  (IEEC UNED, 2017)

1.5. Edición

Para la edición de un proyecto los pasos a seguir serán la creación de un nuevo proyecto, luego se procederá a la creación de un entorno gráfico.

1.5.1. Creación y configuración de un nuevo proyecto

Para iniciar a trabajar se creará un nuevo proyecto el cual el primer paso es abrir CODESYS pulsar en la pestaña en el menú *Archivo* > *Nuevo Proyecto* como se muestra en la Ilustración 2:

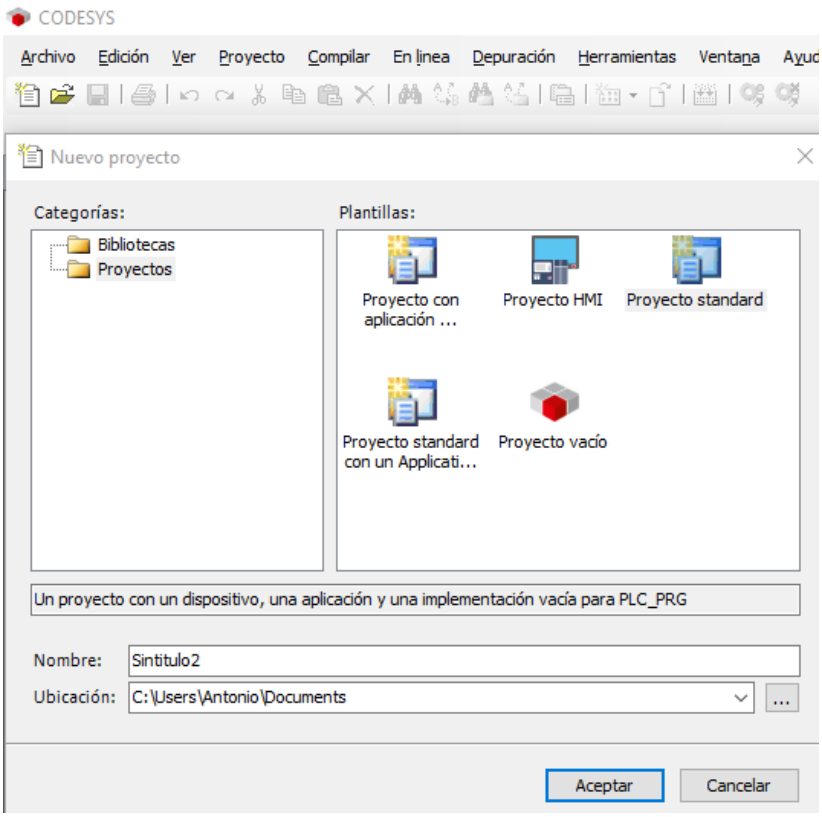

Ilustración 2. Primera ventana que aparece al pulsar la pestaña de nuevo para la creación de un proyecto

El programa ofrece distintas opciones: proyecto vacío, proyecto estándar, proyecto HMI, proyecto estándar con compositor de aplicaciones o proyecto estándar con una aplicación. Se recomienda la opción de proyecto estándar.

(Anton Girod, 2017)

Aparecerá una ventana para seleccionar el tipo de lenguaje en el que vamos trabajar, se elige el deseado y le damos a aceptar, tal y como se ve en la Ilustración 3.

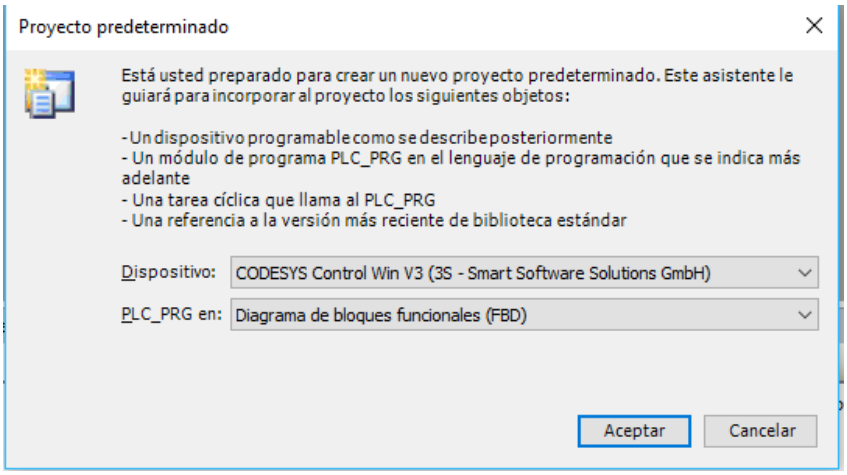

Ilustración 3. Ventana emergente para la configuración de proyecto

Cargará todo el proyecto y ya se tiene configurado. Una vez listo deberá aparecer la lista a la izquierda en la pestaña de dispositivos.

1.5.1. Entorno Grafico del software

Se encontrará en el programa una pantalla de exploración o ventana principal con diferentes menús de trabajo, en la que se ven los siguientes elementos de la Ilustración 4:

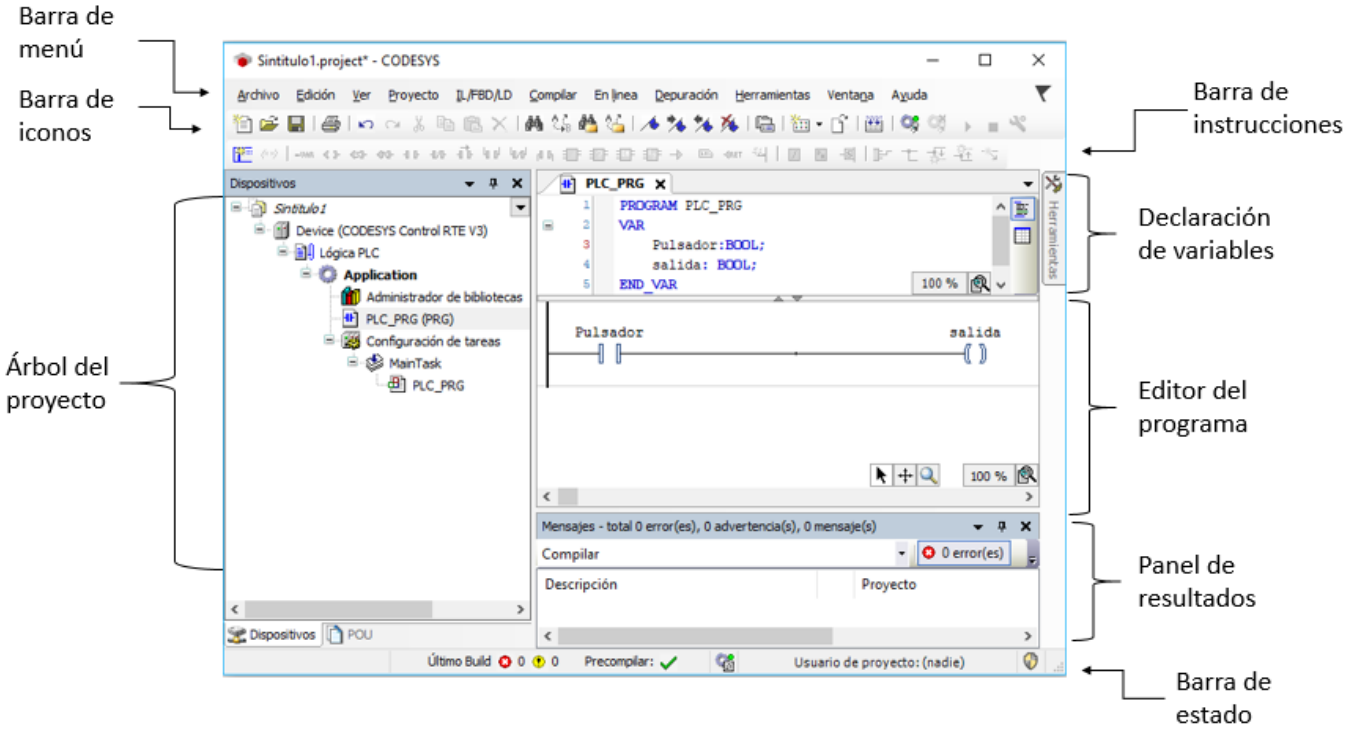

Ilustración 4. Ventana principal del programa CODESYS que se encuentra al iniciar el programa

- **Barra de menú:** Permite realizar las tareas fundamentales de cualquier programa.
- **Barra de iconos:** Muestra los accesos directos a las tareas más utilizadas.
- **Árbol de proyecto:** En esta ventana se hace un pequeño resumen del esquema de cómo está estructurado el programa, en la pestaña dispositivos se muestra el PLC utilizado, también se puede definir una visualización o ver todos los bloques utilizados, en la pestaña **POU**, se utiliza para la visualización y creación de nuevas **POUs,** como Colección de imágenes o nuevas **POUs**.
- **Barra de instrucciones:** Permite rápidamente una lista de instrucciones fundamentales utilizadas más a menudo. A la izquierda se tiene una pestaña de ***Herramientas***, donde encontrar el resto de instrucciones o módulos.

- **Declaración de variables:** Ventana en la cual se hace la declaración local de las variables utilizadas en la POU.
- **Editor del programa:** Es la ventana principal donde se insertará nuestro código de programa. Y la llamada a todos los bloques utilizados.
- **Panel de resultados:** Una vez compilamos el programa, muestra si ha habido algún tipo de error o advertencia de programa. Si todo está correcto indicara *Compilación terminada.*
- **Barra de estado:** Indica el estado actual del programa. Si está en modo ONLINE/OFFLINE, si está en ejecución o en modo *SIMULACIÓN.*

(Universidad de León, 2017)

1.5.3. Agregar una librería o algún paquete

Muchas veces algunos de los módulos que se han utilizado no vienen en las librerías por defecto y se tiene que añadir la librería en donde están. Para ello se hace doble clic sobre *Administrador de bibliotecas > agregar nueva biblioteca, tal y como se ve en la* Ilustración 5*:*

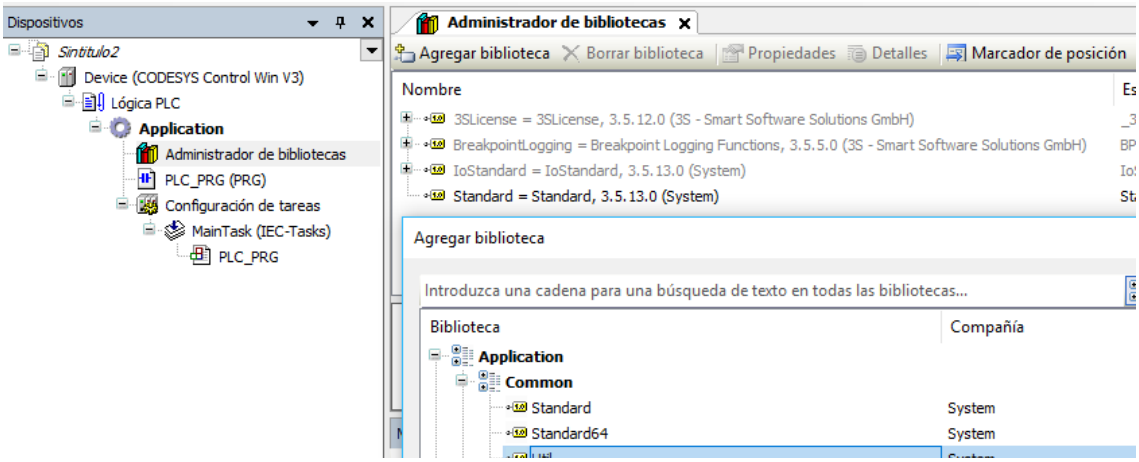

Ilustración 5. Pasos a seguir de como agregar una librería a un programa

Para añadir algún paquete, como en este caso, el de *Control for Raspberry PI,* se ha tenido que descargar primero. Bajando de la propia página oficial del programa CODESYS donde encontramos un *STORE,* en el cual te puedes registrar gratuitamente para descargar tanto el programa que es gratuito como el paquete de

Raspberry. Una vez descargado, para instalarlo en la barra de arriba dentro de *Herramientas > Administrador de paquetes,* clicamos sobre el botón **Instalando,** se busca el paquete en la carpeta descargada. En la siguiente Ilustración 6 se puede apreciar:

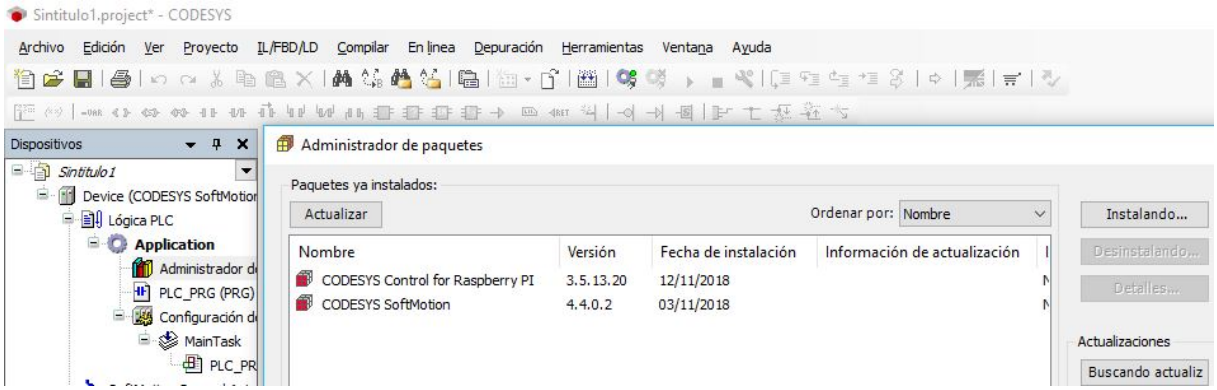

Ilustración 6. Ventana que muestra donde se agregan los paquetes, desinstalan o actualizan

1.5.4. Agregar visualización

En todos los proyectos se han utilizado visualizaciones, para así poder mostrar más fácilmente y tener un alcance mayor a todo el mundo con los ejercicios resueltos. De manera que una persona que no entienda muy claramente el código programado pueda ver de una manera gráfica lo que se ha tratado de mostrar.

Para añadir una visualización al proyecto se tiene que seleccionar con el botón derecho sobre Application > Agregar Objeto > Visualización, a continuación la Ilustración 7 muestra la ruta a seguir:

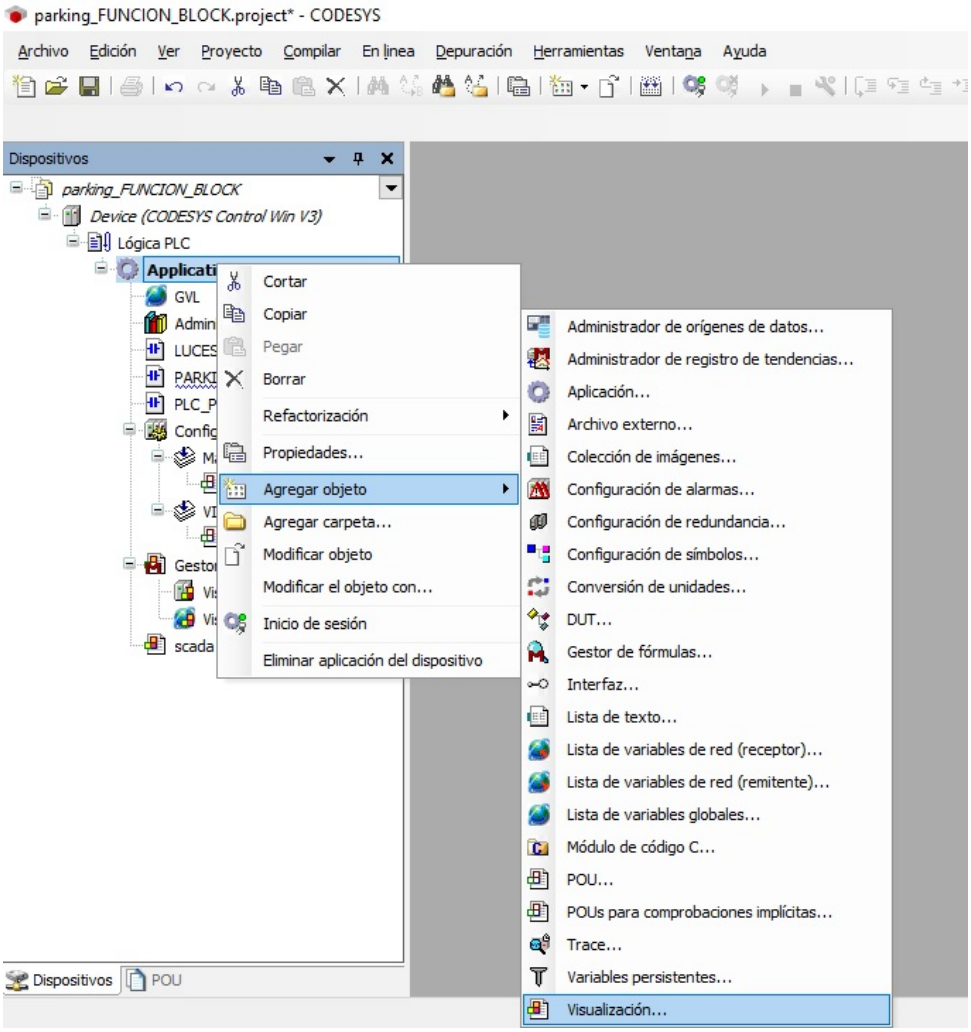

Ilustración 7. Ruta de como agregar una visualización

1.5.5.Insertar Variables Globales

El programa CODESYS te permite insertar variables globales para trabajar con ellas en distintos bloques. Así se puede mantener o modificar el valor de esa variable en distintos bloques. Para ello se selecciona *Application* > *Agregar Objeto* > *Lista de variables globales,* tal y como se ve en la Ilustración 8 *:*

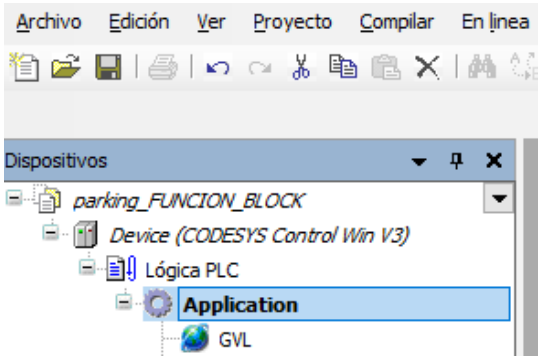

Ilustración 8. Ubicación dentro del esquema de programa de las Variables Globales

1.5.6. Insertar POU (Programas o Bloques)

Una POU es una Unidad de organización del programa en un proyecto CODESYS. Cada POU consiste en una parte de declaración y un cuerpo. El cuerpo del programa se escribe en el editor del programa en uno de los lenguajes anteriormente descritos.

Si se quiere trabajar con distintos lenguajes de programación, como en este caso, se ha utilizado más de un bloque *FBD* (**Bloques de Funciones**) y *LD* (**Diagramas de contactos**) *Application* > *Agregar objeto* > *POU,* a continuación la Ilustración 9 muestra la ventana que aparece:

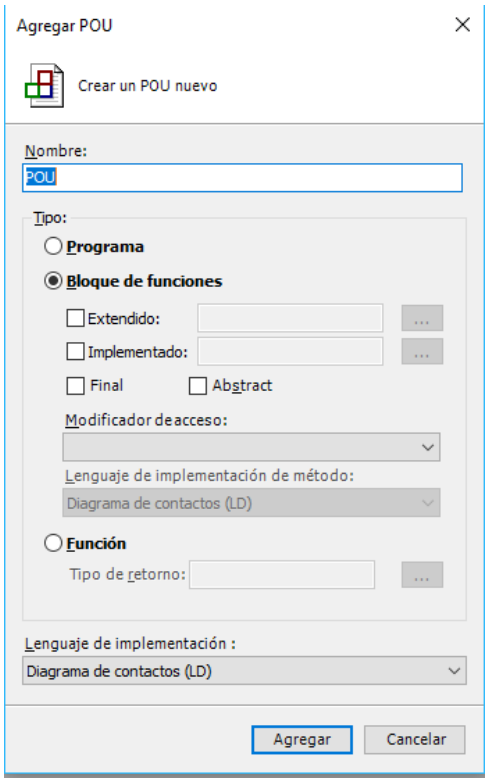

Ilustración 9. Ventana emergente para Agregar una POU

1.5.7. Declarar Variables de programa

Las variables de programa son aquellas que se declaran dentro de un programa o bloque, de manera que sólo existen dentro de ese bloque. Es decir, una vez que salgamos dentro de ese programa o bloque dejará de existir. Nótese la diferencia entre las variables de programa y las variables Globales donde éstas últimas siguen existiendo en todos los bloques o programas. Parra insertar/declarar variables dentro de un programa hay dos formas. Una conforme se está creando el programa, se despliega una ventana emergente conforme se termina de escribir la variable, y se escoge dentro del campo visibilidad el tipo de variable (si es de salida, entrada, global, etc.), el tipo de dato que es (Booleana, Entero, Real, Palabra, Doble Palabra), el valor con el que se quiere iniciar dicha variable, la dirección y algún comentario aclaratorio o referencia, como se aprecia en la Ilustración 10:

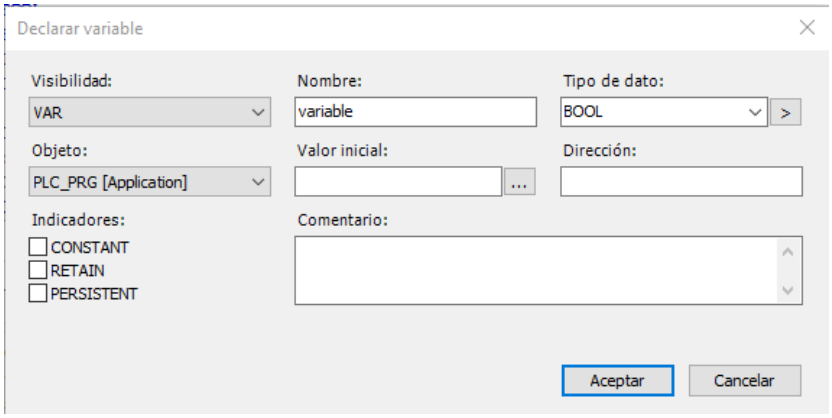

Ilustración 10. Ventana emergente que aparece al escribir una variable y no estar declarada

Y otra forma es directamente sobre el propio programa. Se encuentra una ventana reservada para poder escribir y declarar todas las variables si se tienen ya predefinidas. Por ejemplo, aquí a la variable **SENSOR1** se le ha asignado que es de tipo **BOOL** y se inicializa con valor **0,** a continuación se puede ver en la Ilustración 11:

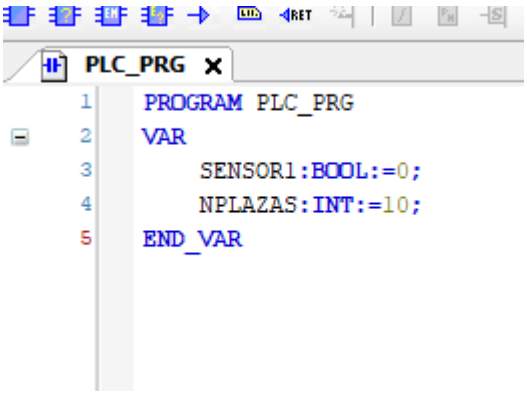

Ilustración 11. Ejemplo de declaración de variables de forma manual, se inicializan las variables con valores

Algunos ejemplos para declarar variables serían:

- **SENSOR: REAL:=-10.5;** se le indica que es tipo real y se le asigna inicialmente el valor -10.5.
- **NPLAZAS: INT:=10;** indica que es de tipo entero y de inicio tendrá el valor 10.
- **control: PARKING;** también se puede asignar a una variable el nombre de un bloque, en este caso a la variable **control** se le asigna el bloque **PARKING.** El cual es un bloque FBD personalizado con sus entradas y salidas.

1.5.8. Insertar Objetos

Para la visualización se puede usar una serie de objetos como botones, lámparas, diagrama de barras, potenciómetros. Todos ellos ya vienen cargados en la biblioteca de símbolos por defecto:

Ilustración 12. Ventana de edición de la visualización de la paleta herramientas

Los objetos para poder interactuar con ellos, se deben cambiar las propiedades. Como el tamaño, color, movimiento e incluso poder utilizarlos como pulsadores, tal y como se puede ver en la Ilustración 12.

1.6. Simulación

Para poder simular antes se ha de compilar y verificar el código para comprobar que todas las variables están bien declaradas y asignadas. Los pasos a seguir serian primero compilar y luego simular.

1.6.1. Compilación.

Para poder compilar, en la barra superior se encuentra la pestaña ***Compilar*** > *Compilar,* una vez pulsado en caso de fallo se muestra una lista de errores o advertencias indicando en que línea o cual es el fallo establecido, y el programa no permite simular hasta corregir los errores existentes, a continuación un ejemplo en la Ilustración 13:

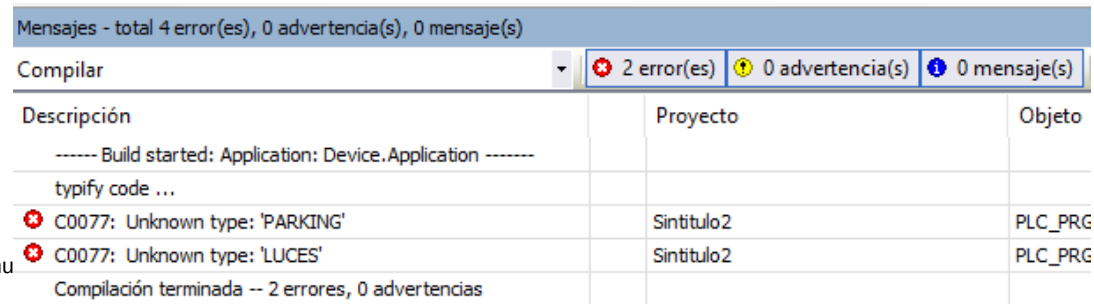

Ilustración 13. Mensajes de error de compilación a la hora de tratar de inicializar el programa

1.6.2. Simulación.

Se pueden hacer simulaciones en caso de no tener conectado ningún PLC al ordenador o sistema. El programa CODESYS permite hacer una simulación virtual a través de sus opciones. En la pestaña superior *En línea* > *Simulación* el cual simula tener conectado un PLC. Una vez dentro de la simulación para poder visualizar el programa y arrancar, se pulsa el botón ***Iniciar sesión***. Esto cargará la aplicación en memoria y una vez todo está todo correcto, permite arrancar la aplicación, ejecutando y cargando todo el programa en memoria. En la Ilustración 14 se muestra el proceso descrito:

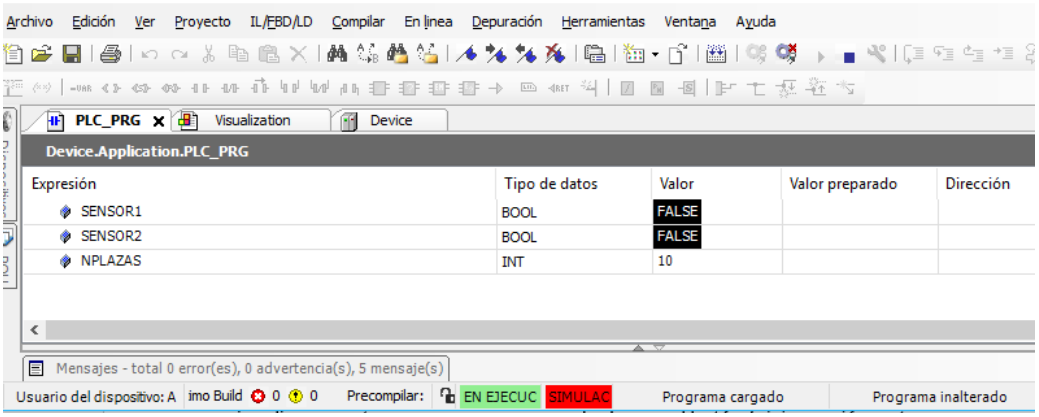

Ilustración 14. Ventana la cual muestra como se está ejecutando el programa

En el modo ***simulación*** se permite visualizar el valor de las variables en tiempo real e incluso poder asignarles un valor, forzando a éstas a entrar en ese valor. Para

Device.Application.PLC_PRG

Expresión	Tipo de datos	Valor	Valor preparado
SENSOR1	BOOL	TRUE	FALSE
SENSOR2	BOOL	FALSE	
NPLAZAS	INT	2	3

Ilustración 15. Variables en tiempo real

hacer esto último se debe de introducir el valor deseado dentro de la columna *Valor preparado* y ejecutar la secuencia de comandos ***Control + F7*** y automáticamente queda cargado el valor deseado. En la figura 15 se muestra un ejemplo de distintas variables donde dos de ellas son variables de tipo booleanas (BOOL) y otra de tipo entero (INT), donde *Valor* representa el valor actual y *Valor preparado* es el valor al cual se fuerza la variable. Es decir, la variable NPLAZAS tiene valor 2 y se forzará a valor 3, a continuación en la Ilustración 15, se muestra lo antes detallado:

Para arrancar la simulación visual se tiene que pulsar el botón ***Inicio (F5)*** en la barra superior, sino indicará en la ventana de simulación *"La visualización en línea se encuentra a la espera de una conexión. Reinicia la aplicación"*. Otra de las opciones que permite CODESYS es modificar el código y volver a cargar la aplicación.

1.7. Módulos Adicionales

CODESYS permite utilizar módulos adicionales, uno de los cuales es el CODESYS Control para Raspberry Pi. Para poder utilizar la Raspberry Pi lo primero que tenemos que hacer es insertar un sistema operativo que permita tener acceso a su escritorio remotamente. En este caso el sistema operativo utilizado es *Raspbian*, un sistema operativo de software libre.

Dentro de la Raspberry Pi lo siguiente que hay que hacer, es configurar el acceso remoto al escritorio concediendo permisos de Superusuario dentro del sistema operativo:

Lo que se debe hacer es instalar el servidor RDP. Para ello se conectará a través de *ssh* a la Raspberry Pi, abrir una aplicación llamada ***Terminal*** e introducir la siguiente línea:

```
sudo apt-get install xrdp
```

Cuando acabe ya estará lista para ser usada.

(Velasco, 2014)

1.7.1.CODESYS Control para Raspberry Pi

Una de las opciones que permite CODESYS es utilizar como PLC una Raspberry Pi. En nuestro caso se ha utilizado la *Raspberry Pi 3b,* como un extra o mejora de la propuesta inicial que se tenía sobre el TFG, para la simulación y comunicación con el programa del control de un motor PID. Al ser un hardware de fácil acceso universal y tiene amplia comunidad detrás.

Una de las motivaciones es debido es el bajo coste que tiene para la automatización de sistemas y además existe la creciente expansión que va teniendo este hardware. CODESYS ha apostado por esta herramienta, con un paquete para poder utilizarla como si fuera un PLC.

(INCIBE CERT, 2018)

1.7.2.Configuración programa.

Para la configuración del programa lo primero que se tiene que hacer es elegir el lenguaje de programación que se va a usar. En el dispositivo PLC se debe elegir *CODESYS Control for Raspberry Pi (3S – Smart Software Solutions GmbH),* como se aprecia en la Ilustración 16*:*

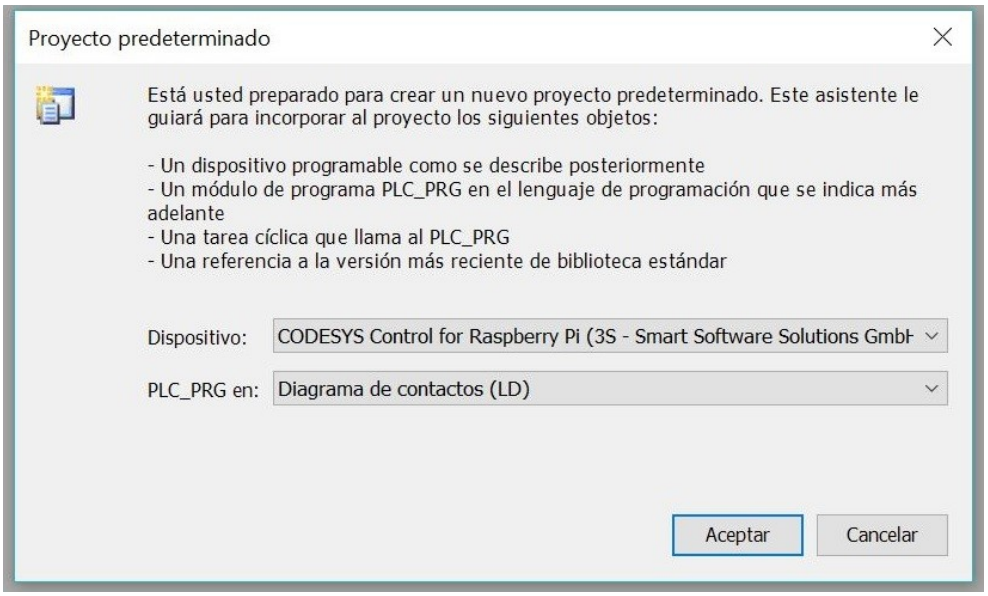

Ilustración 16. Ventana emergente del programa CODESYS para elección del lenguaje

El siguiente paso es configurar el puerto de conexión. Para ello se va a *Herramientas > Update Raspberry Pi.* Ahora existen dos opciones una se mira en el rúter la ip asignada, o se le indica al propio programa que busque la IP asignada a la Raspberry Pi a través del botón ***scan,*** tal y como se ve en la Ilustración 17:

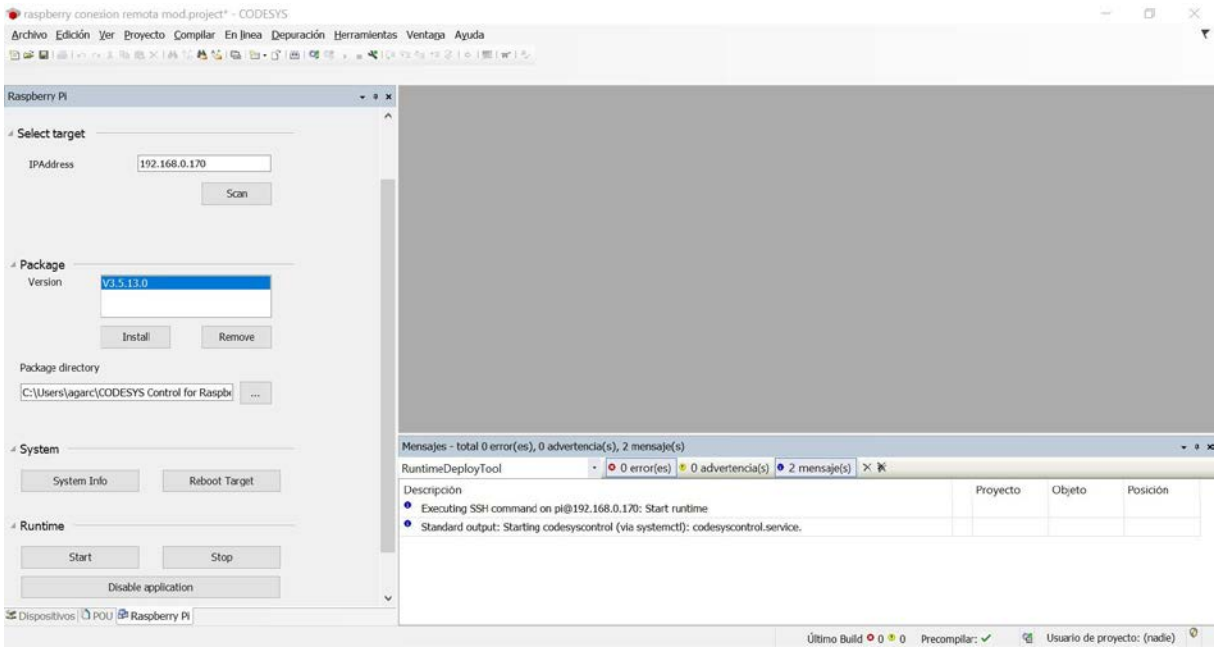

Ilustración 17. Ventana de configuración del puerto a utilizar por la Raspberry Pi

Se comprueba que la conexión se ha establecido entre la Raspberry Pi y el sistema, doble clic con el ratón en la pestaña del PLC *Device (CODESYS Control for Raspberry Pi),* en la Ilustración 18 se puede apreciar cómo está conectada:

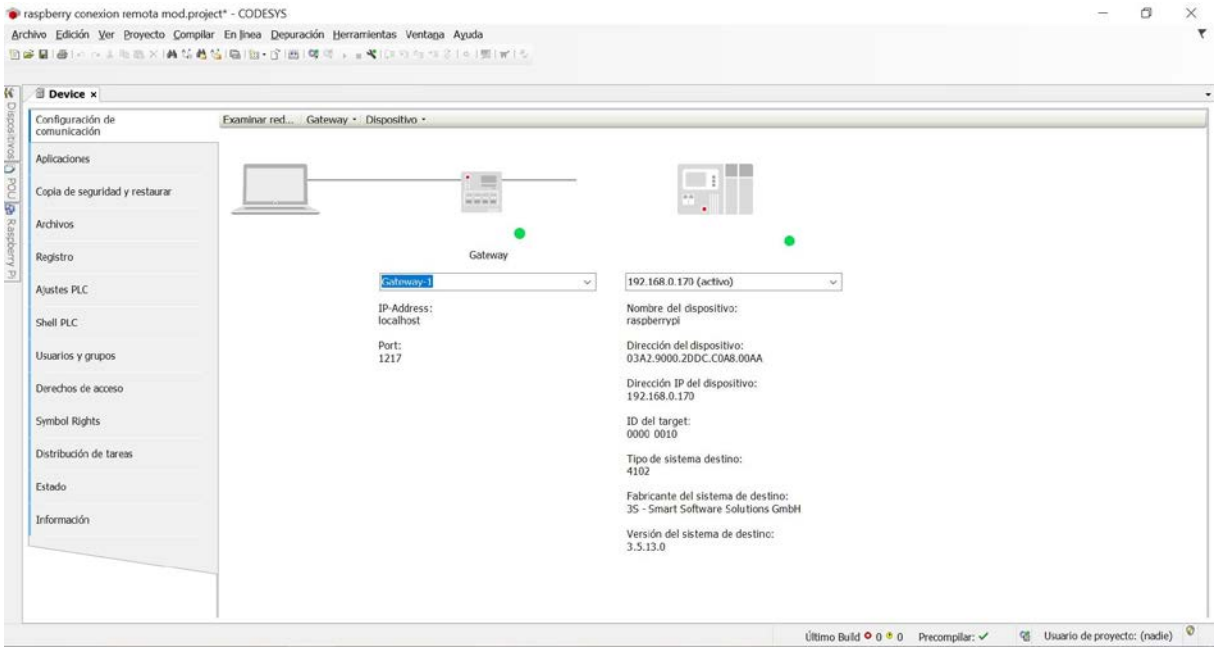

Ilustración 18. Ventana de conexión de la Raspberry Pi en el programa CODESYS

1.7.3. Configuración E/S fisicas Raspberry Pi.

Se puede asignar las entradas/salidas físicas de la Raspberry Pi a las variables del programa, para ello se tienen los GPIO (*General Purpose Input/Output*). Como su propio nombre indica son un sistema de entradas/salidas estos pines están incluidos en todos modelos Raspberry Pi, tal y como se aprecia en la Ilustración 21, se le tiene que especificar el pin que se quiere utilizar y el tipo de dato que será utilizado, en la Ilustración 19 se puede ver los puertos utilizados en un ejemplo de configuración, y en la Ilustración 20, las salidas/entradas utilizadas:

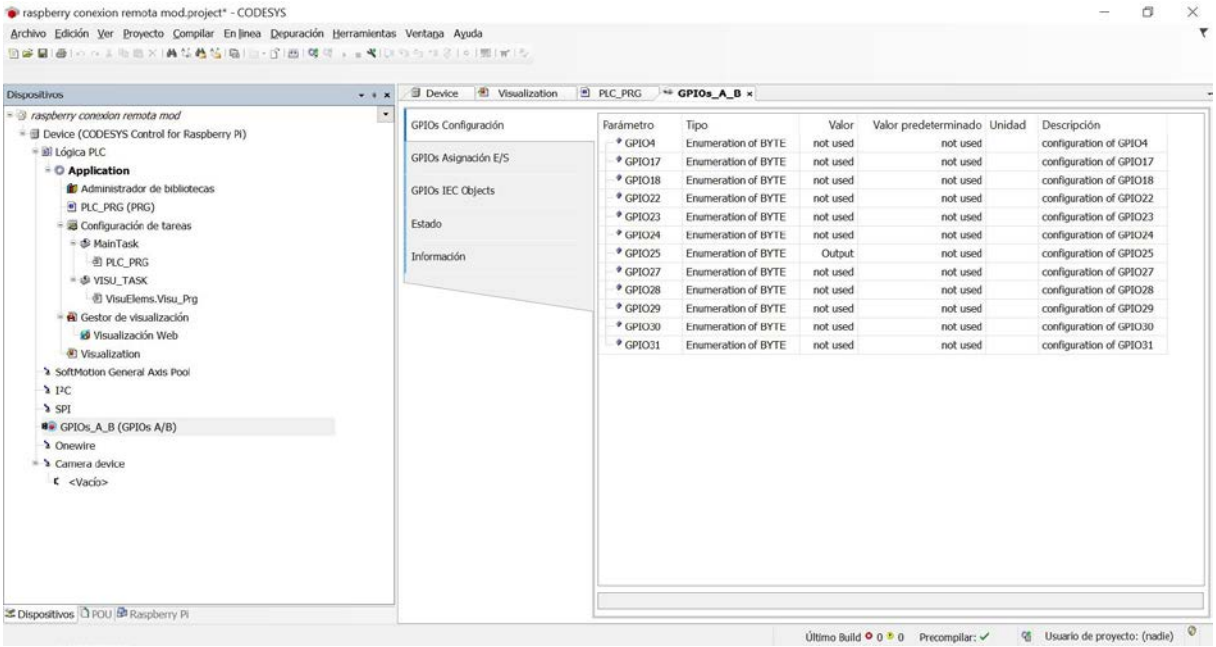

Ilustración 19. Configuración de las E/S de una Raspberry Pi

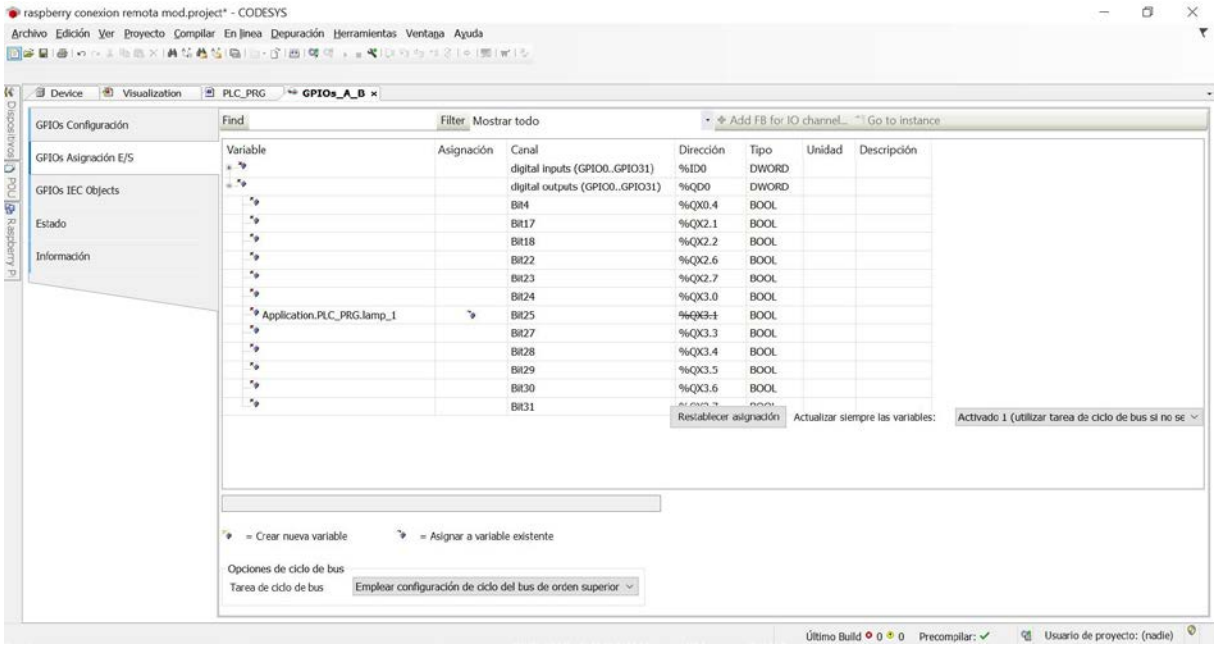

Ilustración 20. Asignación de las E/S de una Raspberry Pi

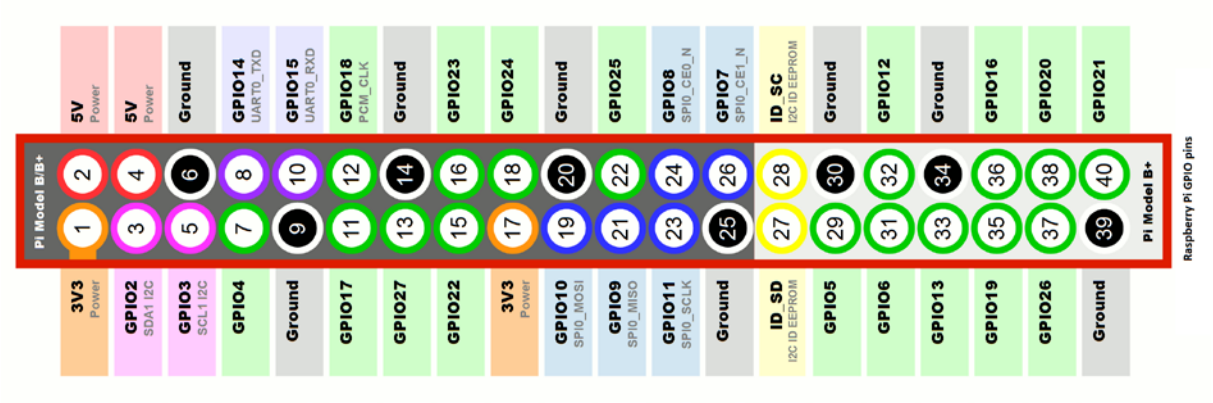

Ilustración 21. Mapa de conexiones GPIO de una Raspberry Pi

2. Análisis de soluciones

En este apartado se detallarán las soluciones consideradas para el desarrollo de este TFG.

2.1. Pilonas

En este ejercicio se ha desarrollado el control de unas pilonas hidráulicas. Se ha utilizado el lenguaje de programación ***SFC***, también conocido como *GRAFCET*, en el cual se tiene el control de dos pilonas hidráulicas, gracias a un **mando** y dos **sensores de proximidad.**

Debido al problema existente en algunas ciudades como Granada o Jaén, donde el tráfico en ciertas zonas de la ciudad está restringido sólo a personas residentes, se ha propuesto la utilización de un control de pilonas hidráulicas de doble efecto, las cuales son accionadas por personas residentes o autorizadas.

2.2. Parking

En este caso el lenguaje de programación utilizado es el LD (Diagrama de Contactos). El objetivo es el control de un parking y sus luces, con un número total de 10 plazas. Se ha tratado de ver cómo se pueden crear bloques de programa para la personalización de programas y así adaptarse a las necesidades de cada problema, para poder optimizar los recursos del sistema como son las salidas o entradas.

2.3. Ascensor

Una gran oportunidad de poner en práctica lo aprendido sobre Informática Industrial es un lenguaje basado en C. El lenguaje utilizado es lenguaje estructurado ST, donde se ha utilizado bucles, condicionales, temporizadores, con el fin de mostrar todo el potencial y la diversidad de opciones que permite CODESYS.

Este ejemplo de programa es debido a que en la actualidad cualquier software para un ascensor tiene un coste elevado y ésta reservado por cada fabricante, de forma que no se puede acceder al código que se tiene detrás. Gracias a este programa se podrían abaratar el coste que tiene un proyecto como es el ascensor de un edificio.

2.4. Control del motor de un tanque

En este ejercicio se pondrá en práctica lo aprendido en asignaturas como Ingeniería de Control o Control por Computador, donde además en ésta última una de las prácticas es precisamente el control de la posición de un motor.

En este caso se ha abordado la problemática que puede tener el control de una planta que tenga señales y variables a controlar, como sería el control de un tanque. Se pueden monitorizar el motor de llenado del tanque con la aplicación de un controlador tipo P, PI o PID.

En este caso se ha utilizado un controlador tipo PID. Se ha creado un bloque de funciones **FBD**, en el cual se ha introducido una función ya preestablecida llamada PID. Como una opción extra se ha creado la comunicación con una Raspberry Pi 3B, simulando que es el propio PLC.

2.5. Puerta de garaje

En este ejercicio se ha desarrollado el control de una puerta de garaje. Tendremos la opción de hacerla de manera automática o de manera manual, se utilizará el lenguaje de programación ***LD (****Diagrama de contactos****)*** y la visualización creada con un *bloque de funciones.*

Se ha decidido utilizar dos lenguajes de programación para mostrar la versatilidad que tiene el programa a la hora de poder mezclar varios lenguajes y poder interactuar entre ellos. Al igual que en ejercicios anteriores, aquí se ha decidido utilizar una puerta de garaje para ver de forma visual el potencial que tiene el programa a la hora de utilizar diferentes recursos como imágenes o bloques predefinidos.

2.6. Conversión Grados Fahrenheit – Celsius

Uno de las opciones que se planteó inicialmente era la posibilidad de si el programa era capaz de poder utilizar señales analógicas como entradas dentro del programa CODESYS, por ejemplo, para reaprovechar sensores ya existentes en algunas instalaciones.

Gracias a un módulo que viene incluido dentro de la biblioteca ***Útil, 3.5.11.0,*** la cual previamente se ha cargado en el programa, se ha aprovechado para crear la conversión de grados Fahrenheit a Celsius. Ya que tienen distintos rangos de trabajo, también se ha implementado para verlo de una manera visual un gráfico en tiempo real.

3. Ejercicio de Pilonas

Tal como se ha comentado previamente, el objetivo es controlar y limitar la entrada de vehículos en una zona. Regulado con dos pilonas hidráulicas controladas por un mando y varios sensores de proximidad. El funcionamiento sería el siguiente, se llega a la zona donde está la primera pilona y al accionar el mando se baja ésta. Al pasar esta primera pilona el sensor detectará nuestra posición y hará que suba la primera y baje la segunda pilona. Una vez se ha sobrepasado la segunda pilona, otro sensor detectará nuestra posición y hará que ésta se suba. En la Ilustración 28, se mostrará gráficamente esta situación. Siendo un ejemplo sencillo, la mayoría de las variables y funciones creadas, se utilizan para hacer posible la visualización.

Este ejercicio se ha resuelto mediante un programa compuesto por 5 POUs:

- PLC_PRG: es el POU que se ejecuta cíclicamente y hace las llamadas al resto de POUs del proyecto. En este ejercicio este ejercicio está escrito en lenguaje SFC, Grafico Funcional Secuencia.
- Visu_Movimiento_pilona1: es el POU encargado del movimiento en la visualización de la primera pilona. Está escrito en lenguaje FBD, Diagrama de Bloques Funcionales.
- Visu_Movimiento_pilona 2: es el POU encargado del movimiento en la visualización de la segunda pilona. Está escrito en lenguaje FBD, Diagrama de Bloques Funcionales.
- GVL: es una POU sin lenguaje de programación, donde se declaran las variables globales de ahí sus iniciales *Global Variable List.*
- GlobalImagePool: es una POU sin lenguaje de programación, donde se declaran todas las imágenes utilizadas en el programa.

En el diseño se han tenido en cuenta variables que se utilizan físicamente y variables de programa para la simulación, en la Tabla 1 y Tabla 2 se tiene un resumen de las variables utilizadas:

Variables utilizadas Físicamente	Descripción
Fis_Mando	Contacto físico del mando, utilizado para la transición en el programa principal
Fis_sensor1	Contacto físico del Sensor 1
Fis_sensor2	Contacto físico del Sensor 2
fis_motor_subir_pilona1	Contacto del motor de subida de la pilona 1
fis_motor_bajar_pilona1	Contacto del motor de bajada de la pilona 1
fis_motor_subir_pilona2	Contacto del motor de subida de la pilona 2
fis_motor_bajar_pilona2	Contacto del motor de bajada de la pilona 2

Tabla 1. Tabla de las variables utilizadas físicamente en el ejercicio de las pilonas

Variables utilizadas para Visualización	Descripción
Visu_inicio	Botón utilizado para iniciar la visualización
Visu_pilona1_subida	Contacto para indicar que la pilona 1 está subida en la visualización
visu_pilona1_bajada	Contacto para indicar que la pilona 1 está bajada en la visualización
GVL.visu_salida_cont_pilona1	Contacto utilizado para la salida del contador interno *visu_movimiento_pilona2* de la pilona 1
GVL.visu_salida_anim_pilona1	Contacto utilizado para el movimiento en la animación de la pilona 1
Visu_pilona2_subida	Contacto para indicar que la pilona 2 está subida en la visualización
visu_pilona2_bajada	Contacto para indicar que la pilona 2 está bajada en la visualización
GVL.visu_salida_cont_pilona2	Contacto utilizado para la salida del contador interno del bloque *visu_movimiento_pilona2* de la pilona 2
GVL.visu_salida_anim_pilona2	Contacto utilizado para el movimiento en la animación de la pilona 2

Tabla 2. Tabla de variables utilizadas en el ejercicio de las pilonas para la animación de programa

3.1 Programa principal

En el programa principal lo primero que se tiene, es la ventana de declaración de variables. Tal y como se ve en la Ilustración 22. Seguido de esta ventana, el esquema de etapas y transiciones por las que pasa el programa. Este se iniciaría con la etapa inicial, donde no se carga ningún valor inicial, ya que no es necesario en este caso, a la espera de que se pulse el botón en la visualización ***visu_inicio***. Esta variable se representará con un botón en el esquema de visualización. Tras el franqueo de esta transición se pasa a la etapa ***Step0*** cuya acción asociada, al igual que las siguientes, viene precedida del indicador ***N.*** Esa ***N*** indica, que esas acciones están funcionando mientras este activa la etapa. Al arrancar esta etapa se hace la llamada a dos bloques de funciones del tipo FBD, llamados ***Visu_movimiento_pilona1*** y ***Visu_movimiento_pilona2,*** gracias a las variables declaradas en la parte superior llamadas ***visu_p1*** y ***visu_p2***. Como se puede apreciar es necesario la llamada a los bloques de funciones de cada pilona a través de las variables declaradas en el programa principal. Hablaremos en el siguiente apartado de estos bloques de funciones, creados para el movimiento en la visualización de las pilonas y el conexionado físico de los motores.

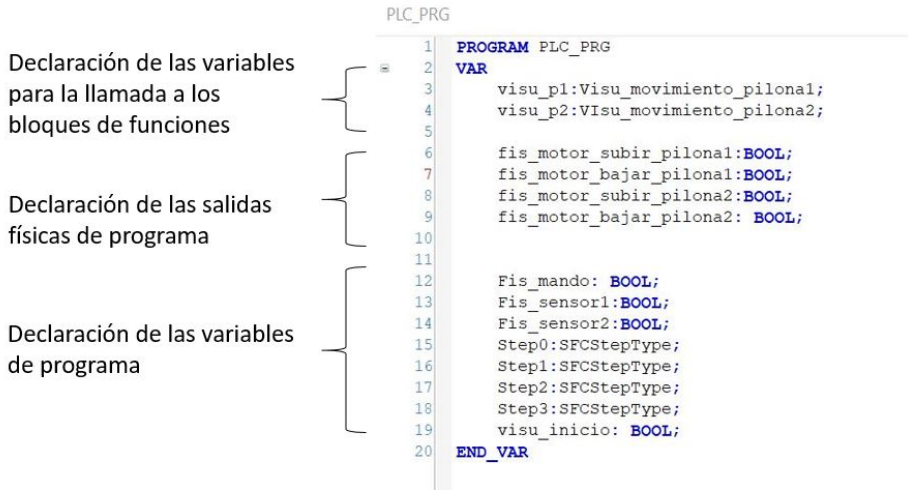

Ilustración 22. Declaración de las variables utilizadas en el programa principal pilonas que es de tipo SFC

Estos bloques de funciones, activan las pilonas y provocan el movimiento de estas en la visualización. El funcionamiento es tal y como se ha visto antes, pero si se focaliza en las variables físicas que se tienen declaradas, sería de la siguiente manera: cuando se acerca un coche y acciona el botón, llamado ***Fis_mando***, este activa el motor de bajada de la pilona 1, llamado ***fis_motor_bajar_pilona1*** y la pilona 2 se mantiene subida. En el momento que el sensor 1, llamado ***Fis_sensor1,*** detecta el coche, sube la pilona 1, a través del contacto, llamado ***fis_motor_subir_pilona1,*** y baja la pilona 2, gracias al contacto llamado ***fis_motor_bajar_pilona2.*** Cuando el coche pasa por el sensor 2, llamado ***Fis_sensor2,*** activa el motor de subida de la pilona 2, llamado ***fis_motor_subir_pilona2***. En la Ilustración 23 se muestra de manera esquemática lo descrito:

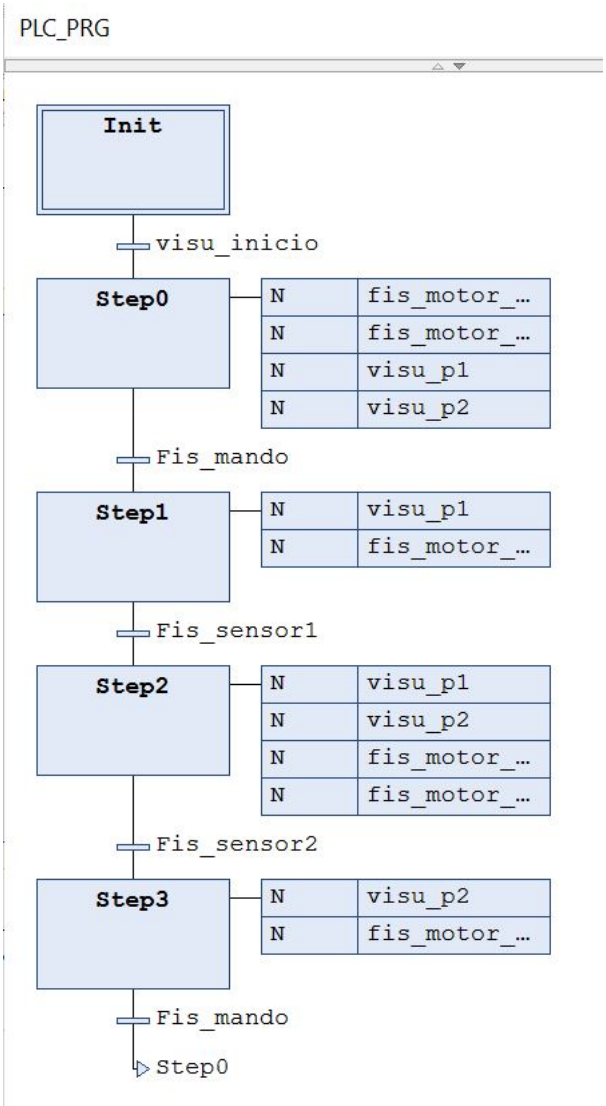

Ilustración 23. Esquema de programa SFC pilonas

3.2. Bloque FBD, Diagrama de Bloques de Funcionales visu_movimiento_*pilona1* y visu_movimiento_*pilona2*.

Vamos a trabajar con dos bloques de funciones, los cuales controlan el movimiento de las pilonas en la visualización, llamados ***visu_movimiento_pilona1*** y ***visu_movimiento_pilona2.*** Cada uno tienen sus salidas conectadas a unas variables globales, llamadas ***GVL.visu_salida_anim_pilona1*** y ***GVL.visu_salida_anim_pilona2***, que son utilizadas para la visualización. El programa CODESYS, no permite crear un único bloque y que sea llamado por distintas partes del programa para hacer el movimiento de las distintas pilonas. Se han tenido que crear dos bloques de funciones iguales, pero cada uno personalizado con las variables utilizadas de cada pilona.

Se ha utilizado un generador de pulsos, para simular el movimiento de las pilonas en la visualización. Éste ha sido creado con una función llamada ***BLINK***. Tal y como se ve en la Ilustración 24.

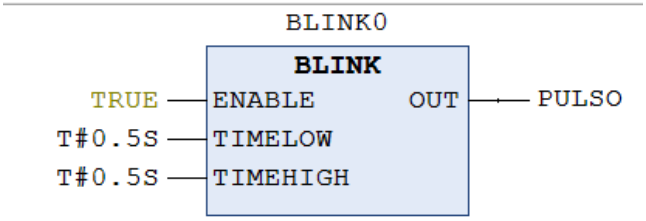

Ilustración 24. Función generadora de pulsos

Este generador de pulsos trabaja dando pulsos a un contador, llamado ***CTUD_0***, el cual se utiliza para el incremento y decremento de una variable Global llamada ***GVL.visu_salida_cont_pilona1***, como se puede apreciar en la Ilustración 25***.***

Además, se pueden ver tres líneas de red, como las llama el programa, o líneas de código. En la primera línea se ve el generador de pulsos ya mencionado, ***generadorpulsos***.

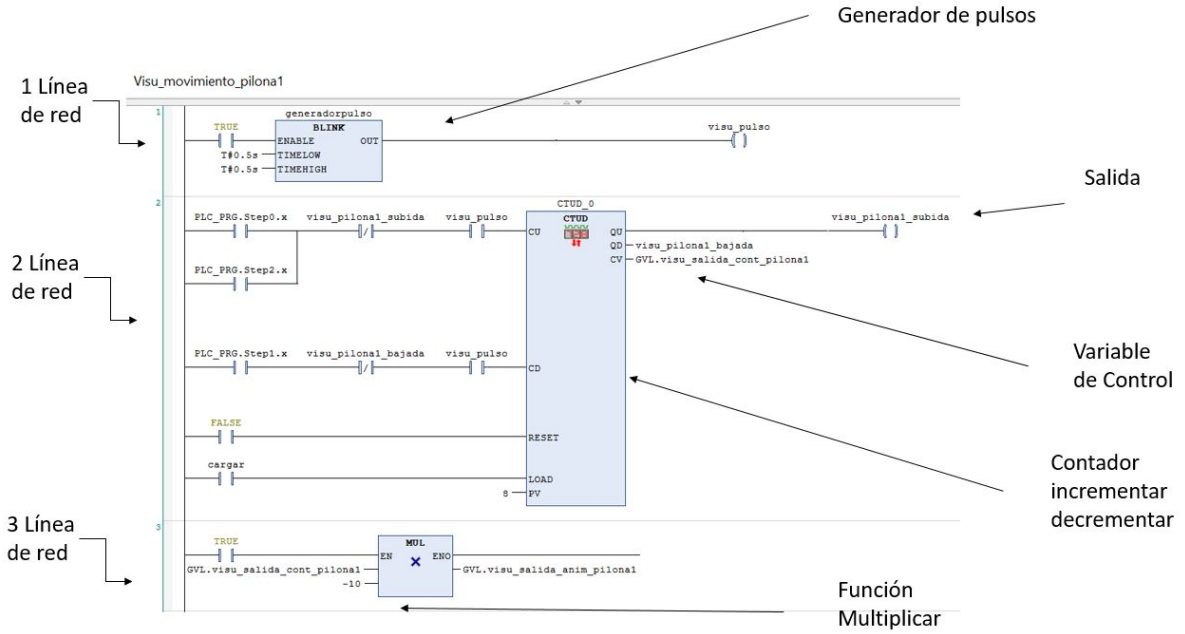

Ilustración 25. Bloque de función FBD llamado visu_movimiento_pilona1

En la segunda línea, hay un contador llamado ***CTUD_0.*** El cual se utiliza tanto para subir la pilona, como para bajada de la pilona en la visualización. Gracias a los dos contactos de la ***etapa 0*** y la ***etapa 2***, que van conectados en la entrada ***CU***, se produce la subida de la pilona, incrementando. También se tiene un contacto cerrado llamado ***visu_pilona1_subida.*** Esta variable se utiliza para indicar que la pilona esta subida del todo. En el caso del decremento del contador (bajada de la pilona en la visualización), se utiliza la entrada ***CD***. A esta entrada, se le conecta la variable ***PLC_PRG.Step1.x*** de la **etapa 1**. Se ha utilizado otro contacto cerrado llamado ***visu_pilona2_bajada***, el cual indica que la pilona ya ha bajado del todo***.*** En las salidas tenemos tres variables, la primera salida del contador por temas de diseño. El programa obliga a conectar la primera salida del contador una bobina, en caso de no colocarla da error de compilación, la segunda y tercera salida no es necesario conectar una bobina, con poner el nombre de la variable es suficiente. Por último, las tres últimas entradas que quedan son ***RESET***, la cual se le tiene puesta siempre a negativo, indicándolo con ***FALSE***, para no utilizarla nunca. Otra entrada, por si se quiere cargar un valor directamente llamada ***cargar,*** y la entrada ***PV***, la cual se le indica el valor 8 (son los pulsos que se han contabilizado para el movimiento de la pilona).

Para terminar, queda la tercera línea, en ella se ha colocado una función llamada ***MUL,*** esta función se ha utilizado para poder ver visualmente el movimiento de la pilona en la visualización y sea algo más rápido, al poder incrementar el valor de la variable de salida ***GVL.visu_salida_cont_pilona1,*** que es controlada por el contador ***CTUD_0.*** La primera entrada de la función es de habilitación, la cual queda habilitada siempre indicándolo con la palabra ***TRUE***, en el contacto colocado***.*** La segunda entrada de la función***,*** es para indicar el valor por el que se quiere multiplicar esa variable de entrada, en este caso ***-10***, indicar que se ha utilizado el valor ***-10*** como factor de escalado gráfico, siendo negativo porque la ordenada 0 está en la parte superior de la pantalla***.*** Y en la salida se ha utilizado otra variable, llamada ***GVL.visu_salida_anim_pilona1.***

3.3. Lista de Variables Globales GVL

Se han utilizado variables Globales, ya que se hace la llamada a ellas desde varias partes del programa. El hecho de utilizar este tipo de variable permite al usuario no duplicar variables que físicamente son las mismas. Estas se ubican dentro de una POU de variables Globales llamada ***GVL***, tal y como se ve en la Ilustración 26:

GVL

```
1  VAR_GLOBAL
2      visu_salida_anim_pilona1: INT;
3      visu_salida_anim_pilona2: INT;
4      visu_salida_cont_pilona1:INT;
5      visu_salida_cont_pilona2: INT;
6  END_VAR
```

Ilustración 26. Ventana con la Lista Variables Globales

Las variables que se han utilizado en el ejercicio, son de tipo entero ***INT,*** las cuales se utilizan en la visualización para el movimiento de objetos.

3.4. Visualización

En la visualización, la mayor parte de los bloques creados en el ejercicio, se utilizan en ella, para crear la animación de movimiento. El programa permite trabajar con objetos ya prestablecidos en sus bibliotecas, o poder insertar imágenes, incluso crearlos a través de sus formas básicas que tienen (rectángulo, elipse, línea, etc.).

Los objetos tienen dos tipos de movimiento. ***Movimiento absoluto,*** que mueve todo el objeto sin fijar ninguna coordenada de él. M***ovimiento relativo***, el cual fija uno de los lados y hace el movimiento del otro alterando las dimensiones del objeto.

Para este caso, se ha utilizado el ***movimiento absoluto*** en las pilonas, las cuales son dos imágenes introducidas en la visualización. Para poder introducir imágenes en la visualización, se tiene que ir en la ventana de visualización, a la columna de la derecha, en el menú desplegable que aparece. Lo primero que hay que hacer es insertar la imagen, dentro de ese menú, clicamos en *Herramientas* > *Base* > *Imagen* y una vez se le indica el lugar donde se quiere colocar la imagen, se indica si la imagen a introducir es una de las que ya tiene cargadas en el proyecto o cargar una imagen nueva.

En este caso, al ser imágenes personalizadas, se tienen que introducir manualmente. Para ello, cuando aparece la ventana con las imágenes ya precargadas, se le da al botón de ***cancelar***. Se pulsa sobre la casilla ***ID estático*** y se le introduce un nombre a esa imagen, la cual se almacena en una POU global llamada ***GlobalImagePool.***

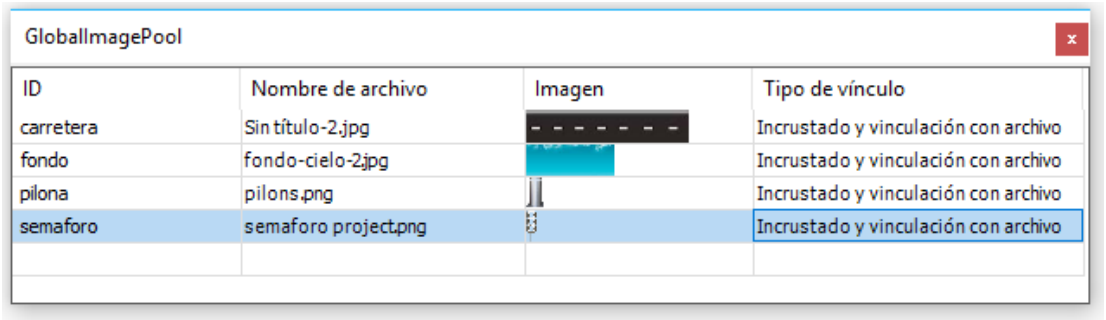

Ilustración 27. Ventana Imágenes Globales ¨GlobalImagePool¨

Una vez definido el nombre, se despliega una ventana para indicar la imagen a cargar. Dentro de la POU ***GlobalImagePool*** se puede seleccionar la forma en la que se carga la imagen. En este caso, para no tener que estar recordando la ruta del archivo o en caso de si se abre en otro ordenador o dispositivo, se ha utilizado ***Incrustado y vinculación con archivo***, lo cual quiere decir que esa imagen se queda cargada en el proyecto para siempre. A continuación se puede ver las imágenes cargadas en el ejercicio, en la Ilustración 27:

En la visualización, se ha representado la variable ***Visu_inicio*** con un botón, la cual es utilizada para inicializar el sistema. Dos sensores para la detección de coches, llamados ***Fis_sensor1*** y ***Fis_sensor2***, están representados con unos botones de la biblioteca que viene predefinida***.*** La variable ***Fis_mando*** representada con un pulsador de color verde, para activar la bajada de la primera pilona, y las pilonas las cuales están representadas con imágenes insertadas, a continuación se puede ver más claramente en la Ilustración 28:

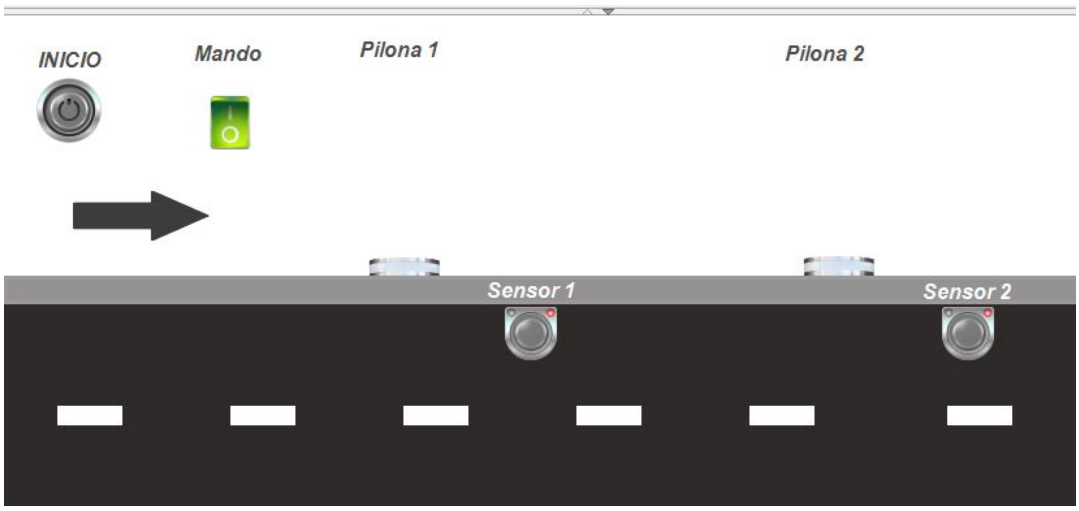

Ilustración 28. Ventana de visualización del ejercicio pilonas

3.5. Problemas encontrados

Uno de los problemas encontrados a la hora de programar, ha sido que un bloque de funciones, no es posible reutilizarlo para otras salidas. Con este lenguaje cada bloque es único, de ahí que se hayan creado dos bloques iguales para el movimiento en la visualización de cada una de las pilonas.

El programa en ciertas funciones te obliga a conectar las variables a una bobina/contacto de salida, para poder simularlo como en el caso del contador de los bloques de funciones utilizado para la visualización.

Para poder hacer la llamada a los bloques de funciones utilizados, siempre se ha de asociar ese bloque de función como puede ser por ejemplo ***visu_movimiento_pilona1*** está asociado dentro del programa principal a una variable llamada ***visu_p1.***

4. Ejercicio Parking

En este ejercicio se ha tratado el funcionamiento que tiene un parking y el control de sus luces. El proceso sería el siguiente: Un primer sensor detecta la llegada de un coche, y como el parking tiene plazas libres, lo deja pasar. Una vez ha sobrepasado el segundo sensor, se incrementa el número de plazas ocupadas. La luz de cada plaza que tiene el parking se ilumina conforme se incrementa el número de plazas ocupadas. Cuando se llega a ocupar el total de plazas existentes, en este caso diez, el letrero de parking cambia a **completo** y el semáforo pasa a rojo, no dejando entrar ningún coche hasta que no salga alguno de los que ya han entrado. Se detectará que un coche ha salido cuando pasa primero por el segundo sensor y después por el sensor uno.

Se ha resuelto mediante un programa compuesto de 4 POUs:

- PLC_PRG: es el POU que se ejecuta cíclicamente y hace las llamadas al resto de POUs del proyecto. En este ejercicio este programa está escrito en lenguaje FBD, básicamente agrupa las otras dos POUs utilizadas.
- PARKING: es un POU también en lenguaje FBD, Diagrama de Bloques funcionales, en el cual tiene asociado las salidas de control del semáforo, y los paneles luminosos.
- LUCES: es el último POU utilizado, de lenguaje FBD, tiene asociada todas las luces del interior del parking para su control.
- GVL: es una POU sin lenguaje de programación, donde se declaran las variables globales de ahí sus iniciales *Global Variable List.*

En el diseño se han tenido en cuenta variables que se utilizan físicamente y variables que se utilizan en la visualización únicamente. A continuación, se muestra las distintas variables empleadas en el conexionado físico en la Tabla 3, como las utilizadas para la visualización en la Tabla 4.

Variables utilizadas Físicamente	Descripción
Fis_SENSOR1	Contacto físico del Sensor 1
Fis_SENSOR2	Contacto físico del Sensor 2
Fis_RESET	Contacto físico botón Reset
Fis_LIBRE	Contacto físico del cartel luminoso
Fis_COMPLETO	Contacto físico del cartel luminoso
Fis_Luz_1	Contacto físico de la luz del parking numero 1
Fis_Luz_2	Contacto físico de la luz del parking numero 2
Fis_Luz_3	Contacto físico de la luz del parking numero 3
Fis_Luz_4	Contacto físico de la luz del parking numero 4
Fis_Luz_5	Contacto físico de la luz del parking numero 5
Fis_Luz_6	Contacto físico de la luz del parking numero 6
Fis_Luz_7	Contacto físico de la luz del parking numero 7
Fis_Luz_8	Contacto físico de la luz del parking numero 8
Fis_Luz_9	Contacto físico de la luz del parking numero 9
Fis_Luz_10	Contacto físico de la luz del parking numero 10
fis_luz_roja_entrada	Contacto físico de la luz roja del semáforo de entrada
fis_luz_verde_entrada	Contacto físico de la luz verde del semáforo de entrada
fis_luz_roja_salida	Contacto físico de la luz roja del semáforo de salida
fis_luz_verde_salida	Contacto físico de la luz verde del semáforo de salida
Fis_sensor_luz_1	Contacto físico del sensor de presencia de la luz 1
Fis_sensor_luz_2	Contacto físico del sensor de presencia de la luz 2
Fis_sensor_luz_3	Contacto físico del sensor de presencia de la luz 3
Fis_sensor_luz_4	Contacto físico del sensor de presencia de la luz 4
Fis_sensor_luz_5	Contacto físico del sensor de presencia de la luz 5
Fis_sensor_luz_6	Contacto físico del sensor de presencia de la luz 6
Fis_sensor_luz_7	Contacto físico del sensor de presencia de la luz 7
Fis_sensor_luz_8	Contacto físico del sensor de presencia de la luz 8
Fis_sensor_luz_9	Contacto físico del sensor de presencia de la luz 9
Fis_sensor_luz_10	Contacto físico del sensor de presencia de la luz 10

Tabla 3. Variables físicas y contactos utilizados en el ejercicio del parking

Variables utilizadas para Visualización	Descripción
visu_entra	Marca interna para indicar que un coche está entrando
visu_sale	Marca interna para indicar que un coche está saliendo
visu_sensor1_up	Contacto de flanco interno del sensor 1
visu_sensor1_down	Contacto de flanco interno del sensor 1
visu_sensor2_up	Contacto de flanco interno del sensor 2
visu_sensor2_down	Contacto de flanco interno del sensor 2
visu_Luz_1	Marca interna para indicar que la luz 1 está encendida en la visualización
visu_Luz_2	Marca interna para indicar que la luz 2 está encendida en la visualización
visu_Luz_3	Marca interna para indicar que la luz 3 está encendida en la visualización
visu_Luz_4	Marca interna para indicar que la luz 4 está encendida en la visualización
visu_Luz_5	Marca interna para indicar que la luz 5 está encendida en la visualización
visu_Luz_6	Marca interna para indicar que la luz 6 está encendida en la visualización
visu_Luz_7	Marca interna para indicar que la luz 7 está encendida en la visualización
visu_Luz_8	Marca interna para indicar que la luz 8 está encendida en la visualización
visu_Luz_9	Marca interna para indicar que la luz 9 está encendida en la visualización
visu_Luz_10	Marca interna para indicar que la luz 10 está encendida en la visualización
GVL.PLAZAS_OCUPADAS	Variable para indicar en la visualización el número de plazas ocupadas
control	Variable interna para hacer la llamada al bloque PARKING desde el programa principal
lucenarias	Variable interna para hacer la llamada al bloque LUCES desde el programa principal
Visu_NPLAZAS	Variable para asignar el número de plazas que tendrá el parking
visu_luz_roja_salida	Variable de salida para la luz roja del semáforo de salida
visu_luz_verde_salida	Variable de salida para la luz verde del semáforo de salida
visu_luz_roja_entrada	Variable de salida para la luz roja del semáforo de entrada
visu_luz_verde_entrada	Variable de salida para la luz roja del semáforo de entrada

Tabla 4, Variables y marcas internas utilizadas para la visualización o los bloques internos de programa

4.1. Programa Principal

En el programa principal, cuando aparece el tipo de lenguaje a utilizar se selecciona ***FBD***, Diagrama de Bloques Funcionales, y se crean dos bloques más, llamados ***PARKING*** y ***LUCES***. Tanto el bloque ***LUCES*** como el bloque ***PARKING***, se crean en el árbol de proyecto a la izquierda, sobre la pestaña ***Application > Agregar objeto > Agregar POU*** y seleccionamos *Bloque de funciones.* A los bloques creados se le indican cuáles serán sus salidas y sus entradas dentro de cada bloque, en la ventana superior de programa. Las salidas de los bloques de funciones FBD creados, se conectan a las salidas de ***Fis_LIBRE***, ***Fis_COMPLETO, GVL.PLAZAS_OCUPADAS*** y a la serie de luces enumeradas de 1 a 10.

Se tiene una variable llamada ***control,*** la cual es la que hace la llamada al bloque ***PARKING***. A este bloque se le conectan las entradas de los dos sensores, ***Fis_SENSOR1, Fis_SENSOR2,*** el número de plazas, ***visu_NPLAZAS,*** y un botón de reset, llamado ***Fis_RESET***, por si en algún caso fuera necesario resetear el parking. Se hace la declaración de todas estas variables tal y como se ve en la Ilustración 30.

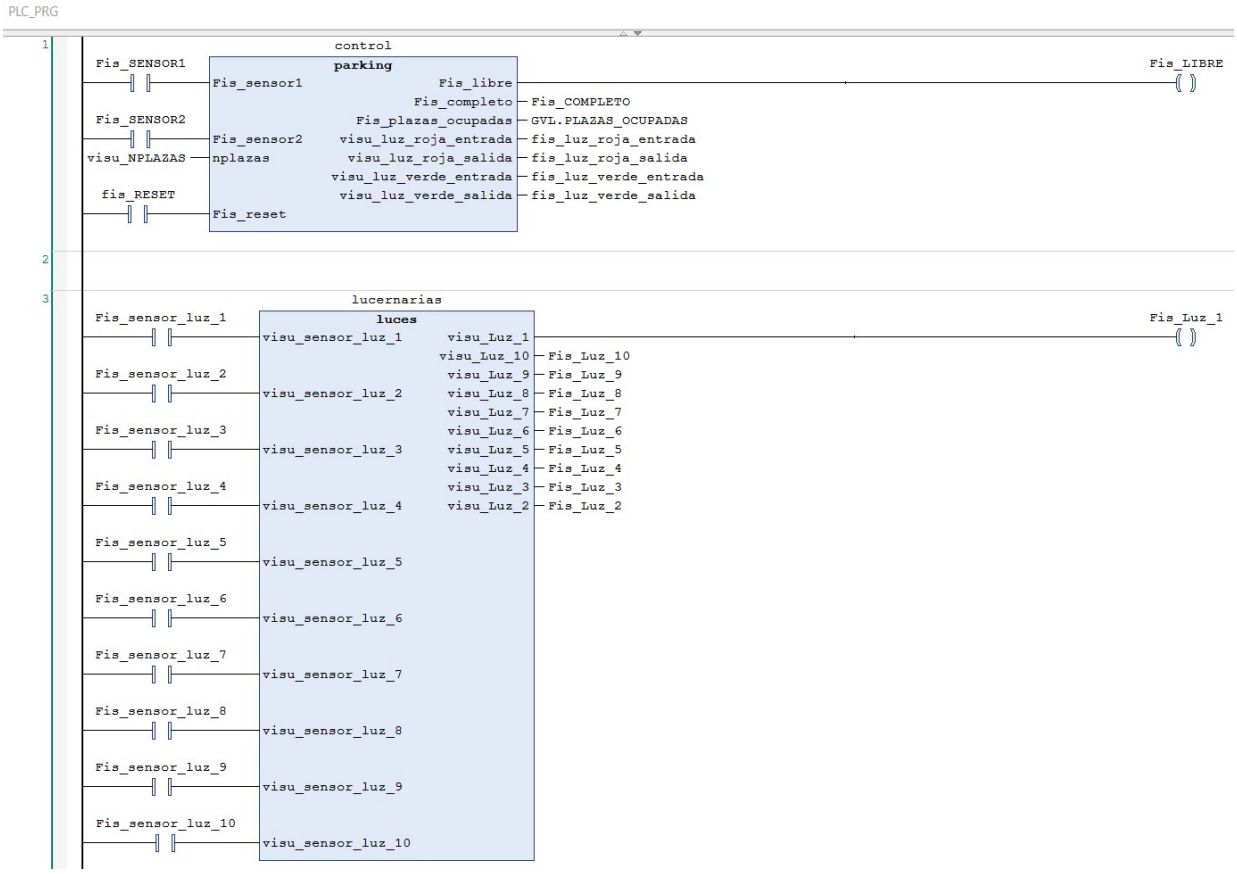

Ilustración 29. Diagrama programa principal, donde se tienen los bloques personalizados

Se tiene otra variable llamada ***lucernarias*** que hace la llamada al bloque de funciones ***LUCES*****,** tal y como se ve en la Ilustración 29***.***

El funcionamiento del parking consiste en el siguiente: se tiene un solo carril por el que entran y salen los coches, monitorizado con dos sensores que controlan cuando un coche ha entrado y dos semáforos para controlar el flujo de coches en el carril, en el exterior hay un semáforo para permitir la entrada de coches. Éste indica el número de plazas ocupadas y si está libre o no. También se tiene un panel luminoso que muestra desde fuera si ese parking está libre o no. Ya dentro del parking gracias a las luces se indican que plazas estarían ocupadas.

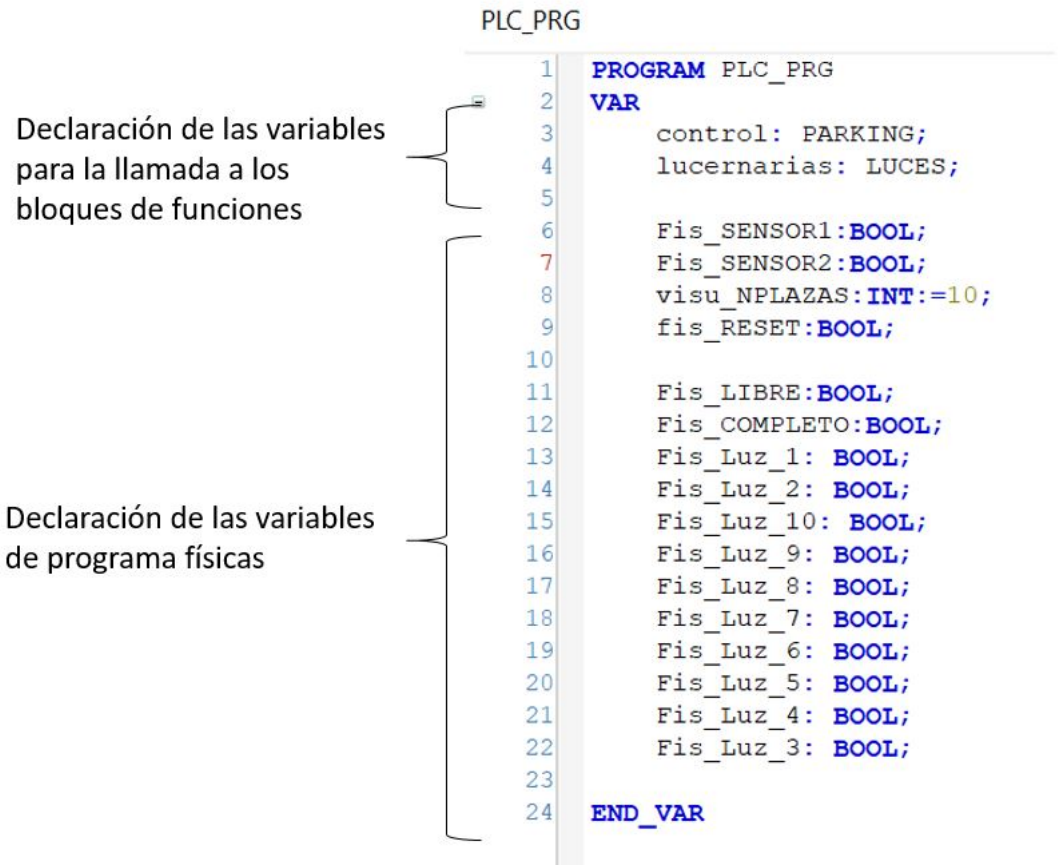

Ilustración 30. Ventana superior para la declaración de las variables en el programa principal del Parking

4.2. Bloque FBD, Diagrama de Bloque Funcional PARKING

En este bloque se hace la configuración para utilizar los dos sensores de proximidad, sin necesidad de tener más sensores para detectar la salida o entrada de un coche. Gracias a los contactos de flanco tanto de subida como de bajada, y los módulos que ya hay predefinidos, para hacer que una entrada a dicho modulo pueda convertirse en un flanco, llamados ***R_TRIG***, los cuales se han utilizado en este bloque. Para que un contacto sea de tipo flanco, a su entrada tiene que llevar asociada una ***P,*** la cual indica que a partir de ahora ese contacto es con flanco de subida, o una ***N*** en el caso de los flancos de bajada. Para el control de los semáforos tantos de entrada como de salida, se han utilizado los contactos de los biestables ***visu_entra*** y ***visu_sale***. A continuación se puede ver más claramente en la Ilustración 31.

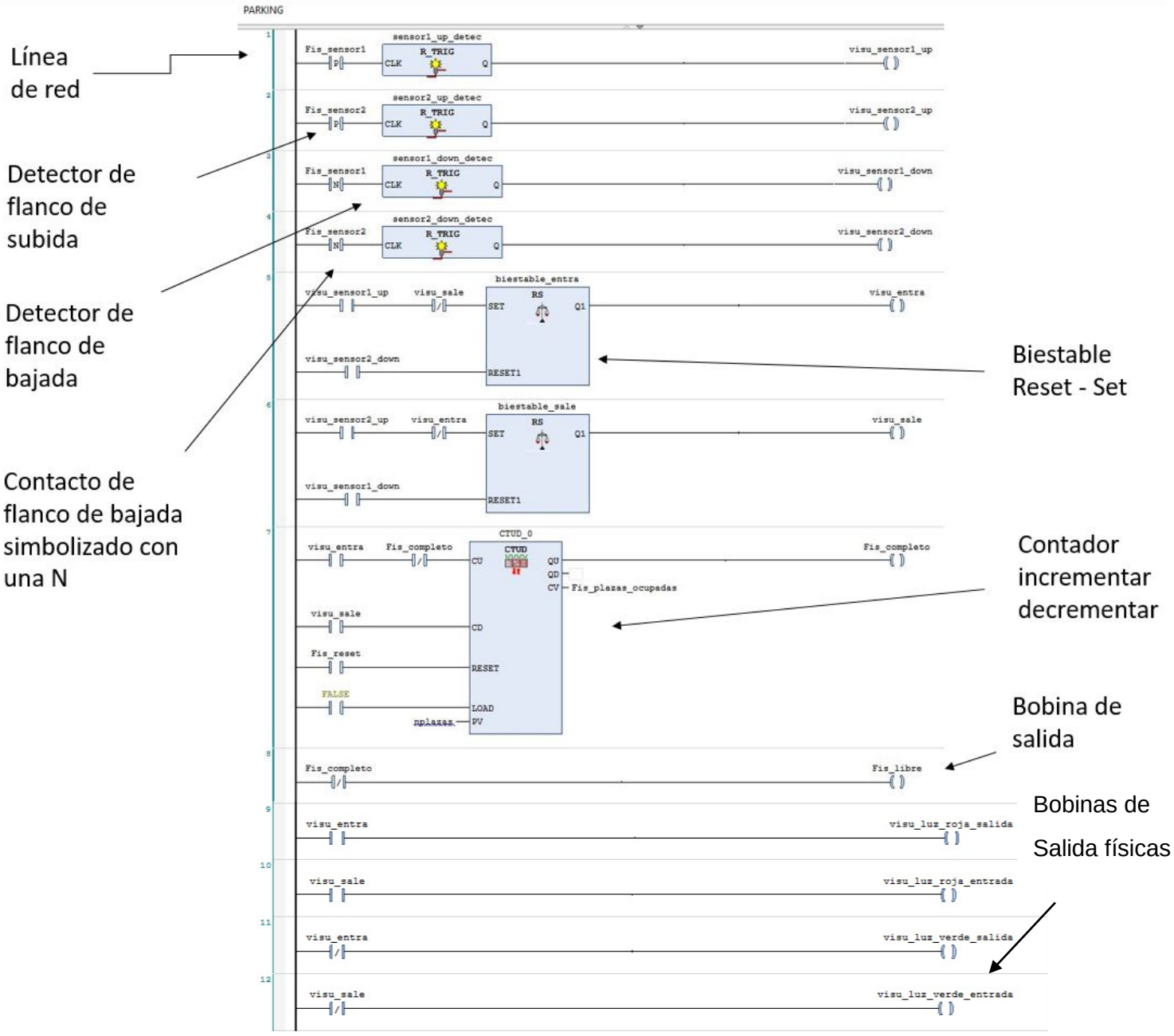

Ilustración 31. Diagrama bloque de funciones Parking

Se han creado cuatro módulos, llamados ***R_TRIG***, para identificar si el flanco es de subida o bajada. Esos módulos activan unos biestables de tipo Reset/Set, llamados ***RS***, estos biestables se utilizan para indicar si un coche está entrando o saliendo. Se han tenido que crear todas estas funciones ya que, a la hora de simular el programa en la visualización, no era posible utilizar un contacto simple de los sensores de presencia.

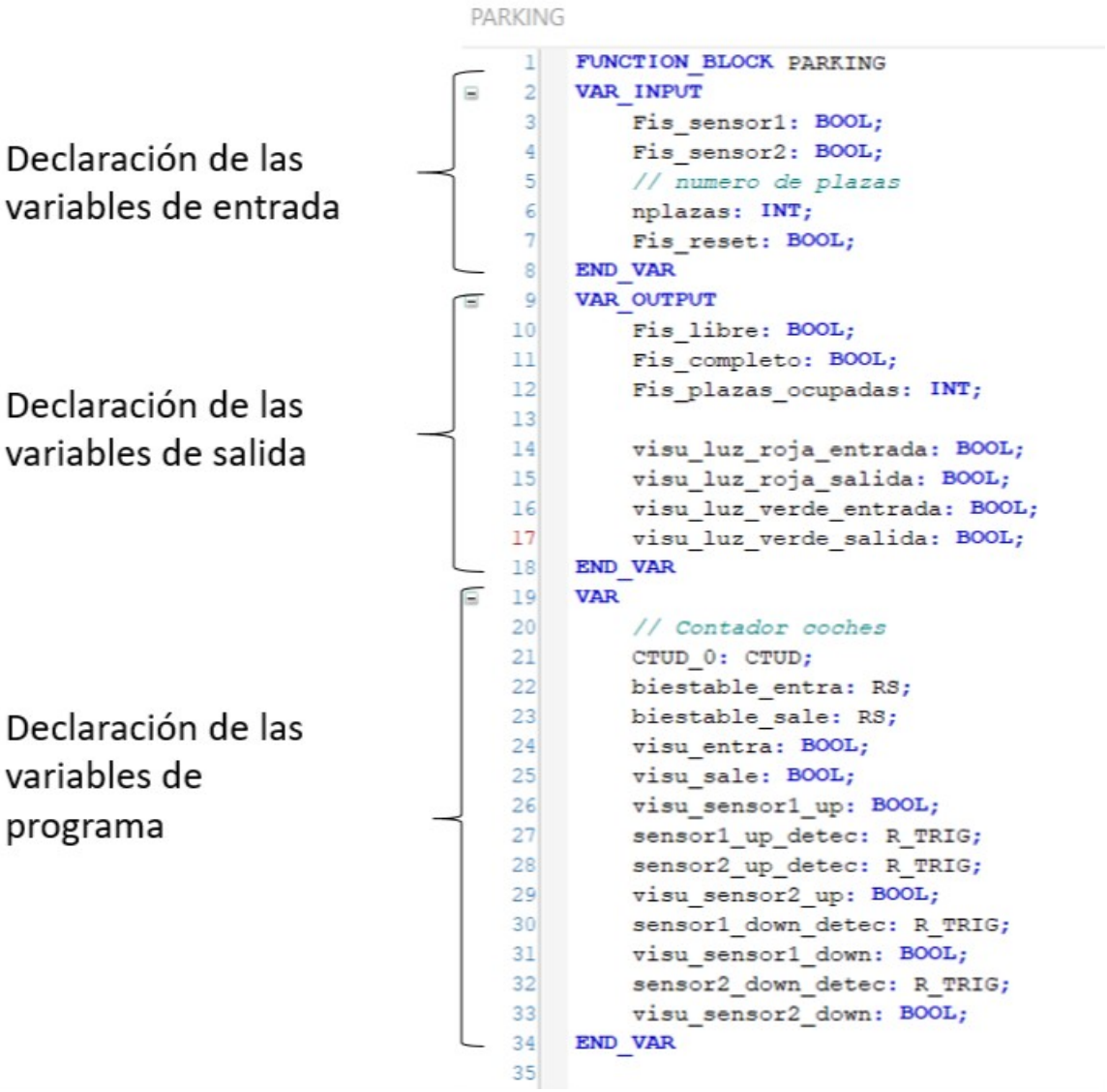

Ilustración 32. Variables bloque función PARKING

Todas estas variables creadas se usan en el contador ***CTUD_0,*** el cual posibilita incrementar el número de plazas hasta que el parking está completo, a través de la variable, llamada ***Fis_completo***, como se puede ver en la Ilustración 32.

4.3. Bloque FBD, Diagrama de Bloque Funcional LUCES

Este bloque se utiliza para el control de luces, en el cual se han utilizado sensores de presencia**.** Los cuales están conectados a cada una de las luces colocadas en la visualización, las salidas de estos son de tipo *BOOLEANO*. A continuación el esquema en la Ilustración 33.

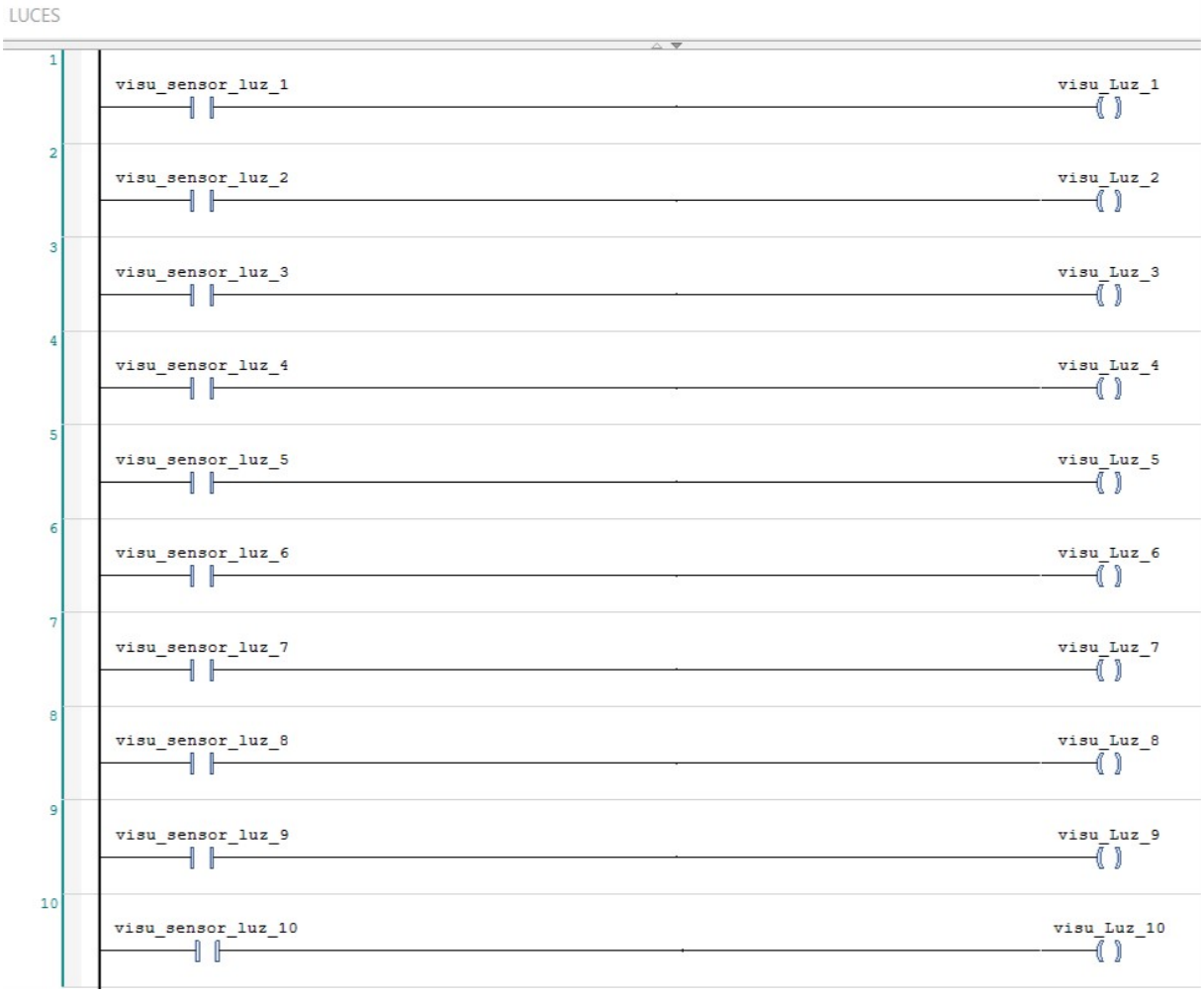

Ilustración 33. Diagrama Bloque de Funciones Luces, utilización de sensores de presencia para las luces.

4.4. Visualización

Para la visualización se han utilizado botones, que vienen preestablecidos en la librería del propio programa, y objetos como imágenes para simular la carretera o paneles luminosos. El semáforo tiene dos luces, una verde para cuando está libre y otra roja cuando el parking está completo, los otros semáforos controlan el flujo de coches en el carril, para nunca encontrar dos coches dentro del mismo. También se indica el número de plazas que están ocupadas, en el hueco central del semáforo. Además, se ha colocado un panel luminoso que muestra cuando está libre o completo. A continuación, en la Ilustración 34, se puede ver la disposición que se ha planteado.

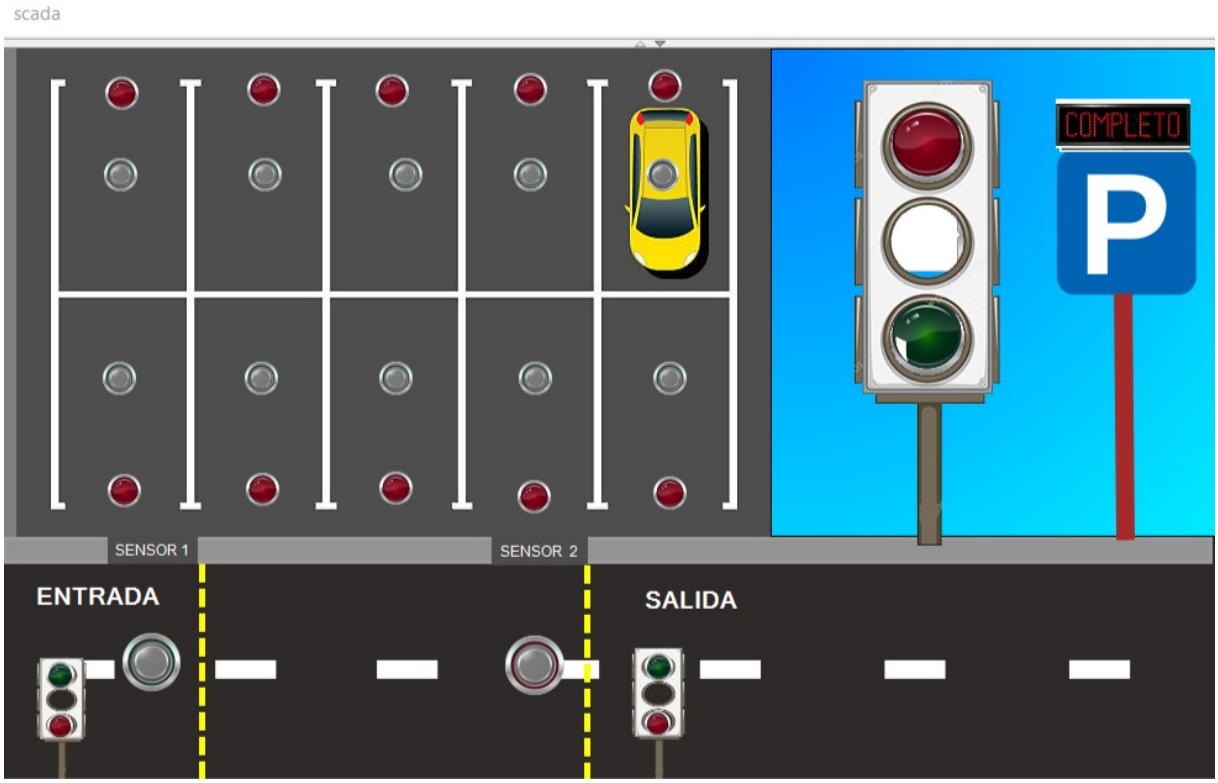

Ilustración 34. Ventana de visualización del programa Parking

4.5. Problemas encontrados

A la hora de visualizar, con solo dos sensores simular la entrada o salida de un coche, se han tenido que crear cuatro funciones de flancos, dos para cada sensor, de subida como de bajada, ya que el programa no era capaz de simular solo con biestables.

5. Ejercicio Ascensor

En este ejercicio se ha tratado el funcionamiento de un ascensor. El funcionamiento común de un ascensor tendría tres modos, en *stand-by o parado*, el cual es con sus puertas cerradas, a la espera de llamada desde cualquiera de sus plantas. El modo de subida, donde el ascensor se eleva desde alguna de las plantas inferiores, llega a la planta deseada, se abren sus puertas salen los pasajeros y se vuelven a cerrar. Por último, el modo de bajada estando la cabina en una planta superior es llamado desde algunas de las plantas inferiores, llega a la planta deseada, abre sus puertas, salen los pasajeros y vuelve a cerrarlas. Para cada planta se han utilizado dos pulsadores, uno para subir y otro para bajar. También se ha incluido una botonera dentro de la cabina, la cual queda inhabilitada, hasta que el ascensor es llamado desde alguna de las plantas, para que la visualización no carezca de sentido, ya que es físicamente imposible accionar un pulsador del interior de la cabina sin encontrarse nadie dentro. Lo primero que se hace es posicionar el ascensor, gracias al botón de la visualización llamado ***INICIO***, es decir el botón representa el inicio de programa, con la llamada del ascensor desde la planta baja, cuando el ascensor llega a esa planta se considera que está posicionado. Una vez ya en la planta 0, se puede hacer la llamada desde cualquiera de las plantas. El edificio consta de 4 plantas, tal y como se ve en la Ilustración 40. El ascensor si es llamado desde varias plantas, guarda en memoria las plantas llamadas. A continuación, se ha creado un flujograma para ver el funcionamiento del mismo en la Ilustración 35.

Este ejercicio se ha resuelto mediante un programa compuesto de un único POU:

- PLC_PRG: es el POU donde se ejecuta todo el programa. Está escrito en lenguaje ST, lenguaje estructurado. Todo el código para el control del ascensor, está incluido dentro de este POU.

En el diseño se han tenido en cuenta variables que se utilizan en la visualización y en el conexionado físico. A continuación, se muestra las distintas variables empleadas en la Tabla 5 y en la Tabla 6.

Variables utilizadas para la visualización	Descripción
boton_arriba	*Vector (Array)* utilizado tanto en la visualización como en el conexionado físico para los botones de llamada hacia arriba
boton_abajo	*Vector (Array)* utilizado tanto en la visualización como en el conexionado físico para los botones de llamada hacia abajo
etapas	*Vector (Array)* utilizado tanto en la visualización como en el conexionado físico para los botones de la cabina del ascensor
etapa_actual	Variable de tipo entero utilizada para saber la planta actual de la cabina
i	Variable de tipo entero utilizada en los bucles internos de programa
dir	Variable utilizada para saber la dirección que tiene el ascensor, 0 si es parado, 1 si es subiendo y 2 si es bajando
nivel	Variable para la altura que lleva la cabina en la visualización
start	Variable interna utilizada en el programa para arrancar los temporizadores
Contador0	Variable utilizada para contar el número de veces que se repite una acción en el movimiento dentro de una planta
TON_0	Temporizador utilizado para insertar un retraso al subir o bajar el ascensor entre planta y poder simular el movimiento del ascensor en la visualización
run	Variable utilizada para arrancar el programa y resetear todas las variables de programa utilizadas
Abrir_puertas	Marca interna que ordena la apertura de puertas
Puerta_izq	Variable utilizada en la visualización para la apertura y cierre de la puerta izquierda
Puerta_der	Variable utilizada en la visualización para la apertura y cierre de la puerta derecha
Puertas_mov	Marca interna para activar el temporizador del movimiento de apertura y cierre de las puertas
etapa	Variable para indicar la etapa hasta la cual tiene que ir el ascensor
Prev_dir	Variable que guarda la dirección previa que tenía el ascensor, con el objetivo de recordar
Contador3	Variable utilizada para contar el número de veces que se repite una acción en el movimiento de las puertas del ascensor
TON_2	Temporizador utilizado para insertar un retraso en la apertura y cierre de las puertas del ascensor y poder simular el movimiento en la visualización

Tabla 5. Variables de programa utilizadas en el ascensor para la visualización

Variables utilizadas fisicamente	Descripción
boton_arriba	*Vector (Array)* utilizado tanto en la visualización como en el conexionado físico para los botones de llamada hacia arriba
boton_abajo	*Vector (Array)* utilizado tanto en la visualización como en el conexionado físico para los botones de llamada hacia abajo
etapas	*Vector (Array)* utilizado tanto en la visualización como en el conexionado físico para los botones de la cabina del ascensor
Fis_motor_subir	Contacto de salida utilizado para el motor de subida
Fis_motor_bajar	Contacto de salida utilizado para el motor de bajada
fis_motor_abrir_puertas	Contacto de salida para la apertura de puertas
fis_motor_cerrar_puertas	Contacto de salida para el cierre de puertas

Tabla 6. Variables utilizadas para el conexionado físico del ascensor

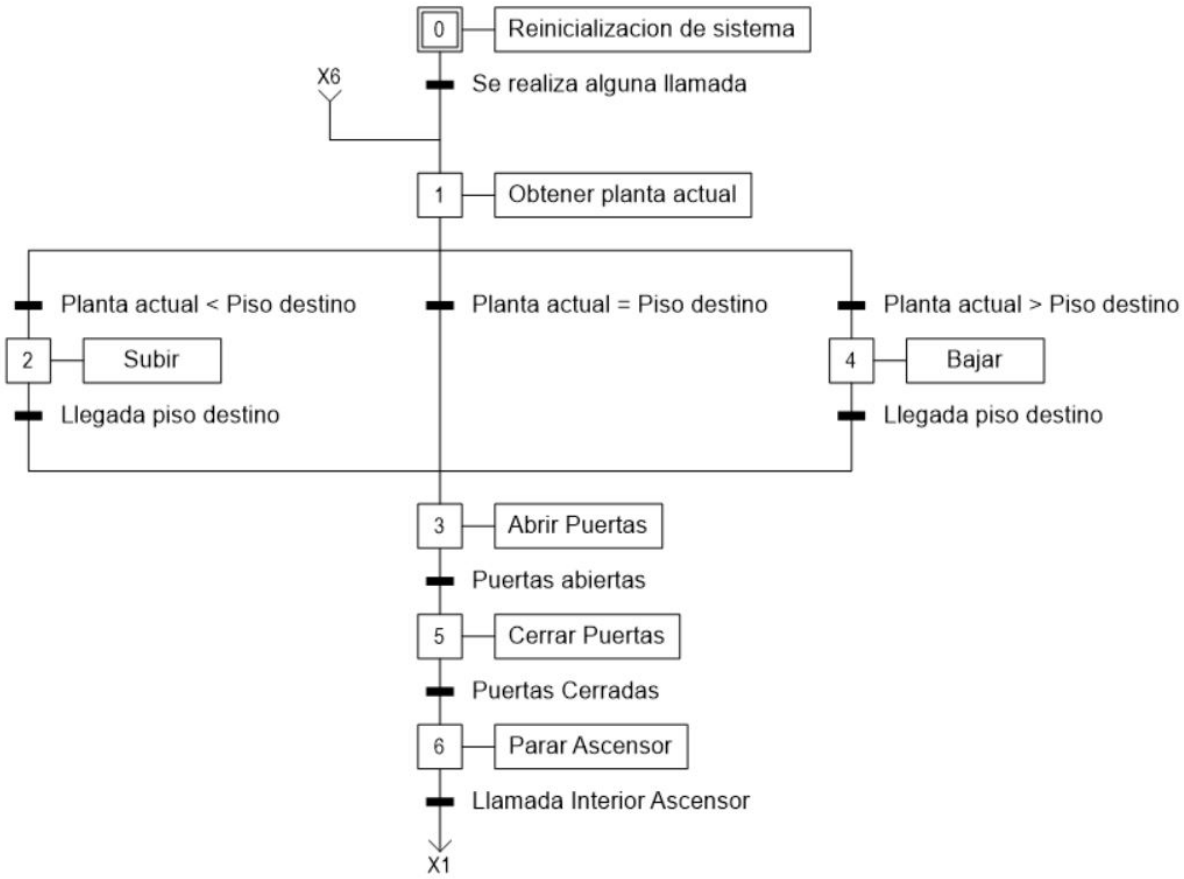

Ilustración 35. Grafcet general del movimiento universal del ascensor

5.1. Programa Principal

En este ejercicio se ha utilizado lenguaje estructurado ***ST,*** en el cual se han creado varias partes de programa o funciones dentro del programa principal, como son los condicionales, o los bucles utilizados. Todos estos bucles o condicionales han sido creados para poder controlar el ascensor tanto en la visualización como físicamente.

Antes de adentrarnos en el programa, se va explicar la estructura que tienen los bucles y los condicionales:

```
FOR (variable a controlar):= (valor de inicio) TO (valor máximo) BY (valor de incremento) DO
        ¨Aquí se indican las acciones a realizar¨
END_FOR
```

```
IF (variable)= TRUE THEN
        ¨Aquí se indican las acciones a realizar¨
END_IF
```

Debido a los problemas que pueden surgir a la hora de colocar los objetos, se ha decidido colocar un botón de ***INICIO***, asociado a una variable de tipo ***BOOL*** llamada ***run***. Así se tendrá siempre el mismo punto de partida y se resetearán los valores de las variables, en cada interacción. Tras todo esto, el ascensor estará listo para iniciar su correcto funcionamiento, comenzando la secuencia en la planta baja. El siguiente paso, para saber cuál es la dirección que tiene el ascensor, se ha creado una variable, llamada ***dir***. Esta variable indica la dirección, dependiendo si

tiene valor 0 (parado), 1 (subiendo) o 2 (bajando). En la Ilustración 36 a continuación mostrada, se puede ver parte del programa principal.

Inicialización de las variables

Bloque condicional tipo IF

Situar las etapas

Bloque bucle tipo For

Saber la dirección que tiene el ascensor

Bloque condicional tipo IF

```
IF run = TRUE THEN //inicializar programa//
    etapa_actual:=0;
    nivel:=537;
    run:=FALSE;
    dir:=0;
    abrir_puertas:=FALSE;
    TON_0(IN:=start, PT := T#0.05S);
    TON_2(IN:=puertas_mov, PT:=T#0.05S);
    contador0:= 0;
    contador3:= 0;
    puerta_izq:=0;
    puerta_der:=0;
    etapa:= 537;
    prev_dir:= 0;
END_IF

FOR i:=0 TO 4 BY 1 DO
    IF prev_dir = 1 THEN
        IF etapas[i]=TRUE THEN
            boton_arriba[i]:= etapas[i];
        END_IF
    ELSIF prev_dir =2 THEN
        IF etapas [i]= TRUE THEN
            boton_abajo[i]:= etapas[i];
        END_IF
    END_IF
END_FOR
//saber que direccion tiene//
IF dir = 0 AND (boton_arriba[1] OR boton_arriba[2] OR boton_arriba[3] OR boton_arriba[4]) THEN
    dir:= 1;
ELSIF dir= 0 AND (boton_abajo[0] OR boton_abajo[1] OR boton_abajo[2] OR boton_abajo[3] OR boton_abajo[4]) THEN
    dir:= 2;
ELSIF prev_dir = 1 THEN
    dir:= 1;
ELSIF prev_dir = 2 THEN
    dir:= 2;
END_IF
```

Ilustración 36. Ventana principal de programa entorno de edición

En el instante que el ascensor recibe la primera llamada, este compara su posición actual con la localización de la planta de destino y en función de si una planta inferior, superior o la misma, el ascensor bajará, subirá o abrirá las puertas al encontrarse en el mismo punto.

Para el movimiento del ascensor se han utilizado temporizadores. Uno de los temporizadores, llamado ***TON_0,*** el cual se utiliza para dar un retraso y poder percibir el movimiento en la visualización y otro temporizador, llamado ***TON_2***, da otro retraso para el movimiento de abrir y cerrar las puertas.

Para poder configurar los temporizadores en lenguaje estructurado se utiliza la siguiente nomenclatura:

TON_0 (**IN** := start , **PT** := T#1S) ;

Los temporizadores se activan a través de alguna variable, esta variable se coloca en la posición ***IN***, y para configurar el tiempo programado del temporizador, se indica en ***PT***.

La creación del temporizador **TON_0** de 0,05 segundos que se activará cuando las puertas del ascensor estén en movimiento, y el ***contador3*** que repetirá la acción hasta 50 veces para hacer el movimiento completo de abrir y cerrar las puertas, se utiliza para hacer un retraso en la visualización.

Para saber la dirección que tiene el ascensor, se realizará la comparación en todo momento, de la cabina con la del piso de destino con el objetivo de conocer si el movimiento que el ascensor debe realizar para atender la llamada.

PLC_PRG

Declaración de los Arrays

Declaración de los temporizadores y el tipo de temporizador utilizado (de puesta a ON)

Declaración del resto de variables utilizados a lo largo del programa

Declaración de las variables físicas del programa

```
1  PROGRAM PLC_PRG
2  VAR
3      boton_arriba: ARRAY [0..4] OF BOOL;
4      boton_abajo: ARRAY [0..4] OF BOOL;
5      etapas: ARRAY [0..5] OF BOOL;
6
7      //retardo tiempo entre planta//
8      TON_0: TON;
9      //retardo tiempo movimiento de puertas//
10     TON_2: TON;
11     etapa_actual: INT := 0;
12     //niveles planta//
13     i: INT;
14     //0 parado, 1 subiendo, 2 bajando//
15     dir: INT;
16     nivel: INT;//para el movimiento del ascensor en la visualización//
17     start: BOOL; //inicio de los temporizadores//
18     contador0: INT;//para el tiempo entre plantas//
19     run: BOOL := FALSE; //Boton start//
20     abrir_puertas: BOOL;
21     puerta_izq: INT;
22     contador3: INT;//para el tiempo del movimiento de puertas//
23     puertas_mov: BOOL;
24     puerta_der: INT;
25     etapa: INT;//planta llamada ascensor//
26     //guardamos la direccion previa donde iba el ascensor//
27     prev_dir: INT;
28
29     //variables fisicas motores
30
31     fis_motor_subir: BOOL;
32     fis_motor_bajar: BOOL;
33     fis_motor_abrir_puertas: BOOL;
34     fis_motor_cerrar_puertas: BOOL;
35 END_VAR
```

Ilustración 37. Ventana para la declaración de variables a utilizar por el programa

Para los pulsadores y las distintas etapas o plantas se han utilizado unos vectores (*ARRAY)* de valores tipo *BOOL,* llamados ***botón_arriba, boton_abajo y etapas.*** El utilizar vectores, ahorra tener que estar declarando una a una las variables, tal y como se ve en la Ilustración 37. En los vectores declarados, se indica el valor inicial desde el cual se inician, hasta el valor final donde llegarán.

Durante la ejecución del programa se pueden visualizar los parámetros actuales de cada variable como se muestra en la Ilustración 38. En los recuadros de color naranja, se ven los valores actuales de cada variable. Por ejemplo, la variable ***contador0*** tiene un valor de 0, tal y como se ve en la *línea 72* del programa.

Al comenzar a abrirse las puertas por primera vez, permitirá desde ese momento accionar los pulsadores interiores. Esto se realiza para impedir que la simulación carezca de sentido, ya que es físicamente imposible accionar un pulsador del interior de la cabina sin encontrarse nadie dentro.

Cada vez que el ascensor inicia un movimiento, activa una variable en función de si el movimiento es de subida o de bajada, con el objetivo de recordar, en el momento que se detenga, cuál fue su último movimiento.

Si durante el movimiento del ascensor, es llamado desde alguna planta intermedia a la planta de destino, este guardará en memoria el orden de llamada de las plantas, para una vez llegado al destino continuar con el orden de llamada.

```
70                      TON_0(IN[FALSE] := start[FALSE], PT[T#1s] := T#0.05S);
71                      IF contador0[0] = 100 THEN
72                          contador0[0] := 0;
73                          etapa_actual[2] := etapa_actual[2] - 1;
74                      END_IF
75                      IF boton_arriba[etapa_actual[2]][FALSE] = TRUE THEN
76                          etapa[137] := nivel[137];
77                          i[5] := etapa_actual[2];
78                      END_IF
79                  END_IF
80
81              END_WHILE
82              contador0[0] := 0 ;
83              contador2[0] := contador2[0] + 1;
84          END_WHILE
85          abrir_puertas[FALSE] := TRUE;
86          WHILE contador3[0] < 50 DO
87              puertas_mov[FALSE] := TRUE;
88              TON_2(IN[FALSE] := puertas_mov[FALSE], PT[T#50ms] := T#0.05S);
89              IF TON_2.Q[FALSE] THEN
90                  puerta_izq[0] := puerta_izq[0] - 1;
91                  puerta_der[0] := puerta_der[0] +1;
92                  puertas_mov[FALSE] := FALSE;
93                  TON_2(IN[FALSE] := puertas_mov[FALSE], PT[T#50ms] := T#0.05S);
94                  contador3[0] := contador3[0] + 1;
95              END_IF
96              TON_2(IN[FALSE] := puertas_mov[FALSE], PT[T#50ms] := T#0.05S);
97          END_WHILE
98          contador3[0] := 0;
99          WHILE contador3[0] < 50 DO
100             puertas_mov[FALSE] := TRUE;
101             TON_2(IN[FALSE] := puertas_mov[FALSE], PT[T#50ms] := T#0.05S);
102             IF TON_2.Q[FALSE] THEN
103                 puerta_izq[0] := puerta_izq[0] + 1;
104                 puerta_der[0] := puerta_der[0] - 1;
105                 puertas_mov[FALSE] := FALSE;
106                 TON_2(IN[FALSE] := puertas_mov[FALSE], PT[T#50ms] := T#0.05S);
107                 contador3[0] := contador3[0] + 1;
108             END_IF
```

Ilustración 38. Ventana entorno de programación del programa principal durante la ejecución de programa

Dentro del programa principal hay dos funciones, las cuales son la mayor parte del programa, una función es para el movimiento de subir y otra para bajar. Las únicas diferencias entre ellas, es el valor de la variable ***dir***, esta variable indica la dirección del ascensor. A continuación se puede ver el código de programa de la función de subida del ascensor en la Ilustración 39.

PLC_PRG

```
43  IF dir = 1 THEN //subiendo//
44      FOR i := (4) TO 0 BY -1 DO
45          IF boton_arriba[i]= TRUE THEN
46              etapa:= 537- i *200 ;
47              WHILE etapa_actual <> i DO
48                  WHILE nivel <> etapa DO
49                      start:= TRUE;
50                      TON_0(IN := start, PT := T#0.05S);
51                      IF TON_0.Q AND nivel > etapa THEN
52                          nivel:= nivel - 2;
53                          contador0:= contador0 + 1;
54                          start := FALSE;
55                          TON_0(IN:= start, PT:= T#0.05S);
56                          IF contador0 = 100 THEN
57                              contador0 := 0;
58                              etapa_actual:= etapa_actual +1;
59                          END_IF
60                          IF boton_arriba[etapa_actual]= TRUE THEN
61                              etapa := nivel;
62                              i:= etapa_actual;
63                          END_IF
64                      ELSIF TON_0.Q AND nivel < etapa THEN
65                          nivel := nivel + 2;
66                          contador0 := contador0 +1;
67                          start:=FALSE;
68                          TON_0(IN := start, PT:= T#0.05S);
69                          IF contador0 = 100 THEN
70                              contador0:= 0;
71                              etapa_actual:= etapa_actual - 1;
72                          END_IF
73                          IF boton_arriba[etapa_actual]= TRUE THEN
74                              etapa:= nivel;
75                              i:= etapa_actual;
76                          END_IF
77                      END_IF
78
79                  END_WHILE
80                  contador0:= 0 ;
81              END_WHILE
82              abrir_puertas:= TRUE;
83              WHILE contador3< 50 DO
84                  puertas_mov:= TRUE;
85                  TON_2(IN:= puertas_mov, PT:= T#0.05S);
86                  IF TON_2.Q THEN
87                      puerta_izq:= puerta_izq - 1;
88                      puerta_der:= puerta_der +1;
89                      puertas_mov:= FALSE;
90                      TON_2(IN:= puertas_mov, PT:= T#0.05S);
91                      contador3:= contador3+ 1;
92                  END_IF
93                  TON_2(IN:= puertas_mov, PT:= T#0.05S);
94              END_WHILE
95              contador3:= 0;
96              WHILE contador3< 50 DO
97                  puertas_mov := TRUE;
98                  TON_2(IN:= puertas_mov, PT:= T#0.05S);
99                  IF TON_2.Q THEN
100                     puerta_izq:= puerta_izq + 1;
101                     puerta_der:= puerta_der - 1;
102                     puertas_mov:= FALSE;
103                     TON_2(IN:= puertas_mov, PT:= T#0.05S);
104                     contador3:= contador3+ 1;
105                 END_IF
106             END_WHILE
107             contador3:= 0;
108             abrir_puertas:= FALSE;
109             boton_arriba[i]:= FALSE;
110             etapas[i]:= 0;
111         END_IF
112     END_FOR
113     prev_dir:= 1;
114     dir:= 0;
```

Bloque condicional WHILE

Esta activo mientras el nivel del ascensor no sea igual planta llamado

Bloque condicional IF

Para el bajar el ascensor

Bloque condicional IF

Para resetear el contador0

Bloque condicional IF

Para el subir el ascensor

Bloque condicional WHILE

Esta activo mientras el ***contador3*** sea menor del número de operaciones a realizar, en este caso 50

Ilustración 39. Función de subida del ascensor dentro del código del programa del ascensor

5.2. Visualización

Todo el código de programa creado está destinado a la visualización, esta se ha creado a través de formas rectangulares para las puertas, la fachada del edificio y las ventanas se han creado con imágenes. Como puede observarse en la Ilustración 40. Se tienen distintos escenarios, alzado del edificio y el interior de la cabina. Uno de ellos es el exterior del edificio, para ver el desplazamiento del ascensor entre plantas, y la otra de las vistas es la pared interior de la cabina, donde están las botoneras del ascensor.

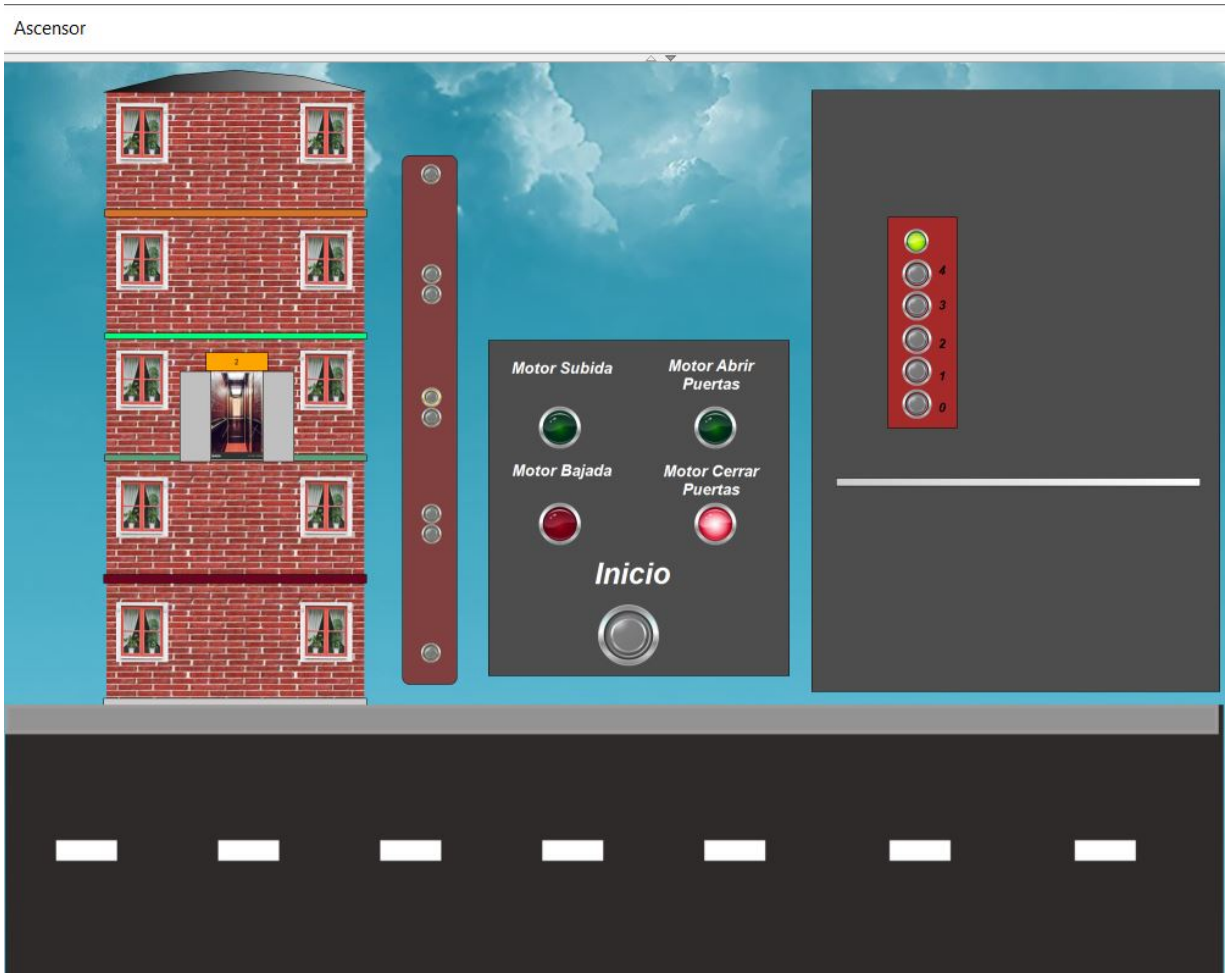

Ilustración 40. Ventana de visualización animación ascensor

5.3. Problemas encontrados

La mayoría de los problemas surgidos, han sido a la hora de coordinar los movimientos y colocar los objetos en la visualización, ya que se han tenido que estar encuadrando para su perfecta coordinación con el código, como es el caso de las puertas de la cabina y la altura de cada planta.

6. Ejercicio Control PID del motor de un tanque

En este ejercicio, se ha tratado en control del motor de llenado de un tanque, el cual está controlado a través de un PID. Este PID se le introducen los parámetros de configuración a través de las cajas, que se han colocado en la visualización para ello. También para que se haga una forma más visual se ha pegado un corte en el tanque para ver como sube el nivel del agua y va disminuyendo la potencia del motor. Como complemento extra se ha hecho la comunicación del programa con una Raspberry Pi, la cual simula tener un PLC conectado a nuestro programa.

Este ejercicio se ha resuelto mediante un programa compuesto de un POU:

- PLC_PRG: es el POU principal y único en el cual se ha programado en lenguaje FBD, Diagrama de Bloques Funcionales, y se compone de una línea de red. Básicamente es un módulo de la librería, se ha programado para el control de un motor de un tanque.

En el diseño se han tenido en cuenta variables que se utilizan en la visualización y en el conexionado físico. A continuación, se muestran las distintas variables empleadas en la Tabla 7.

Variables utilizadas	Descripción
Fis_sensor	Contacto físico del Sensor de medida de la altura del tanque
Fis_motor	Salida que se conecta al motor de llenado del tanque
Visu_set_point0	Variable utilizada en la visualización para indicar el valor de consigna deseado al cual se quiere tener el nivel del tanque
KP	Variable utilizada para asignar el valor de la ganancia proporcional de nuestro sistema
TN	Variable utilizada para asignar el valor del tiempo de ajuste integral de nuestro sistema
TV	Variable utilizada para asignar el valor del tiempo de ganancia derivativo de nuestro sistema

Tabla 7. Variables de programa utilizado en el ejercicio Control PID del motor de un tanque

6.1. Programa Principal

Para el programa principal, se ha utilizado un bloque de función PID, llamado ***Control_PID***, ya preestablecido en la librería ***Útil, 3.5.11.0***, la cual se ha tenido que agregar previamente al programa, como ya se explicó previamente. Se hace doble clic en el árbol de proyecto en el lado izquierdo, sobre *Administrador de bibliotecas* > *Agregar biblioteca* > *Application* > *Common* y se selecciona ***Útil.*** .

PLC_PRG

```
1  PROGRAM PLC_PRG
2  VAR
3          Control_PID: PID;
4
5      Fis_sensor: REAL; // nivel actual del tanque
6
7      visu_set_point0: REAL;//variable utilizada para indicar en la
8                            //visualización el valor de consigna deseado
9      KP: REAL; // Ganancia Proporcional
10     TN: REAL; // Tiempo de ajuste Integral
11     TV: REAL; // tiempo de ganancia Derivativo
12
13     marca_interna_rebosa: BOOL;
14     Fis_motor: BOOL; //contacto para la conexion al motor de llenado
15
16
17 END_VAR
```

Ilustración 41. Ventana de declaración de las variables de control del motor a través de un PID

Lo primero que se hace, es en la ventana superior declarar todas las variables a usar, tal y como se ve en la Ilustración 41, se tiene es un sensor, llamado ***Fis_sensor***, que es el que indica el valor del nivel del tanque, el valor de consigna llamado ***visu_set_point0***.

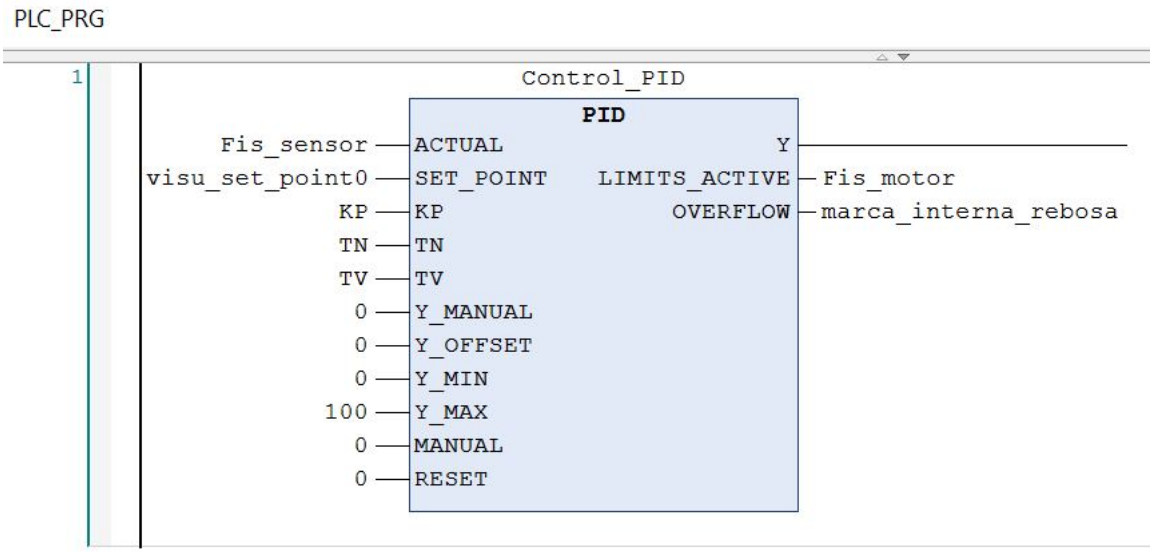

Ilustración 42. Ventana principal entorno de edición control PID

La ganancia proporcional llamada ***KP*** de tipo ***REAL***, el tiempo de ajuste integral ***TN*** y el tiempo de ganancia derivativo ***TN***. El mismo módulo PID, permite introducir los valores de salida manualmente. O en caso de tener algún offset colocar el valor. En las siguientes entradas, se indica cual van a ser los valores máximos y mínimos de la variable de control de salida. En este caso, es la potencia a la que funciona el motor en porcentaje de 0 a 100%. En la siguiente Ilustración 42 se ve el esquema de la función.

6.2. Visualización

Para la visualización se ha creado el entorno, utilizando imágenes personalizadas y barras de control que trae la biblioteca. Para poder simular, lo primero que se tiene que hacer, una vez arrancado el programa, es introducir los valores de ***KP, TN*** y ***TV.*** Los cuales se pueden introducir de dos formas, sobre los cuadros de texto en la misma ventana de visualización, o desde dentro del sistema en la ventana principal del programa principal. Cuando se encuentra ejecutando el programa, en el programa principal aparece una tabla en la cual podemos introducir los valores, forzando a que tengan esos valores con los comandos *CONTROL + F7*. La luz roja indica que el motor está funcionando y el valor del cuadro de texto, indica la potencia a la cual está trabajando el mismo. También se han utilizado controles deslizantes para fijar los valores de consigna o el valor actual del nivel del tanque. Aquí se puede ver el entorno gráfico en la Ilustración 43.

Ilustración 43. Ventana de visualización del Motor PID

6.3. Conexión a Raspberry Pi 3B

Para poder utilizar la Raspberry Pi, lo primero que se tiene que hacer es conectarla a la misma red en la que está conectado el ordenador. Se puede hacer por Wifi o por cable, en este caso se ha optado por hacerlo por cable, para garantizar que la conexión es estable. La visualización en el escritorio remoto se hace con la aplicación de Windows *escritorio remoto*. Aquí se puede ver en

Ilustración 44, el escritorio remoto de la Raspberry Pi y el programa ejecutándose en el navegador *Chromium*:

Ilustración 44. Conexión de la Raspberry Pi con el escritorio remoto

Al estar todo conectado en la misma red, te permite visualizar la visualización desde cualquier dispositivo, introduciendo la dirección: ***192.168.0.169:8080/tanque.htm*** la primera numeración es la dirección IP que ha asignado el rúter, lo siguiente es el puerto por el que se accede a la Raspberry Pi y por último el nombre del programa que se le ha asignado en la visualización, en este caso ***tanque***, tal y como se ve en la Ilustración 45.

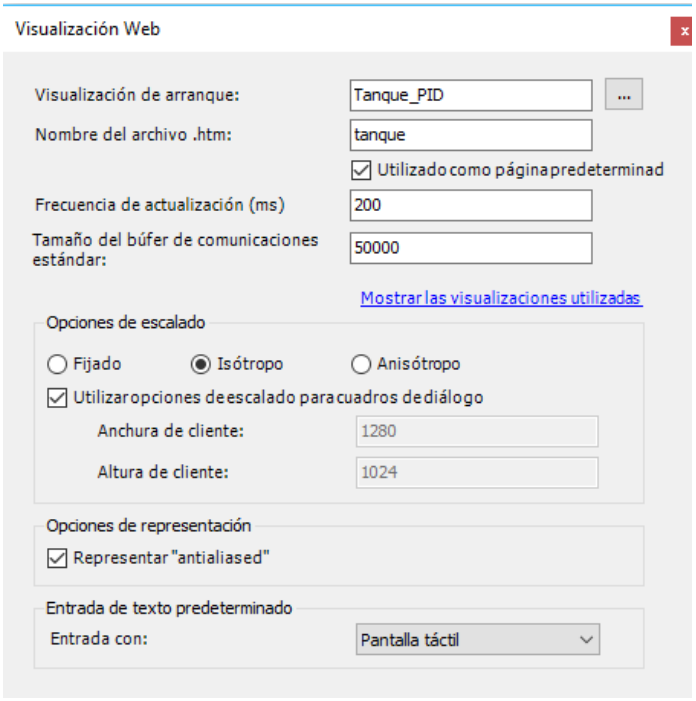

Ilustración 45. Ventana configuración visualización web Raspberry

Para la conexión remota desde el ordenador se ha utilizado la herramienta de Windows 10 ***Conexión a escritorio remoto.*** En *Equipo* se coloca la dirección asignada por el rúter, y una vez dentro el nombre de usuario es ***pi*** y la contraseña es ***raspberry*** todo en minúscula. Aquí podemos ver en la Ilustración 46, la ventana de conexión del escritorio remoto de Windows.

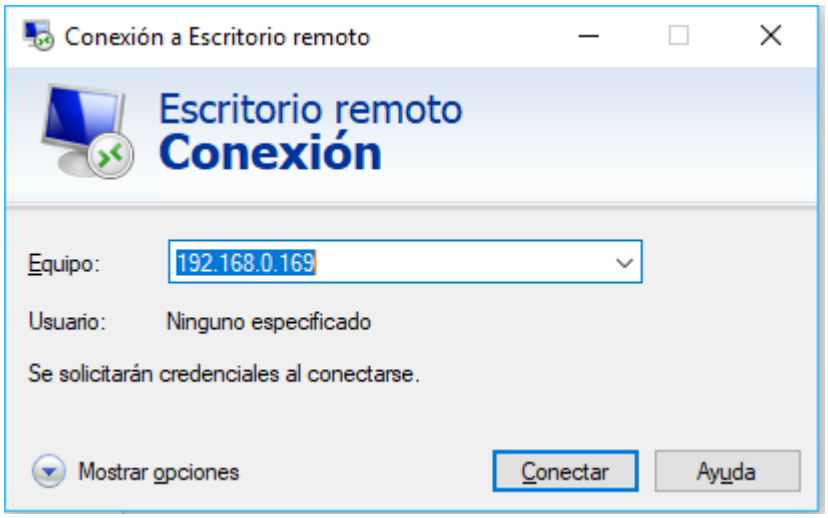

Ilustración 46. Ventana de programa *Escritorio remoto de Windows*

Una vez dentro del programa *Escritorio remoto de Windows*, se controla igual que si estuviera usando la pantalla del ordenador. También te permite la opción de si un dispositivo móvil está conectado a la misma red, introduciendo la dirección antes mencionada ***192.168.0.169:8080/tanque.htm***. Te lleva a la ventana de visualización y puedes interactuar con el programa de CODESYS.

7. Ejercicio Puerta de Garaje

La puerta de garaje su funcionamiento es el siguiente, tiene dos modos de funcionamiento. Manual, en el cual los finales de carrera son accionados por el operador, pulsando sobre ellos. Y el modo automático que una vez llega arriba tiene un temporizador que transcurrido un tiempo baja. Se ha colocado un pulsador de emergencia, por si en el momento que la puerta está bajando se encuentra algún obstáculo o problema poder rápidamente parar y volver a subirla.

Este ejercicio se ha resuelto utilizando mediante un programa compuesto de tres POUs:

- PLC_PRG: es el POU que se ejecuta cíclicamente y hace las llamadas al resto de POUs, es un lenguaje LD, Diagrama de Contactos.
- Visualiza: es un POU programado en lenguaje FBD, Diagrama de Bloques Funcionales, la cual se encarga de controlar los motores de la puerta de garaje en el modo automático.
- GVL: es una POU sin lenguaje de programación, donde se declaran las variables globales de ahí sus iniciales *Global Variable List.*

En el diseño se han tenido en cuenta variables que se utilizan físicamente y variables de programa para la simulación, en la Tabla 8 y Tabla 9, se tiene un resumen de las variables utilizadas:

Variables utilizadas Físicamente	Descripción
GVL.fis_ABRIR	Contacto físico para accionar la apertura de la puerta de garaje
GVL.fis_EMERGENCIA	Contacto físico del pulsador de emergencia
GVL.fis_FC_ARRIBA	Contacto físico del final de carrera superior
GVL.fis_FC_ABAJO	Contacto físico del final de carrera inferior
GVL.fis_MOT_SUBIR	Contacto del motor de subida de la puerta de garaje
GVL.fis_MOT_BAJAR	Contacto del motor de bajada de la puerta de garaje
GVL.fis_MAN_AUT	Contacto del selector de accionamiento manual o automático.

Tabla 8. Variables de programa para el conexionado físico de la puerta de garaje

Variables utilizadas para la visualización	Descripción
pulso	Variable utilizada como salida del generador de pulsos utilizado en la visualización para el movimiento de la puerta
Visu_contador	Contador utilizado para incrementar el valor de la variable de la puerta en la visualización
LOAD	Variable utilizada en caso de querer cargar algún valor en el contador anteriormente mencionado
generadorpulsos	Función que genera un pulso de valor preestablecido para el movimiento de subida y bajada de la puerta en la visualización
Visu_puerta	Variable para controlar la altura de la puerta en la visualización
Visu_salida	Variable de salida del contador utilizada internamente dentro del bloque visualiza para la altura de la puerta
Visu_subir	Contacto para indicar que la puerta de garaje esta subida
Visu_bajar	Contacto para indicar que la puerta de garaje esta bajada

Tabla 9. Variables de programa utilizadas para la visualización de la puerta de garaje

7.1. Programa principal

Lo primero que se ve es un bloque llamado ***Visualiza***, el cual hace una llamada al bloque de funciones, para así poder visualizar el sistema. Se tienen dos motores declarados de forma global para poder utilizarlos tanto en la visualización llamada *SCADA,* como en el programa principal. En la ventana principal se han declarado los temporizadores utilizados, tal y como se ve en la Ilustración 47.

PLC_PRG

```
1  PROGRAM PLC_PRG
2  VAR
3      TON1: TON;
4      VTON1: TIME;
5      TON3: TON;
6      VTON3: TIME;
7  END_VAR
```

Ilustración 47. Declaración de las variables del programa principal puerta garaje

Los motores de subida y bajada de la puerta, llamados ***GVL.fis_MOT_SUBIR*** y ***GVL.fis_MOT_BAJAR***, están puestos con marcas de SET y RESET, las cuales están indicadas en los contactos con una ***S*** y con una ***R***, tal y como se ve en la Ilustración 47. Hay dos finales de carrera uno superior y otro inferior, llamados ***GVL.fis_FC_ARRIBA*** y ***GVL.fis_FC_ABAJO***, en el modo manual, y en el modo automático se ha simulado a través del bloque de función **Visualiza**, con la variable ***visu_subir***, para el motor de subida, y otra variable llamada ***visu_bajar***, que para el motor de bajada.

El funcionamiento consiste en pulsar el botón de ***ABRIR,*** y esto acciona la puerta, accionando el motor de subida. Cuando llega arriba y toca el final de carrera, a través del temporizador ***TON1***, pasado el tiempo de 5 segundos, da la orden al motor de bajada. Si bajando se encuentra con un problema y se pulsa el botón de ***EMERGENCIA***, se paran los motores, y hasta que no se desactiva la emergencia, no es posible activar el motor de subida:

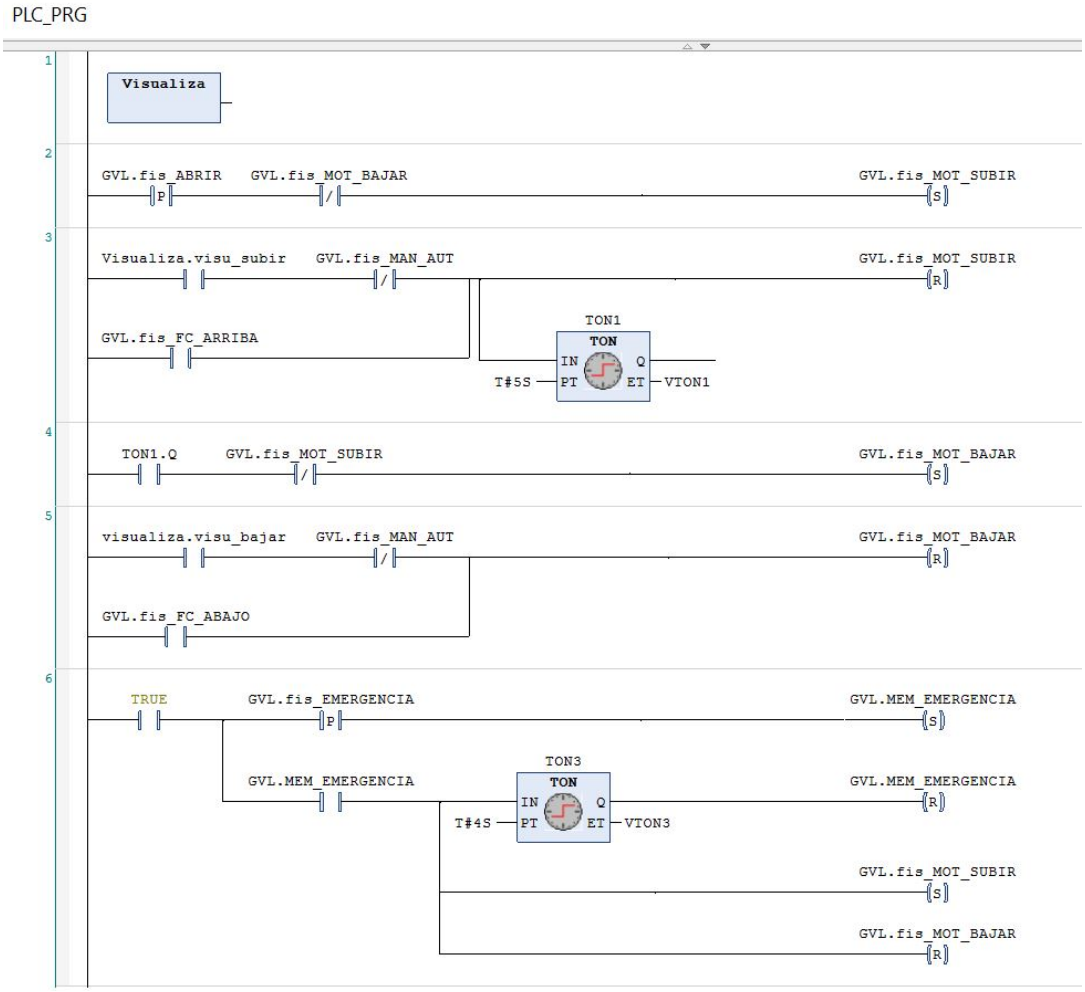

Ilustración 48. Diagrama de contactos del programa principal de la puerta garaje

Se ha utilizado otro temporizador llamado ***TON3,*** de 4 segundos para el reseteo de la marca interna utilizada para las emergencias, llamada ***GVL.MEM_EMERGENCIA***. En el modo manual, para que los motores se detengan, hay que pulsar físicamente los finales de carrera, colocados en la visualización, sino el motor no dejará de funcionar.

7.2. Diagrama de Bloques Funcionales Visualiza

Este bloque ha sido utilizado para el movimiento de la puerta en la visualización, se ha utilizado un diagrama de bloques funcionales que maneja el control de la puerta llamado ***Visualiza***. Este contiene un generador de pulsos, llamado ***generadorpulsos,*** que incrementa y decremento según la dirección de la puerta, si es subida o bajada. Para hacer esa subida o bajada se ha utilizado un contador, gracias a la variable de salida de ese generador de pulsos llamada ***pulso***. Va incrementando la variable del contador ***visu_contador*** llamada ***visu_puerta***.

La variable ***visu_puerta***, se introduce en un módulo llamado ***MUL,*** que multiplica esa variable por *-40*, este valor ha sido utilizado como un factor de escalado gráfico, siendo negativo porque la ordenada 0 está en la parte superior de la pantalla. Se tiene una nueva variable, llamada ***visu_salida***, la cual es la salida del módulo ***MUL.*** Se han utilizado bobinas también, para conectar las luces de cada motor en paralelo con el funcionamiento del motor. A continuación se puede ver en la Ilustración 49.

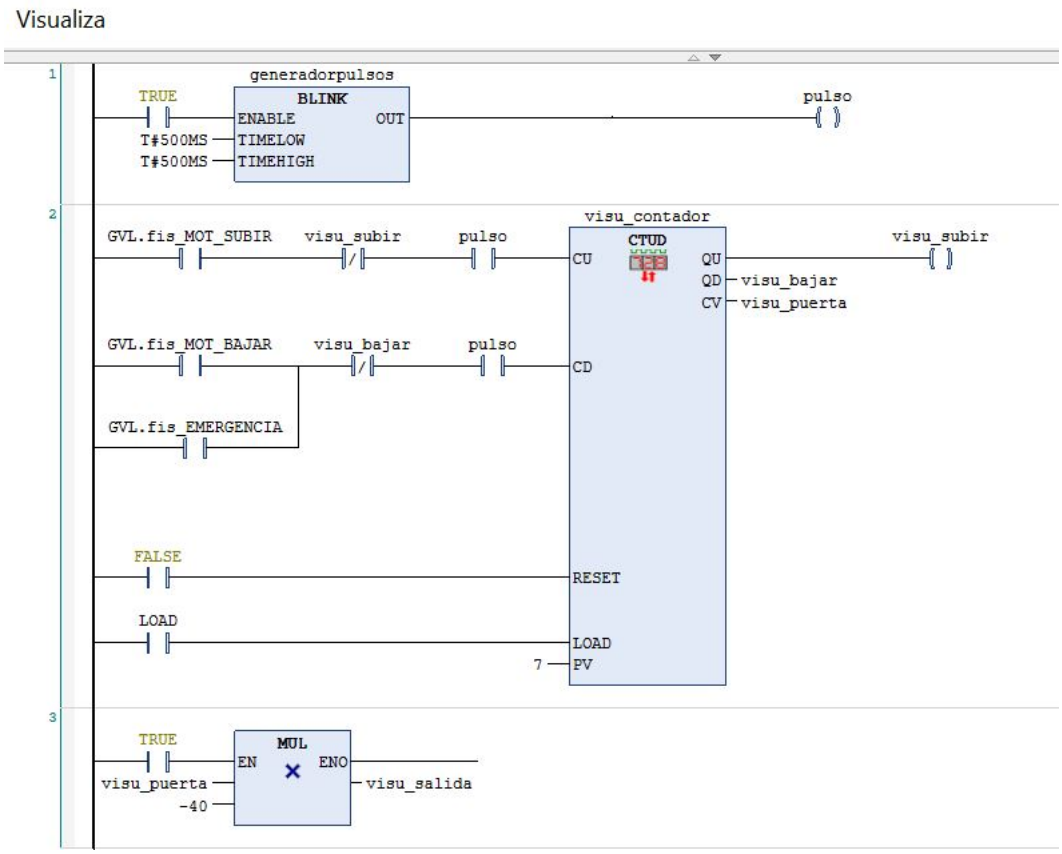

Ilustración 49. Bloque de funciones creado con diagrama de contactos y bloques

7.3. Visualización

Para la visualización, se ha utilizado imágenes con ***movimiento relativo***. Esto permite poder fijar la imagen de la puerta en el margen superior y solo desplazar el margen inferior con la variable, llamada ***visu_salida*** de tipo ***INT.*** Para los finales de carrera se ha utilizado unas imágenes de finales de carrera industriales. Las imágenes de los finales de carrera, se han habilitado dentro de su configuración, para cuando se pulsa directamente sobre la imagen, al hacer *clic* con el ratón. Las variables ***GVL.fis_FC_ARRIBA*** o ***GVL.fis_FC_ABAJO*** conmuten a 0 o 1.

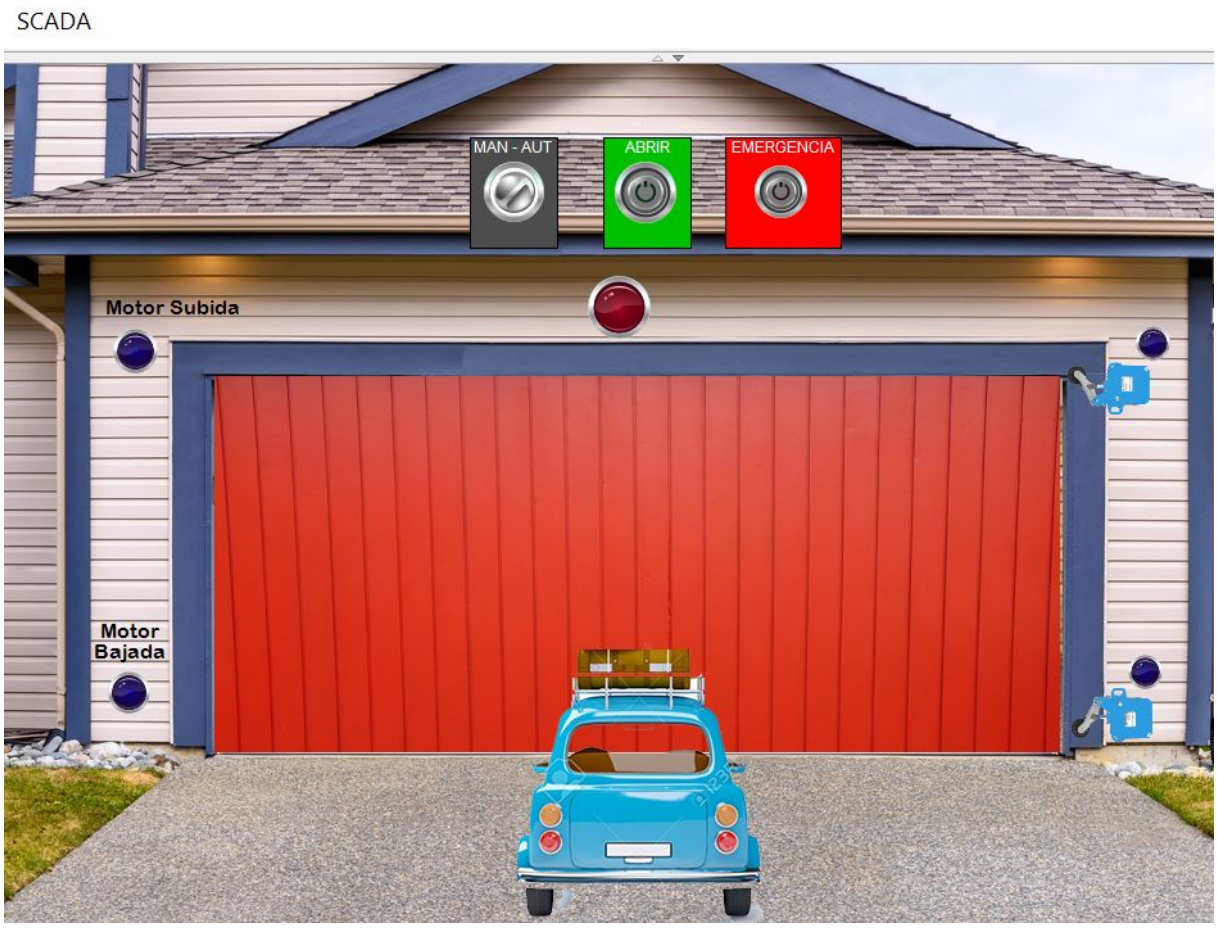

Ilustración 50. Ventana de visualización de la puerta de garaje

Se han colocado tres botones, llamados ***manual/automático***, ***START*** y ***EMERGENCIA,*** unas lámparas que indican cuando está funcionando cada motor, o si están o no activos los finales de carrera, todas están incluidas en la biblioteca de imágenes ya precargada. Tal y como se ve en la Ilustración 50.

8. Ejercicio Conversión Fahrenheit-Celsius analógico-digital

En este ejercicio se ha tratado el problema que puede haber a la hora de conectar dispositivos analógicos existentes en alguna instalación para su adaptación a un sistema digital. Estos dispositivos analógicos muchas veces no tienen los mismos rangos de trabajo que se utilizan en el programa que controla el dispositivo programado. En este caso, se estudiará el control de la temperatura. La sonda de temperatura analógica existente está en grados Fahrenheit, y se necesitan las medidas digitalmente en grados Celsius. Gracias a una función que está incluida en CODESYS se puede hacer posible la conversión.

Este ejercicio se ha resuelto programando todo en una única POU:

- PLC_PRG: Es el POU que ejecuta todo el programa. Este programa está escrito en lenguaje FBD, Diagrama de Bloques Funcionales, y se compone de 2 líneas de red.

En el diseño se han tenido en cuenta variables que se utilizan físicamente y variables de programa para la simulación. En la Tabla 10 y Tabla 11 se tiene un resumen de las variables utilizadas:

Variables utilizadas Físicamente	Descripción
Fis_TempF	Contacto de entrada de la sonda analógica de temperatura en grados Fahrenheit
Fis_TempC	Contacto de salida de temperatura en grados Celsius
Fis_fallo	Contacto de salida del indicador de fallo en la sonda analógica de temperatura

Tabla 10. Variables Físicas utilizadas en el convertidor analógico-digital

Variables utilizadas para la Visualización	Descripción
interna_TempFMin	Variable de entrada con el valor mínimo de la sonda de entrada en grados Fahrenheit
interna_TempFMax	Variable de entrada con el valor máximo de la sonda de entrada en grados Fahrenheit
interna_TempCMin	Variable de salida con el valor mínimo de salida en grados Celsius
interna_TempCMax	Variable de salida con el valor máximo de salida en grados Celsius
convertidor_unidades	Función para la conversión de unidades
visu_TempCINT	Variable de salida utilizada para la conversión a valores enteros
REAL_TO_UINT	Función que convierte los valores de tipo REAL en valores de tipo ENTERO

Tabla 11. Variables y marcas internas utilizadas en la conversión analógica-digital para la visualización

8.1. Programa Principal

Primero se indica el lenguaje de programación, en este caso *Diagrama de Bloques funcionales.* Una vez creado el programa, se declaran todas las variables a utilizar en la ventana superior. Para el programa principal se ha utilizado el modulo llamado ***convertidor_unidades,*** el cual se encuentra dentro de la librería ***Util.*** Este módulo convierte una señal, como en este caso de tipo *REAL*, llamada ***fis_TempF***, a otra con otro rango distinto, llamada ***fis_TempC.*** A las variables se le pueden precargar inicialmente un valor, como se puede ver en la Ilustración 51.

PLC_PRG

```
1  PROGRAM PLC_PRG
2  VAR
3      fis_TempF:REAL:=30;
4      interna_TempFMin:REAL:=32;
5      interna_TempFMax:REAL:=212;
6      interna_TempCMin:REAL:=0;
7      interna_TempCMax:REAL:=100;
8      fis_TempC:REAL;
9      fis_Fallo:BOOL;
10     convertidor_unidades: LIN_TRAFO;
11     visu_TempCINT:INT;
12 END_VAR
```

Ilustración 51. Declaración de variable del convertidor analógico-digital

La sonda de temperatura representada por la variable ***fis_TempF*** tiene sus rangos definidos por las variables de temperatura Fahrenheit mínimo y máximo, llamadas ***interna_TempFMin*** y ***interna_TempFMax*** respectivamente. Estos son los rangos por los que se puede mover desde 32 que sería la mínima, a 212 que sería la máxima. Esos valores se van a convertir en valores de tipo REAL, pero en este caso, con otro rango distinto de trabajo, que son las variables, llamadas ***interna_TempCMin*** y ***interna_TempCMax.*** Después para poder tener esos valores normalizados de tipo entero, se ha convertido la variable de tipo REAL en variable de tipo INT, con un módulo de la librería llamado ***REAL_TO_UINT.*** A continuación podemos ver las funciones en la Ilustración 52.

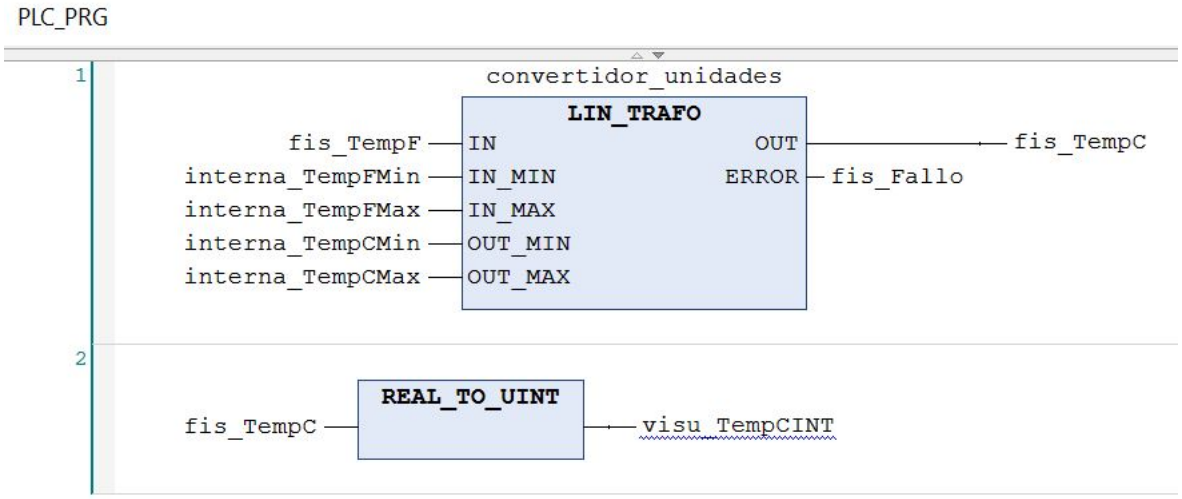

Ilustración 52. Diagrama de bloques del convertidor analógico-digital

8.2. Visualización

Para la visualización se han utilizado potenciómetros e indicadores disponibles en la librería de objetos. También se ha incluido una gráfica llamada ***Traza,*** la cual pinta gráficamente los valores que toman las variables***.*** Esta gráfica se encuentra dentro de la librería predefinida llamada ***Elementos especial de control***. Se ha incluido para poder monitorizar en tiempo real los valores de las variables. Esto ha permitido poder ver con precisión como van evolucionando y actualizándose dichos valores, tal y como se ve en la Ilustración 54.

Al grafico se le ha agregado el sensor de entrada analógico, la salida en grados Celsius y la conversión de esa salida a números enteros, representados por las variables ***fis_TempF***, ***fis_TempC*** y **visu_*TempCINT*** respectivamente***.*** Para la configuración del gráfico se tiene que indicar de que tarea se quieren coger los valores. En este caso ***MainTask,*** la cual es la tarea principal de programa. Para añadir tanto la entrada analógica como las salidas, se ha de especificar cuáles son las variables a mostrar en el gráfico. Para ello, hacemos clic derecho sobre el grafico y seleccionamos ***configurar traza.*** Una vez dentro en la casilla ***tarea,*** se selecciona ***MainTask*** y se añaden las variables las variables a controlar, tal y como se ve en la Ilustración 53.

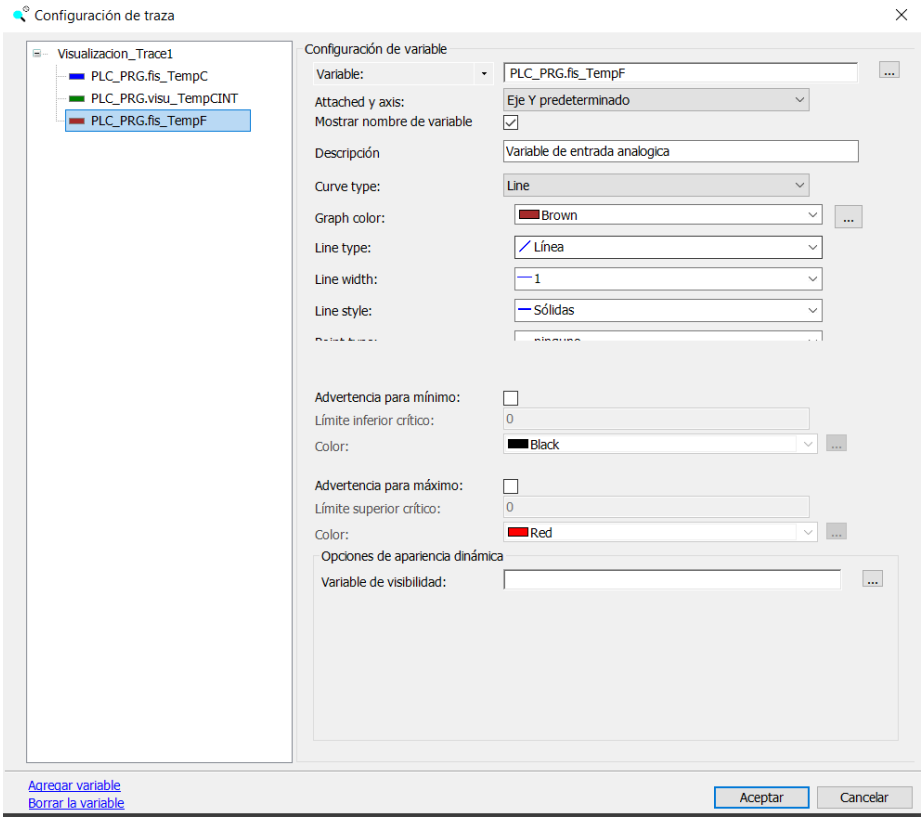

Ilustración 53. Configuración del grafico trazo para la visualización

Otro añadido en la visualización es la colocación de una luz de fallo, en caso de que el rango de temperaturas suministrado por el sensor de entrada este fuera del rango preestablecido inicialmente asociado a la variable ***fis_fallo***.

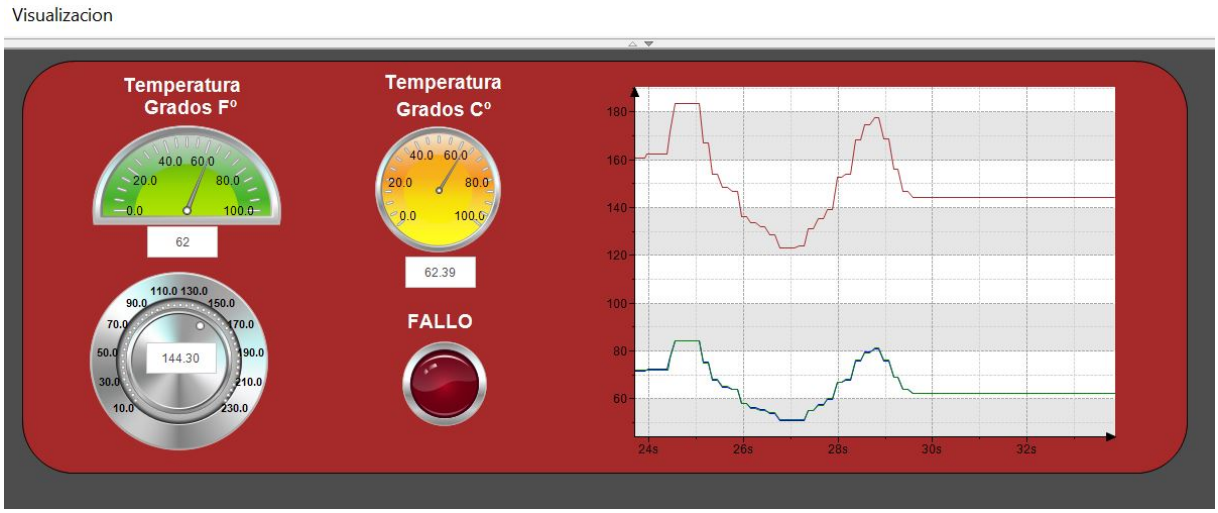

Ilustración 54. Visualización del convertidor Analógico-Digital

9. Conclusiones

Una vez profundizado en el programa CODESYS, se ha descubierto el abanico de posibilidades que ofrece, desde la creación de visualizaciones, el poder utilizar señales analógicas, o el control de un motor. Como teniendo unas nociones básicas sobre programación se pueden hacer grandes proyectos.

Es un programa intuitivo, el cual tiene una gran comunidad detrás. Lo cual hace más fácil encontrar soluciones a los problemas surgidos, en los ejercicios mostrados se ha intentado demostrar cómo se puede adaptar este programa a prácticas de asignaturas cursadas en el Grado de Electrónica.

El ver como el programa también muestra soporte para dispositivos, como la Raspberry Pi, elementos de bajo coste que pueden funcionar como un PLC, algo que lo hace atractivo para la enseñanza.

Para poder crear los mismos movimientos de los objetos, que hay en la realidad, se hace mucho más complejo en el programa CODESYS, debido a que para crear esos movimientos hay que utilizar muchas más funciones y líneas de código de las que se utilizarían en un ejercicio sin visualización. Todo esto ha hecho que ejercicios simples en los que con un par de líneas se habrían resuelto, se hayan tenido que utilizar hasta tres bloques de funciones distintas.

10. Problemas encontrados

- Uno de los problemas que se puede encontrar a la hora de compilar es que no sobrescribe la aplicación en la memoria, sino que es subida a continuación de la anterior. El programa tiene una limitación de memoria (25 MB), el cual no indica cuando está completa, sino que salta un error que dice así "*El índice (basado en cero) debe ser igual o mayor a cero y menor que el tamaño de la lista argumento*", como se puede ver en la Ilustración 54 y en la Ilustración 55. Se soluciona en la barra superior *Compilar* > *Limpiar todo* lo que hace esto último es liberar la memoria existente de cualquier programa precargado anteriormente:

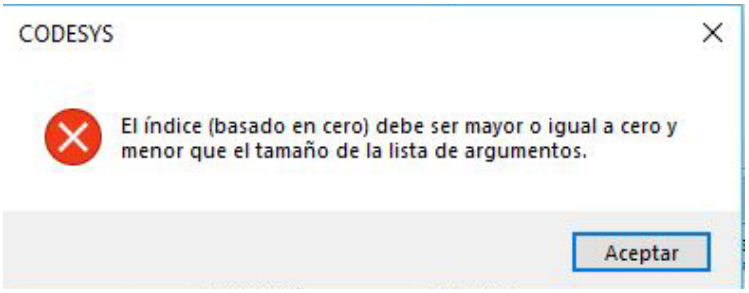

Ilustración 55. Ventana emergente compilación

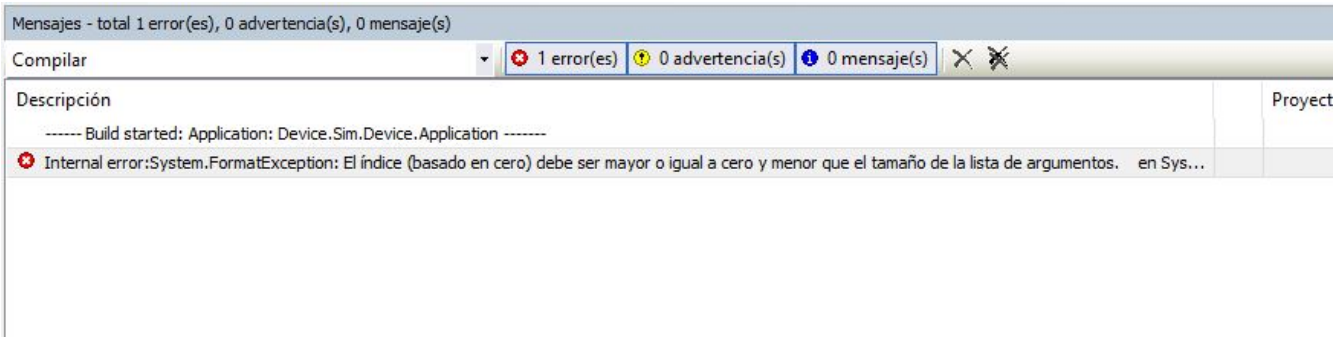

Ilustración 56. Error de compilación, debido a tener la memoria de programa llena

- Problemas a la hora de la conexión de la Raspberry Pi, en alguna ocasión no conecta a la primera y sale error de conexión, en la pestaña de la derecha, se puede observar que no está establecida. Para solucionar este problema se vuelve a escribir la dirección ip asignada por el rúter, en la misma caja de texto donde está y se pulsa *intro*.

- Otro de los problemas encontrados a la hora de programar, ha sido que un bloque de funciones, no es posible reutilizarlo para otras salidas. Con este lenguaje cada bloque es único, de ahí que se hayan creado dos bloques iguales para el movimiento en la visualización de cada una de las pilonas.
- El programa en ciertas funciones te obliga a conectar las variables a una bobina/contacto de salida, para poder simularlo como en el caso del contador de los bloques de funciones utilizado para la visualización.
- Para poder hacer la llamada a los bloques de funciones utilizados, siempre se ha de asociar ese bloque de función como puede ser por ejemplo ***visu_movimiento_pilona1*** está asociado dentro del programa principal a una variable llamada ***visu_p1.***
- A la hora de visualizar en el ejercicio del parking, con solo dos sensores simular la entrada o salida de un coche, se han tenido que crear cuatro funciones de flancos, dos para cada sensor, de subida como de bajada, ya que el programa no era capaz de simular solo con biestables.

Referencias

3S-Smart Software Solutions GmbH CODESYS. (2018, Noviembre). *www.codesys.com*. Retrieved 2018, from https://www.codesys.com/the-system.html

Anton Girod, A. (2017, Marzo). *Infoplc*. Retrieved 2018, from Infoplc: http://www.infoplc.net/descargas/42-codesys/2587-codesys-7-tips-empezar-programar

CODESYS. (2018, Diciembre). *Lista fabricantes CODESYS*. Retrieved from http://devices.codesys.com/device-directory.html

CODESYS. (2018, Diciembre). *Store CODESYS*. Retrieved from https://store.codesys.com/

DBR Automation, D. (2015). *dbrautomation*. Retrieved diciembre 2018, from http://www.dbrautomation.com/doc/pdf/Control/manual_programacion_PLC.pdf

IEEC UNED, I. (2017). *Controladores Lógicos Programables (PLCs), Departamento de Ingeniería Eléctrica, Electrónica y Control.* Retrieved 2018, from IEEC UNED: http://www.ieec.uned.es/investigacion/Dipseil/PAC/archivos/Informacion_de_referencia_ISE6_1_2.pdf

INCIBE CERT, I. (2018, Marzo). *INCIBE-CERT*. Retrieved Diciembre 2018, from https://www.incibe-cert.es/blog/automatizacion-coste

InfoPLC. (2018). *Foro InfoPLC .* www.infoplc.net.

INTRA Automation, I. (2017, marzo). *INTRA automation sl*. Retrieved 2018, from http://www.intraautomationsl.com/por-que-elegir-la-plataforma-de-programacion-codesys/

Larraioz. (2018, Diciembre). *Larraioz Elektronika*. Retrieved from https://larraioz.com/codesys

Opiron Satoshi, O. (2017). *Opiron*. Retrieved 2018, from https://www.opiron.com/2017/02/25/codesys-5-razones-para-aprender/

Universidad de León, U. d. (2017). *Introducción al software de programación Codesys, Laboratorio Remoto de Automática (LRA-ULE).* Retrieved 2018, from http://lra.unileon.es/sites/lra.unileon.es/files/Documents/plc/Moeller/Manual%20Software_Codesys.pdf

Velasco, R. (2014, febrero 6). *RedesZone*. Retrieved Octubre 2018, from RedesZone: https://www.redeszone.net/raspberry-pi/controla-tu-raspberry-pi-de-forma-remota-traves-del-protocolo-rdp/?utm_source=related_posts&utm_medium=manual