

IES POLITÉCNICO  
HERMENEGILDO LANZ  
GRANADA

DEPARTAMENTO DE ELECTRICIDAD

PROYECTO FINAL CFGS AUTOMATIZACIÓN Y ROBÓTICA INDUSTRIAL



MANEJO DE UN SISTEMA ROBÓTICO SIMULADO  
MEDIANTE UN SISTEMA DE CONTROL INALÁMBRICO

AUTOR:

JUAN ESTÉVEZ DELGADO

Granada, 7 de Junio de 2022

## Índice

Resumen del proyecto .....	4
1.Introducción y justificación .....	5
2. Objetivos .....	6
3. Antecedentes y estado de la cuestión.....	7
4. Desarrollo del proyecto .....	8
4.1 Descripción y especificaciones de la instalación .....	10
4.2 Valoración de soluciones propuestas y justificación de la solución adoptada .....	13
4.3 Selección y descripción de equipos y elementos de la instalación. ....	17
4.3.1 Hardware .....	17
4.3.2 Software.....	21
4.4 Programación del sistema .....	23
4.4.1 Consideraciones generales .....	23
4.4.2 Programación del sistema de control manual inalámbrico.....	32
4.4.3 Programación del sistema de control de recepción y transmisión de datos (ESP32) .....	34
4.4.4 Programación del sistema de control de recepción y transmisión de datos (Arduino UNO) .....	40
4.4.5 Programación del sistema en TIA Portal.....	46
4.4.6 WinCC Advanced.....	54
4.4.7 Servidor OPC (ABB IRC5 OPC Server).....	58
4.4.8 Programación en Robot Studio.....	60
5. Planos y esquemas .....	65
5.1 Esquema general de los sistemas involucrados .....	65
5.2 Dispositivo de control manual inalámbrico .....	66
5.3 Dispositivo de recepción y transmisión de datos .....	67
Manejo de un sistema robótico simulado mediante un sistema de control inalámbrico .....	2

6. Presupuesto.....	69
7. Trabajos futuros .....	70
7.1 Mejorar el protocolo J2WC.....	70
7.2 Mejorar el conexionado eléctrico de los componentes de hardware .....	70
7.3 Rediseñar la estructura de la estación del programa Robot Studio.....	71
8. Conclusiones .....	72
8.1 La importancia y relevancia de las comunicaciones .....	72
9. Bibliografía .....	74
Anexos .....	77
Anexo I. Programación del sistema de control manual inalámbrico.....	77
Anexo II. Programación del sistema de recepción y transmisión de datos (ESP32).....	80
Anexo III. Programación del sistema de recepción y transmisión de datos (Arduino UNO) .....	88
Anexo IV. Programación en TIA PORTAL.....	95
Anexo V. Programación del sistema robótico en Robot Studio .....	107

## Resumen del proyecto

Este proyecto consiste en realizar una implementación e integración de varios sistemas pertenecientes al ámbito de automatización y de la robótica industrial, con el fin de conseguir controlar un sistema robótico simulado mediante un sistema de control manual inalámbrico en el que hay integrado un dispositivo que cumple con el objetivo de proporcionar al usuario un medio para que pueda interactuar con el sistema y manejar el robot. Se trata de un controlador usado para conectarse a un mando de la videoconsola *Wii*.

Para ello, se ha buscado una conjunción hardware-software que permita alcanzar este objetivo de manera satisfactoria, cada componente que se ha utilizado es un eslabón imprescindible en el proceso, y la comunicación entre los mismos lo más importante.

En la solución propuesta intervienen elementos de control físicos e informáticos (pulsadores, joysticks, potenciómetros, botones de control y campos de entrada virtuales) tanto para controlar y parametrizar los movimientos del brazo robótico simulado como para poder intervenir en el proceso de las comunicaciones con el fin de modificar su comportamiento y funcionamiento. Se encuentran implementados también dispositivos que se encargan de visualizar el estado de la simulación del sistema (posición de los ejes del robot, velocidad de movimiento, ejes activos, estado de las comunicaciones entre los dispositivos y la latencia o retraso de la conexión entre los mismos)

Se procede a continuación a redactar el desarrollo completo del proyecto, que consistirá principalmente en mostrar los objetivos que se desean cumplir, las soluciones y recursos que se han creído convenientes implementar y el proceso que se ha seguido para poder conseguir los objetivos que se han propuesto, mostrando finalmente las conclusiones a las que se han llegado.

## 1.Introducción y justificación

Este es un documento en el que se presenta el desarrollo de un proyecto de fin de ciclo del módulo de formación profesional superior de Robótica y Automatización Industrial.

Se ha orientado el proyecto al uso de las comunicaciones, implementando estas en todos los dispositivos que intervienen en el funcionamiento del sistema. El objetivo es de darle protagonismo a las mismas debido a la importancia y relevancia que tienen hoy en día casi todos los ámbitos, centrándose en el ámbito de la robótica y de la automatización.

La elección de los componentes y dispositivos que forman parte de la integración se ha hecho con el propósito de utilizar recursos que estén al alcance y a la disponibilidad de los usuarios, refiriéndose al coste reducido de estos, y a la facilidad que presentan los mismos a la hora de su implementación.

Al estar la causa de la creación del proyecto estrechamente relacionada con el módulo de formación profesional, se ha buscado que el contenido y los conocimientos aprendidos en cada una de las asignaturas que se han cursado estén presentes en el mismo de una manera u otra, con la finalidad de mostrar la importancia que tiene el saber hacer uso de las herramientas y recursos que han aportado cada una de las materias de manera individual para elaborar un sistema en el que se haga uso de todas ellas de forma conjunta para cumplir los objetivos propuestos.

## 2. Objetivos

El objetivo principal será el de conseguir controlar un sistema robótico simulado en el programa *Robot Studio* mediante elementos físicos: un joystick, pulsadores, giroscopio y acelerómetro, todos ellos integrados en un mando de la videoconsola Wii (se hará referencia a él como *nunchuk* a partir de ahora). Para ello, a continuación, se detallan los hitos y objetivos secundarios que se proponen para conseguir llevar a cabo esta tarea.

- Conseguir leer los datos enviados por los sensores y dispositivos de control del *nunchuk* haciendo uso de una placa Arduino.
- Enviar los datos recibidos por el *nunchuk* a un PLC simulado para que este los procese.
- El PLC simulado deberá de comunicarse con el sistema robot, para así poder controlar su movimiento.
- Los dispositivos que conforman el control manual físico del robot, se comunicarán inalámbricamente con el sistema que recibe estos datos y los envía al PLC simulado.
- Se hará uso de una interfaz gráfica para que el usuario pueda visualizar tanto el estado del sistema robot (posición de sus ejes, velocidad de movimiento, estado de la comunicación, etc.), como el estado de la comunicación entre los sistemas.
- El control del sistema robot ha de ser sencillo, personalizable e intuitivo. El usuario tendrá la opción de configurar y parametrizar el control de los movimientos del brazo robótico libremente, bien mediante la interfaz o haciendo uso de la programación estructurada, que se hará de manera que sea fácilmente escalable y modificable.

Se buscará que todo lo descrito anteriormente pueda ser implementado por el usuario de una manera sencilla e intuitiva.

### 3. Antecedentes y estado de la cuestión

Se empezará hablando de las causas que han motivado el desarrollo del proyecto, finalizando con el estado y el panorama actual al que hace referencia este.

Actualmente existe un amplio abanico de opciones y posibilidades para implementar dispositivos de control tales como microcontroladores o PLC's en sistemas de automatización y control industrial. Sin embargo, al querer implementar estos en conjunto y si pertenecen a distintos fabricantes, existe la problemática de que en ocasiones es complicado realizar la comunicación entre los mismos ya que cada fabricante ofrece sus propias soluciones que en general aplican a sus propios sistemas, no permitiendo un estándar que facilite una integración global de todos ellos.

Por ello, en este proyecto se ha buscado ofrecer una solución a este problema haciendo uso de los recursos que ofrecen cada uno de los dispositivos de manera individual y de conocimientos personales referentes tanto al ámbito de la programación como al de las comunicaciones.

En lo que a la situación actual de la cuestión se refiere, se han buscado soluciones para dar solución a este problema. Al protocolo OPC, por ejemplo, acuden muchos fabricantes y marcas para realizar las comunicaciones debido a que su implementación es sencilla y permite transmitir datos de forma encriptada entre los dispositivos.

También se están haciendo trabajos e investigaciones para integrar el IoT (*Internet of Things*), que permite que los dispositivos puedan conectarse a internet con la finalidad de poder comunicarse y compartir datos a través de este medio que, como bien es sabido, es utilizado mundialmente para permitir la comunicación tanto de los humanos como de los dispositivos entre sí.

No obstante, queda todavía trabajo por hacer y mucha investigación que se debe realizar para permitir una estandarización global de un protocolo de comunicaciones que facilite el realizar proyectos de automatización que requieran de elementos de distintos fabricantes.

## 4. Desarrollo del proyecto

En este apartado del documento se realizará una identificación de todos los componentes que forman parte de la instalación, así como las funciones que desempeñan para poder lograr conjuntamente los objetivos descritos en los puntos anteriores.

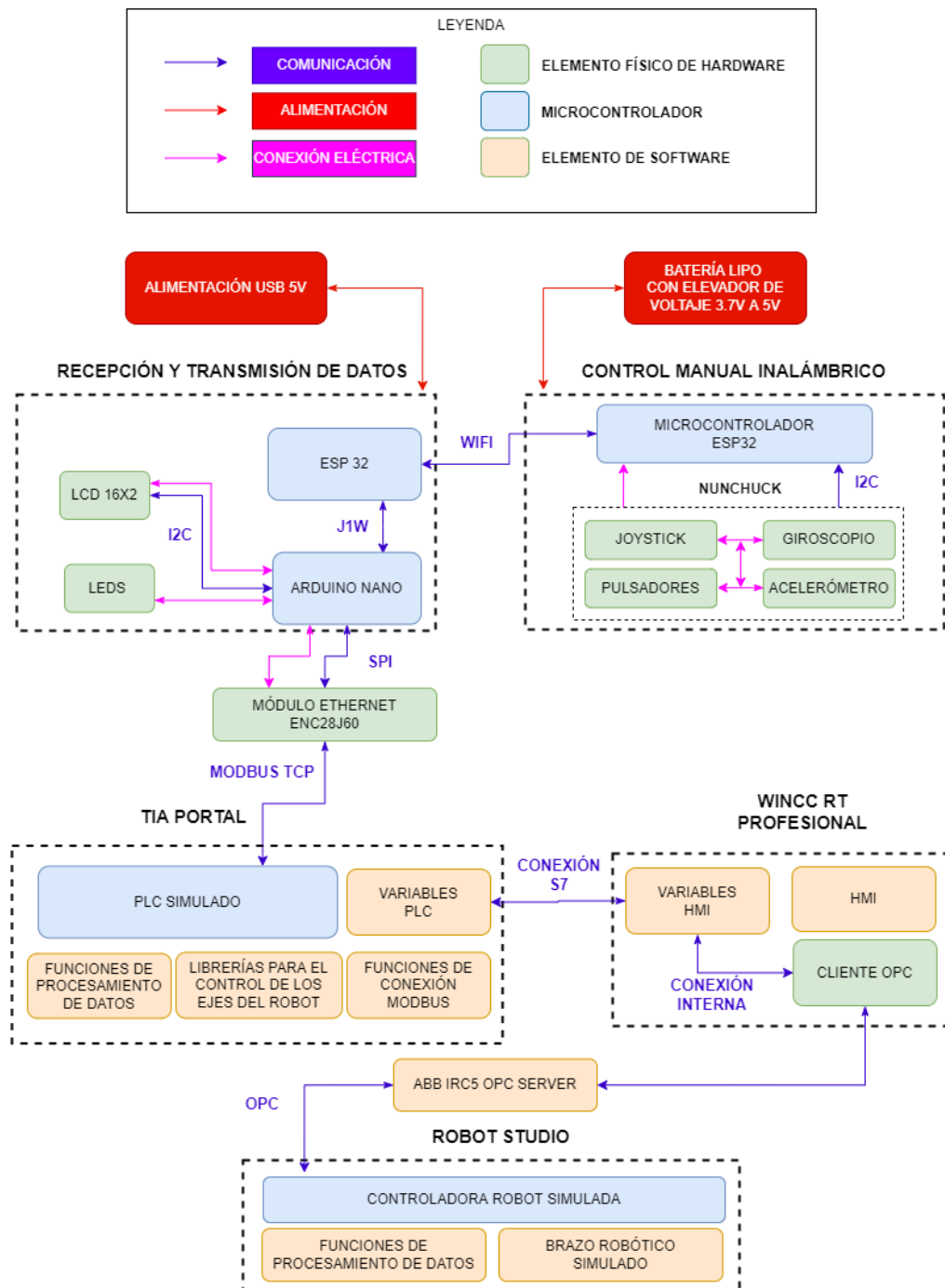
Para ello, se empezará describiendo brevemente las características técnicas, y la función que desempeñan los elementos que componen el sistema.

Antes de pasar a explicar cómo funciona cada sistema por separado, se hablará sobre algunas características y funcionalidades comunes a todos ellos.

Seguidamente, se hará una pequeña introducción teórica sobre los protocolos de comunicación que se han implementado, y para finalizar, se describirán todos los pasos, procedimientos, funciones y estrategias que se han seguido para poder realizar el proyecto.

A continuación, se muestra un diagrama que representa a todos los sistemas que formarán parte de la instalación, así como el conexionado entre los mismos





Esquema que muestra todos los sistemas que intervienen en el proceso y la relación entre los mismos

## 4.1 Descripción y especificaciones de la instalación

Los componentes que forman la instalación se agrupan siguiendo el criterio del papel que desempeñan de manera particular. Son los siguientes:

- **Sistema de control manual inalámbrico:** encargado de leer los datos enviados por los distintos periféricos de entrada y de transmitirlos inalámbricamente a los dispositivos receptores que procesan estos datos y los envían a través de un medio físico cableado al PLC simulado.
  - Utiliza el protocolo I2C para leer los datos del *nunchuk*.
  - Mediante WI-FI, envía los datos a la unidad receptora.
  - Utiliza una fuente de alimentación externa (batería LIPO) con un circuito de elevador de voltaje para conseguir los 5V requeridos para la alimentación.
- **Sistema de recepción y transmisión de datos:** recoge los datos recibidos de los sensores conectados al sistema de control manual inalámbrico, y los reenvía al PLC simulado.
  - Recibe los datos enviados por el control manual mediante WIFI, y los envía a la placa Arduino haciendo uso del JW2 (protocolo de diseño propio de comunicaciones).
  - Los datos recogidos se transmitirán mediante el protocolo SPI a un módulo ethernet ENC28J60, que conectará el Arduino con el PLC Simulado mediante el protocolo Modbus TCP.
  - Muestra también información del estado de la comunicación entre los distintos sistemas que intervienen en el proceso mediante Leds y una pantalla LCD:
    - Estado de la comunicación entre el *nunchuk* y el sistema de recogida de datos.
    - Estado de la comunicación Modbus: latencia y envío y recepción de datos realizados correctamente.
    - Tiempo de ciclo de la rutina de procesamiento de datos del Arduino.

- **PLCSIM Advanced**

Encargado de simular un PLC que ejecutará el programa. Habilitará un Switch virtual para permitir que el cable ethernet que se conecta al PLC pueda transmitir la información desde el sistema de recogida de datos hacia el PLC simulado, haciendo uso de una tarjeta de red virtual.

- **TIA Portal:** Software de Siemens para el control de sus PLCs.

Haciendo uso del PLCSIM Advanced, simula un PLC S71511, creando un switch virtual al que el módulo ethernet ENC28J60 pueda conectarse mediante el puerto ethernet físico del PC, para compartir la información recibida por los sensores.

Mediante rutinas, procesa los datos enviados por los sensores del control manual, para que puedan ser leídos e interpretados correctamente por el brazo robótico simulado.

Utilizando una librería de elaboración personal, asigna esos datos a cada uno de los seis ejes del robot para que puedan ser controlados de manera autónoma individualmente, además de poder visualizar el estado de los mismos mediante el panel de visualización de datos HMI (velocidad de movimiento, posición, estado de la ventosa neumática, etc.).

Comparte las variables de control antes mencionadas mediante una conexión interna que comparten el PLC con el sistema SCADA (WINCC RT Professional). Estos datos serán posteriormente enviados al sistema que simula el robot, haciendo uso de la comunicación OPC.

- **WINCC RT Professional**

Utilizado para implementar un sistema de visualización de datos y de configuración de los mismos, haciendo uso de una interfaz HMI en la que se representarán:

**Para visualización:**

- Los estados de los seis ejes del robot (posición, velocidad angular).
- Los ejes que están siendo controlados actualmente.
- El estado de la comunicación OPC y Modbus entre los sistemas.

**Para el control:**

- Velocidad angular máxima del movimiento.
- El tiempo de ciclo en el cual se actualizan las variables.
- Permitir una conexión bidireccional entre el sistema PLC-SCADA con el Software Robot Studio en el que se encuentra el brazo robótico simulado mediante un cliente OPC, en el cual se escribirán las variables de control de los ejes en el sistema Robot y serán leídas posteriormente por el servidor OPC.

- **ABB IRC5 OPC**

Software utilizado para crear el servidor OPC necesario en el que se compartirán las variables alojadas en la controladora virtual del robot simulado para que puedan ser manipuladas posteriormente por el cliente OPC que corre en el sistema SCADA (WinCC RT Professional).

- **Robot Studio**

Programa de ABB en el que se correrá la simulación del sistema robótico

En él, la controladora virtual leerá las variables OPC compartidas entre el servidor y el cliente OPC, para posteriormente utilizarlas en la manipulación de los ejes del robot mediante una rutina de procesamiento de datos.

Compartirá, también mediante OPC, el estado y la posición de los ejes del robot, para que puedan ser representadas en el HMI.

## 4.2 Valoración de soluciones propuestas y justificación de la solución adoptada

En este apartado se pretende aportar la razón y el motivo de utilización de cada uno de los componentes que conforman la instalación:

- **Recogida y lectura de datos de los periféricos y sensores externos**

Las placas Arduino facilitan la implementación de diversos sensores que son necesarios para que el usuario pueda interactuar con el proceso utilizando dispositivos físicos. Al ser de código abierto y al tener una gran comunidad de usuarios detrás, facilita la tarea de encontrar la información que se necesita para poder obtener información de cómo leer y procesar la información que transmiten dichos sensores, utilizando los protocolos de comunicación correspondientes.

- **Sistema compacto que contenga sensores capaces de controlar el sistema**

Para este proyecto se ha elegido implementar un *nunchuk* (controlador remoto de la consola Wii), que contiene diversos componentes que pueden ser aprovechados para el control del sistema: joystick, acelerómetro y giroscopio y dos pulsadores).

- **Dispositivo que permita una comunicación sencilla para transmitir información de manera inalámbrica desde el control manual hasta el sistema de recogida de datos**

Se ha elegido un microcontrolador ESP32, que cuenta con chips de Bluetooth y WIFI integrados. Mediante el uso de librerías y usando el protocolo de comunicaciones *ESP-NOW*, se consigue transmitir los datos entre los periféricos de entrada y el sistema de recogida de datos (ambos con este microcontrolador integrado).

- **Comunicación entre el dispositivo que recoge los datos enviados por el control manual (ESP32) y los transmite a la placa Arduino**

Se hace uso de un protocolo de comunicación de elaboración propia (JW2). Existe una amplia variedad de protocolos estandarizados de comunicación entre

microcontroladores, pero en este proyecto no se hará uso de ellos para comunicar estos dispositivos. Al ser un protocolo de elaboración propia, puede ser libremente configurado para adaptarse a las necesidades de la comunicación, además que, al hacer uso de él, se dota al proyecto de más personalidad, y permite que el autor del mismo pueda resolver una necesidad sin hacer uso de las soluciones propuestas por otros usuarios.

- **Sistema que reciba y transmita los datos enviados por los periféricos de entrada**

- Utilización de un ESP32, para la recogida de datos enviados por el sistema que envía los datos recogidos por los periféricos de entrada.
- Uso de una placa Arduino nano para recibir los datos enviados por el ESP32 mediante un protocolo de comunicaciones propio (JW2), descrito en los puntos posteriores, y transmitirlos al PLC simulado haciendo uso del módulo ethernet ENC28J60 a través del protocolo SPI.

- **Interfaz de comunicación entre la placa Arduino y el PLC simulado.**

Se ha hecho uso del módulo ethernet ECN28J60, que se comunica con el PLC simulado haciendo uso del protocolo Modbus TCP.

- **Elementos para la representación de datos y estado del sistema**

- Diodos LED conectados a la placa Arduino mediante salidas digitales y un LCD 16x2 conectado a la placa mediante I2C. Son dispositivos que no necesitan apenas cableado y que la programación necesaria para poder utilizarlos es relativamente sencilla.
- Adaptador conectado al LCD para comunicarse con el dispositivo a través del protocolo I2C.
- Utilización del WINCC RT Advanced para representar gráficamente el estado del robot y de las comunicaciones. Este ofrece la versatilidad de dar un formato personalizado a la interfaz, y el poder configurar y parametrizar los datos de manera sencilla.

- **Comunicación entre el dispositivo de recolección de datos y el PLC simulado**

- Se ofrece como solución la implementación del protocolo Modbus TCP. Es un protocolo estándar de comunicaciones, que permite transmitir datos de manera sencilla bidireccionalmente entre el emisor y el receptor.
- Se descartó la comunicación OPC entre estos sistemas por ciertas dificultades para implantar en la placa Arduino dicho protocolo.

- **Comunicación entre el PLC y el brazo robótico simulados**

- El protocolo OPC ha sido el elegido para satisfacer esta necesidad. El fabricante ABB cuenta con un programa que desempeña la función de servidor OPC (ABB IRC5 Server) que permite acceso a las variables de la controladora virtual del robot simulado para así modificar y leer sus valores.
- Otro de los motivos para elegir este protocolo, es que el WINCC RT Advanced cuenta con un cliente OPC integrado, lo que facilita en gran medida el poder compartir las variables de proceso entre ambos sistemas.

- **Software para realizar la simulación del brazo robótico**

- El fabricante ABB proporciona una herramienta llamada Robot Studio, con el que se pueden controlar sus robots física y virtualmente, además de permitir una versatilidad a la hora de ajustar, parametrizar y modificar muchos de los controles de los parámetros tanto del robot como de las comunicaciones. Por todo ello se ha elegido esta herramienta para aprovechar estos recursos que brinda el programa para poder realizar de forma satisfactoria el proyecto.

- **Herramienta para procesar los datos recibidos por los sensores de entrada y para realizar el control de los ejes del robot simulado**

Para llevar a cabo esta función, se ha utilizado el software TIA Portal. Es el programa que utiliza Siemens para programar sus PLC, que cuenta con una amplia variedad de herramientas para realizar todo tipo de funciones relacionadas con el sector de la automatización. En este proyecto en particular, se han hecho uso de los siguientes componentes:

- Funciones para implementar el protocolo Modbus TCP, que permiten leer y escribir datos de los registros de entrada y salida necesarios para comunicar todos los sistemas que forman parte del proyecto.
- Uso de la programación orientada a objetos, utilizada para crear seis instancias distintas de control para cada uno de los ejes, utilizando el mismo bloque fuente.
- Gestión de los posibles códigos de error que puedan manifestarse en la comunicación Modbus, para informar a los dispositivos pertinentes y que puedan tomar acciones al respecto.
- Permite también de manera sencilla compartir variables internas del PLC con el sistema SCADA.



## 4.3 Selección y descripción de equipos y elementos de la instalación.

En este apartado se describen las especificaciones físicas de cada uno de los elementos que conforman la instalación, centrándose principalmente en sus datos técnicos y en el papel que desempeñan.

### 4.3.1 Hardware

- **Microcontrolador ESP32**

La instalación contará con dos de estos dispositivos, uno para recoger los datos del nunchuk y otro para recibirlos y enviarlos a la placa Arduino

- Voltaje de alimentación: 5V.
- Voltaje de E/S: 3.3V.
- Corriente de funcionamiento: mínimo 500mA.
- SoC: ESP32-WROOM 32.
- Frecuencia de reloj: 80MHz / 240MHz.
- RAM: 512kb.
- Memoria flash externa: 4MB.
- Pines I/O: 34.
- Interfaces: SPI, I2C, I2S, CAN, UART.
- Protocolos Wifi: 802.11 b/g/n (802.11n hasta 150 Mbps).
- Frecuencia Wifi: 2.4 GHz - 2.5 GHz.
- Bluetooth: V4.2 - BLE y Bluetooth clásico.
- Dimensiones: 56x28x13mm.

- **Microcontrolador Arduino Uno**

Empleado para recibir los datos provenientes del ESP32 y enviarlos vía Modbus al PLC simulado.

- Voltaje de alimentación: 5V-12V.
- Voltaje de E/S: 5V.
- Consumo de corriente: 19 mA.
- SoC: ATmega328p.
- Frecuencia de reloj: 16 MHz.

- RAM: 2kb.
- Memoria flash externa: 32 kB.
- Pines I/O: 22.
- Interfaces: SPI, I2C.
- Dimensiones: 18 x 45 mm.

- **Nunchuk**

Controlador de la videoconsola Wii, empleado como periférico de entrada para el control de los elementos del sistema.

- Sensores:
  - Dos pulsadores.
  - Un joystick.
  - Acelerómetro.
- Circuitería para la lectura de los sensores.
- Circuitería para establecer una conexión mediante el protocolo de comunicaciones I2C con el dispositivo receptor.
- Cable de conexión con 6 hilos.

- **LCD 16x2**

Se conectará al microcontrolador ESP32 que recibe los datos del nunchuk para mostrar información del estado de la comunicación.

- Voltaje de alimentación: 4,7 – 5,2V.
- Pines I/O: 16.
- Número de caracteres que puede representar: 32 (distribuidos en una matriz de 16 columnas y dos filas).
- Consumo de corriente con led de retroiluminación: 125mA.
- Dimensiones: 85.0 x 29.5 mm.

- **Módulo I2C para LCD 16x2**

Adaptador conectado a los Pines I/O del LCD 16x2 para permitir una comunicación de la misma vía I2C con el microcontrolador ESP32.

- Voltaje de alimentación: 5V.
- Pines I/O: 20.

- 16 para conectarse al LCD.
- 4 para conectarse al microcontrolador vía I2C.
- Potenciómetro para regular el contraste de la pantalla.
- Dirección de interfaz I2C: 0x27.
- Consumo de corriente: 2mA.
- Dimensiones: 4.35 x 2.95 mm.

- **Leds y resistencias**

Se utilizarán para visualizar el estado de la comunicación entre los sistemas implementados y de las líneas de datos del protocolo J2W.

- 2 led rojo con una resistencia de 220  $\Omega$ .
- 1 led ámbar con una resistencia de 220  $\Omega$ .
- 1 led azul con una resistencia de 220  $\Omega$ .
- 1 led verde con una resistencia de 100  $\Omega$ .

- **Módulo ethernet ENC28J60**

Dispositivo conectado a la placa Arduino, utilizado como interfaz de comunicación entre este y el PLC simulado utilizando el protocolo Modbus.

- Voltaje de alimentación: 5V.
- Pines I/O: 10.
- Frecuencia de reloj: 20Mhz.
- Interfaz de comunicación: protocolo SPI.
- Dimensiones: 56.0 x34.0mm.

- **Batería LIPO**

Encargada de alimentar el sistema de control inalámbrico compuesto por el nunchuk y el ESP32.

- Voltaje nominal: 3,7V.
- Voltaje máximo de carga: 4.2V.
- Capacidad: 220mAh.
- Dimensiones: 33.5 x 97mm.

- **Módulo de elevador de voltaje (MT3608)**

Eleva el voltaje de la batería LIPO, entre 3.7 a 4.2, a 5V, necesarios para la alimentación del sistema de control inalámbrico.

- Voltaje de entrada: 2.0V a 24V.
- Voltaje de salida: 5.0V a 28V.
- Pines I/O: 4.
- Cuenta con potenciómetro para ajustar el voltaje de salida.
- Corriente máxima de salida: 2A.
- Velocidad de conmutación: 1.2MHz.
- Dimensiones: 37.0 x 17mm.

- **Cables para el conexionado**

- 1 cable USB para la alimentación del sistema de recogida de datos.
- 1 cable ethernet para realizar la conexión entre el módulo ethernet ENC28J60 y el PLC simulado.
- Cables macho-macho y macho-hembra de tipo *jumper* para realizar la conexión física entre los dispositivos conectados a los microcontroladores.

- **3 placas de prototipo**

En ellas estarán conectados eléctricamente y también situados los componentes forman parte del sistema. Al contar con puntos de conexión conectados en serie entre ellos, facilitan el conexionado de los dispositivos.

- Dimensiones: 165.0 x 55.0mm.

- **Ordenador PC**

En él se ejecutarán todos los programas que el proyecto necesita, y permitirá mediante sus puertos comunicarse con el resto de componentes.

- Modelo: Latitude 3410.
- CPU: 1.6 GHz Intel Core i5-10210U Quad-Core (10th Gen).
- Memoria RAM: 8GB.
- Memoria ROM: 256GB.
- Tamaño de pantalla: 14 pulgadas.
- Sistema operativo: Microsoft Windows 10 Pro.

### 4.3.2 Software

En este apartado se detallan los programas de los que se han hecho uso para realizar el proyecto.

#### 4.3.2.1 Programas de Siemens

- **TIA Portal V16**

Software empleado para programar los PLCs del fabricante. Proporciona un amplio abanico de herramientas y entornos que serán utilizados en este proyecto para cumplir con los objetivos descritos anteriormente:

- Permite realizar una programación orientada a objetos.
- Cuenta con estructuras que facilitan la organización de los programas.
- Facilita la comunicación entre los programas y dispositivos de su fabricante, a través de protocolos de comunicación propios.

- **PLCSIM Advanced 3.0**

- Permite la simulación del PLC que se utilizará en el proyecto.
- Crea un switch virtual mediante el cual se comunicará el PLC simulado con el dispositivo de recolección de datos vía Modbus.

- **WinCC RT Advanced**

- Sistema centralizado que integra dispositivos de visualización y control de datos (HMI), y permite el diseño y el desarrollo de sistemas SCADA.
- Cuenta con un cliente OPC, que se utilizara posteriormente para la transmisión de datos con el sistema robótico.

#### 4.3.2.2 Programas de ABB

- **Robot Studio v6.8**

Herramienta para crear simulaciones de sistemas robóticos.

- Cuenta con un lenguaje de programación (RAPID) con el que se pueden crear programas para controlar los diversos elementos que conforman la estación.
- Permite crear una controladora virtual con variables que pueden ser leídas y modificadas haciendo uso de un programa externo, como el que se describe a continuación:

- **ABB IRC5 OPC Server**

Herramienta empleada para enlazar la controladora virtual creada por el programa Robot Studio para poder acceder a sus variables mediante OPC.

- Permite configurar parámetros de la comunicación, tales como el tiempo de actualización de las variables compartidas a través del servidor OPC.

#### 4.3.2.3 Programas utilizados para la programación de los microcontroladores.

- **Platform IO**

Entorno de desarrollo utilizado para facilitar la programación de los microcontroladores del proyecto.

- Cuenta con un compilador avanzado que detecta errores de sintaxis y fallos en el uso de variables para un propósito distinto al que tienen.
- Integra también una herramienta de autocompletado, que facilita el desarrollo del programa.

- **Arduino IDE**

Software de código abierto creado por los desarrolladores de las placas Arduino.

- Permite programar gran multitud de microcontroladores, no necesariamente de su marca, ya que cuenta con un gran repositorio de placas de distintos fabricantes.
- Facilita la depuración y la corrección de errores del programa haciendo uso de un monitor que visualiza el estado de la comunicación Serial, permitiendo ver el valor de las variables internas y el estado de ejecución del programa.

## 4.4 Programación del sistema

### 4.4.1 Consideraciones generales

En este punto se detallan las funciones que son comunes a todos los dispositivos que forman la instalación. Más adelante se especificará cómo están implementadas cada una de ellas en cada uno de los dispositivos.

#### 4.4.1.1 Cálculo de los tiempos de latencia entre los dispositivos

La latencia es el retraso temporal que existe en la transmisión de datos entre dos o más dispositivos, desde que el emisor emite la información hasta que el receptor la recibe.

Esta instalación cuenta con varios sistemas comunicados entre sí mediante distintos protocolos de comunicación, por lo que se ha decidido implementar en todos los dispositivos una función que se encargue de calcular el tiempo que tarda en transmitirse la información entre ellos.

Para llevar a cabo esta tarea, se ha propuesto lo siguiente: utilizar contadores de vida y temporizadores.

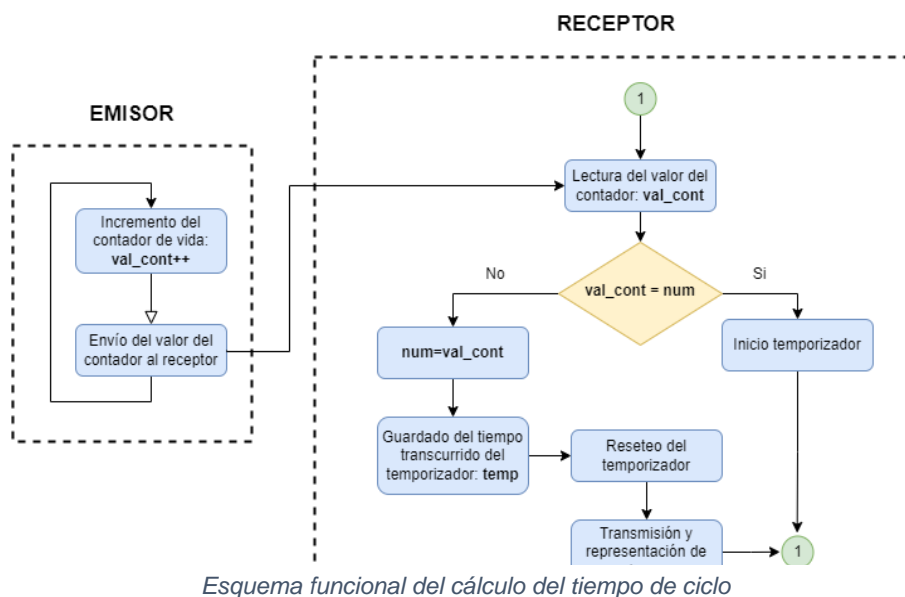
- **Contador de vida**

Variable de programa de tipo entero que se va incrementando secuencialmente a medida que este se ejecuta.

- **Temporizador**

Subrutina que se tiene el papel de calcular el tiempo transcurrido entre dos eventos.

A continuación, se muestra un diagrama de flujo que muestra cómo se realiza el cálculo del tiempo de latencia.



- **Dispositivo emisor**

Incrementa paródicamente el valor de su contador en cada ciclo, y lo envía este al dispositivo receptor cuando el bus de comunicaciones esté libre y pueda enviar dicho dato.

- **Dispositivo receptor**

Calcula el tiempo que transcurre desde que la variable que almacena el valor recibido por el contador del dispositivo emisor (**val\_cont**), sea igual a la variable auxiliar (**num**).

En el instante en el que sus valores sean distintos, implicará que el dato de entrada transmitido por el dispositivo emisor ha sido actualizado, en ese momento se guarda el valor actual del temporizador que representará el tiempo de latencia de la conexión. Dicho valor se representa visualmente y se comunica a los demás dispositivos. Se procede a resetear el temporizador y se vuelve al inicio del ciclo.



#### 4.4.1.2 Representación del estado de las comunicaciones

Para este proyecto se han utilizado diversos recursos para visualizar el estado de la comunicación entre los distintos dispositivos que intervienen en el proceso. El estado de la comunicación hace referencia a:

- Enlace correcto entre el dispositivo emisor y receptor: la comunicación ha sido establecida.
- La lectura de los datos de los sensores que realiza el dispositivo que se encarga de transmitirlos a los dispositivos receptores se está realizando de manera correcta.
- Representación del estado lógico de las líneas de datos físicas entre el dispositivo emisor y receptor.

Los elementos empleados para visualizar estos parámetros se detallarán en los siguientes apartados.

#### 4.4.1.3 Protocolos de comunicación implementados

Se detallan a continuación los protocolos de comunicación que se han utilizado para compartir información entre los dispositivos que conforman el sistema.

- **Protocolo I2C**

I2C es un puerto y protocolo de comunicación serial, define la trama de datos y las conexiones físicas para transferir bits entre 2 o más dispositivos digitales. El puerto incluye dos cables de comunicación, **SDA** y **SCL**.

- **SDA**: Línea por la que se transmiten los datos, las siglas hacen referencia a *Serial Data*.
- **SCL**: Línea que transmite una señal de reloj para permitir una correcta sincronización entre los elementos conectados. Es necesaria el uso de esta señal, ya que se trata de una comunicación síncrona.

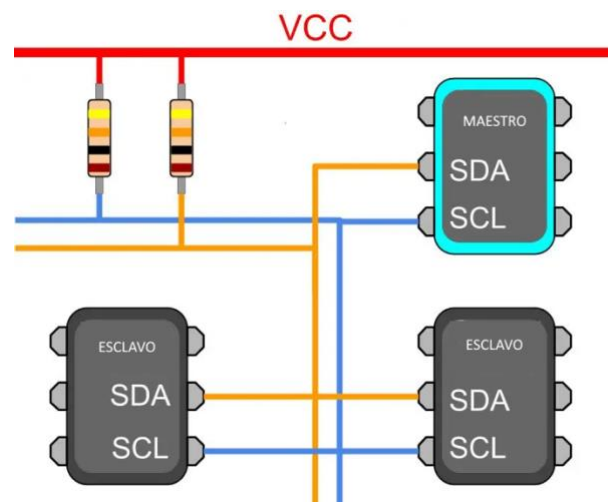
**En la comunicación intervienen:**

- **Maestro**

- Se encarga de controlar la línea de reloj (SCL).
- Responsable de iniciar y parar la comunicación.
- Envía los datos seriales por la línea de datos (SDA).

➤ **Esclavo**

- Lee la línea de reloj (SCL) para sincronizarse con el maestro.
- Cuenta con un identificador hexadecimal para poder ser identificado por el maestro (16#xx).
- En función de si recibe o transmite información, leerá de la línea de datos o actuará sobre ella cuando ésta esté disponible.



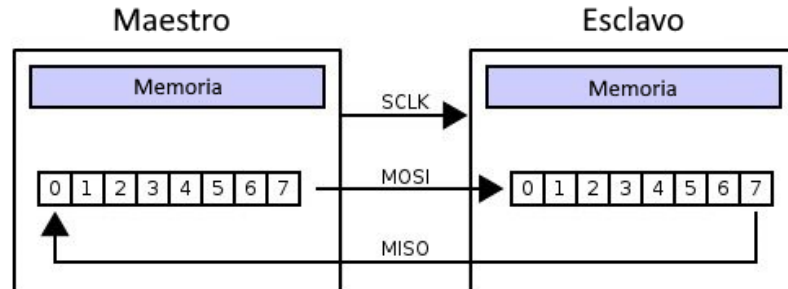
*Conexión entre el maestro y los esclavos en la comunicación I2C*

- **Protocolo SPI**

El SPI es un protocolo síncrono que trabaja en modo *full duplex* para recibir y transmitir información, permitiendo que dos dispositivos pueden comunicarse entre sí al mismo tiempo utilizando canales diferentes o líneas diferentes en el mismo cable. Al ser un protocolo síncrono el sistema cuenta con una línea adicional a la de datos encargada de llevar el proceso de sincronismo.

Existen cuatro líneas lógicas encargadas de realizar todo el proceso:

- **MOSI**: Línea utilizada para llevar los bits que provienen del maestro hacia el esclavo.
- **MISO**: Línea utilizada para llevar los bits que provienen del esclavo hacia el maestro.
- **CLK**: Línea proveniente del maestro encargada de enviar la señal de reloj para sincronizar los dispositivos.
- **SS**: Línea encargada de seleccionar y a su vez, habilitar un esclavo.



*Conexión entre el maestro y el esclavo en la comunicación SPI*

- **Protocolo ESPNOW**

Permite la comunicación entre los microcontroladores ESP32 utilizando la banda de 2.4GHz, sin necesidad de utilizar ninguna conexión Wifi, permitiendo así que la comunicación sea más rápida.

- Tamaño máximo de paquetes enviados por ciclo: 250 bytes.
- Permite una encriptación de la conexión de punto a punto.
- Utiliza las direcciones MAC de los dispositivos para establecer la comunicación.



*Conexión entre los microcontroladores ESP32 mediante ESPNOW*

- **Protocolo Modbus TCP/IP**

Modbus es un protocolo de solicitud-respuesta que se implementa utilizando una relación maestro-esclavo. En una relación maestro-esclavo, la comunicación siempre se produce en pares, un dispositivo debe iniciar una solicitud y luego esperar una respuesta y el dispositivo de inicio (el maestro) es responsable de iniciar cada interacción. El contenido de estas solicitudes y respuestas, y las capas de la red a través de las cuales se envían estos mensajes, son definidas por las diferentes capas del protocolo.

### **Funciones para enviar y recibir datos**

Haciendo uso de diversas funciones implementadas dentro del protocolo, se pueden leer y escribir datos de distinto tipo tanto en el maestro como en el esclavo, mediante los registros de registros de entrada y salida.

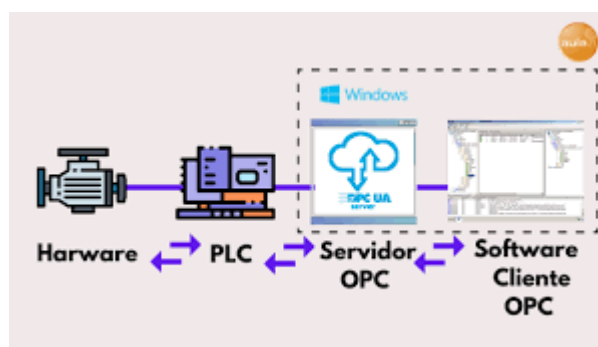
Función	Código	Descripción
Leer bobinas	1	Leer estado de salidas discretas (en general el estado de los relés).
Leer entradas discretas	2	Leer estado de entradas discretas.
Leer registro fijo	3	Leer valores de registros "Holding".
Leer registro de entrada	4	Leer valores de registros de entrada de solo lectura.
Escribir en una bobina	5	Permite modificar el valor de una sola salida discreta.
Escribir en un registro	6	Escribe un valor en un registro.
Escribir en varias bobinas	15	Permite modificar el valor de múltiples salidas discretas al mismo tiempo.
Escribir en varios registros	16	Escribe múltiples registros al mismo tiempo.
Leer/escribir en varios registros	23	Escribe y lee múltiples registros al mismo tiempo.

*Funciones empleadas en la comunicación Modbus TCP/IP*

- **Protocolo OPC**

Es un estándar de comunicación en el campo del control y supervisión de procesos industriales, basado en una tecnología Microsoft, que ofrece una interfaz común para comunicación que permite que componentes de software individuales interactúen y compartan datos. La comunicación OPC se realiza a través de una arquitectura Cliente-servidor. El servidor OPC es la fuente de datos (como un dispositivo hardware a nivel de planta) y cualquier aplicación basada en OPC puede acceder a dicho servidor para leer/escribir cualquier variable que ofrezca el servidor. Es una solución abierta y flexible al clásico problema de los drivers propietarios.

Es un estándar abierto, por lo que facilita la comunicación entre dispositivos de distintos fabricantes.



*Esquema de conexión entre los elementos del sistema de comunicación OPC*

- **Protocolo J2WC (Juan 2 Wire Communication)**

Es un protocolo de elaboración personal, diseñado para comunicar dos microcontroladores mediante un bus de dos hilos. Se ha creado con el objetivo de poder personalizar y configurar libremente la comunicación entre los microcontroladores pudiendo elegir los datos que se envían, así como la forma que tienen de hacerlo. A continuación, se describen sus características y funcionalidades principales, su implementación específica en el dispositivo emisor y receptor de datos se detallarán más a fondo en los siguientes apartados.

➤ **Características principales:**

- Protocolo half-duplex: permite una comunicación entre el emisor y el receptor bidireccional no simultánea.
- Comunicación de bus a 2 hilos:  
Se utilizará uno de ellos para enviar la señal de reloj necesaria para la sincronización, y el otro para enviar los bits.
- Configurable para enviar un número determinado de paquetes con un tamaño de 11 bits.
- Velocidad de transmisión de datos ajustable mediante un potenciómetro.
- Cálculo del tiempo que transcurre entre que el emisor envía y el receptor recibe los datos correspondientes (latencia).
- Cuenta con funciones internas que se encargan de procesar los datos tanto de los sensores enviados como de los bits de comunicación recibidos para procesarlos de manera adecuada y asociarlos a las variables correspondientes.

➤ **Estructura de la trama de datos comunicada.**

**Compuesta por:**

▪ **Bits identificadores de dato (0-2):**

Asignarán a cada dato enviado un identificador para facilitar el procesamiento posterior de la información. Permitirán la identificación de 8 dispositivos ( $2^3$  combinaciones).

▪ **Bit de signo (3):**

Si el número a enviar es negativo, este bit tomará el valor 1, de lo contrario, será 0.

▪ **Bits que representan el número entero a enviar (4-10):**

Estos bits representarán el estado de la parte entera del número a enviar, que tendrá un rango de 0 a  $127(2^7 - 1)$ .

TRAMA DE DATOS											
Nº bit	0	1	2	3	4	5	6	7	8	9	10
Descripción	Identificador (ID)			Bit de signo	Bits del número entero						

*Estructura de la trama de datos enviada por el protocolo J2WC*

## 4.4.2 Programación del sistema de control manual inalámbrico

### 4.4.2.1 Dispositivos que se incluyen en la instalación

- Nunchuk (controlador de la videoconsola Wii).
- Batería LIPO.
- Módulo de elevador de voltaje (MT3608).
- Microcontrolador ESP32.
- 1 led rojo con su correspondiente resistencia.
- 1 led verde con su correspondiente resistencia.
- 1 placa de prototipo.
- Cableado mediante cable tipo *jumper*.

### 4.4.2.2 Protocolos de comunicación utilizados

- I2C.
- ESPNOW.

### 4.4.2.3 Programación del sistema

- **Lectura de los datos enviados por el *nunchuk*:**

Mediante el protocolo de comunicaciones I2C, leerá los datos enviados por este, realizando el siguiente procedimiento.

Se declararán en el programa las librerías correspondientes para permitir la comunicación con el dispositivo.

En la función *setup()* de inicialización del programa, se establecerá lo que comúnmente se denomina *handshake*. Es una función que prepara al *nunchuk* para enviar los datos, y que el ESP32 recibe confirmación del mismo para indicarle que está disponible para enviarlos.

En la ejecución del resto del programa, se ira solicitando la información de los sensores del nunchuk mediante una función propia de la librería. Dicha función, al terminar de recopilar los datos, devuelve un 1 lógico, en ese momento se almacenarán los datos de los sensores recibidos en las variables internas del programa para que luego sus valores puedan ser leídos por las funciones de comunicación.



- **Transmisión de los datos recogidos por el nunchuk al ESP32 encargado de recibir los datos:**

Mediante el protocolo de comunicaciones ESPNOW, se establecerá una conexión con el ESP32 receptor, para enviar tanto los datos recogidos por los sensores del nunchuk como un contador de vida para el cálculo de latencia posterior.

Se declararán en el programa las librerías correspondientes para permitir la comunicación con el ESP32 receptor.

Se crea una estructura compuesta por los datos que se van a enviar, es en esta estructura donde se almacenarán los datos de los sensores recibidos por el *nunchuk*, y la rutina que envía los datos al ESP32 receptor enviará los datos que hay alojados en dicha estructura.

Se declaran en las variables la dirección MAC del ESP32 receptor y se realiza un setup inicial de la comunicación mediante una función de la librería.

Cada vez que se lean los datos del nunchuk, se guardarán los datos en la estructura ya mencionada y se llamará a la función que enviará los datos al ESP32.

- **Visualización del estado de la comunicación:**

Se harán uso de dos leds para visualizar el estado de la comunicación. Para ello se declaran los pines a los que están conectados como salida antes de escribir datos en los mismos.

Para controlar el estado de los leds, se utilizarán los valores que devuelven las funciones que controlan tanto la comunicación con el *nunchuk* del ESP32 emisor como la que comunica con el ESP32 receptor. Los valores que devuelven dichas funciones indican que la comunicación y el estado de lectura se han llevado a cabo satisfactoriamente.

➤ **Led Rojo**

- Apagado: el sistema no se encuentra alimentado eléctricamente.
- Encendido: el sistema está alimentado, pero no hay conexión con el nunchuk.
- Parpadeando: La comunicación con el nunchuk ha sido establecida y se están leyendo correctamente los datos de los sensores.

➤ Led Verde

- Apagado: no se ha establecido la conexión con el ESP receptor.
- Encendido: conexión establecida con el ESP receptor.

### 4.4.3 Programación del sistema de control de recepción y transmisión de datos (ESP32)

#### 4.4.3.1 Dispositivos que se incluyen en la instalación

- Microcontrolador ESP32.
- 1 led rojo con su correspondiente resistencia.
- 1 led azul con su correspondiente resistencia.
- Pantalla LCD 16x2.
- 1 potenciómetro de 10k.
- Una placa de prototipo.
- Cableado mediante cable tipo *jumper*.

#### 4.4.3.2 Protocolos de comunicación utilizados

- I2C.
- J2WC.

#### 4.4.3.3 Programación del sistema

- **Lectura de los datos recibidos por el ESP32 encargado de leer los datos del nunchuk:**

Se declararán en el programa las librerías correspondientes para permitir la comunicación con el ESP32 emisor.

Se crea una estructura compuesta por los datos que se van a recibir, es en esta estructura donde se almacenarán los datos de los sensores recibidos por el ESP32 emisor. La rutina que envía los datos a la placa Arduino mediante el protocolo J2WC leerá de la misma algunas de las variables que se transmitirán,

incluyendo además variables de estado de latencia de conexiones de la comunicación

Se realiza un *setup* inicial de la comunicación mediante una función de la librería. Cada vez que se lean los datos del ESP32 emisor (que se guardarán en la estructura antes mencionada), se accederá a los datos de la misma para almacenar sus valores en variables internas de programa que serán necesarias, para que las otras funciones del programa las utilicen, tal y como se describe en los siguientes apartados.

- **Implementación del protocolo J2WC para transmitir los datos a la placa Arduino:**

**Se procesan los datos recibidos**

Mediante una rutina que tiene la función de formar una trama de 11 bits para poder ser transmitida posteriormente por el bus de comunicaciones. Dicha rutina, tiene la siguiente estructura:

- Como entradas recibe:
  - El valor del número entero recibido
  - El id correspondiente
  - El vector en el que almacenar los 11 bits para poder asignarle los valores correspondientes.
- Realiza el siguiente procesamiento:

**Para formar los últimos 7 bits de la trama:**

- Si el valor de entrada supera el valor numérico 127, se iguala a 127.
- Mediante un bucle *for* va asignando valores booleanos a cada una de las posiciones del vector de salida (bits 4 a 10) dividiendo el número entero entre las potencias de 2 pertinentes.

**Para formar el bit de signo**

- El estado del bit de signo se corresponde con el bit nº3, por lo que se realiza una comparación lógica de si el número entero es mayor o menor que 0.

**Para formar el identificador**

- Recibe como entrada el dato de la función asociado con el identificador y lo procesa de igual manera que con el número entero, asignándole los valores pertinentes a los bits (0 al 2) de la trama

**Los datos se envían haciendo uso del protocolo J2WC con el siguiente procedimiento:**

- Se utiliza una estructura de tipo *case* que tiene asociada una variable: *stepindex* que se va incrementando secuencialmente según se terminan de realizar las tareas que se incluyen en cada uno de los pasos.
  - El programa cuenta con una función de control de la señal de reloj, que recibe como entrada un permiso de activación de tipo booleano y un tiempo de oscilación. La función devuelve una salida de tipo booleana, que se corresponde con la señal de salida del reloj.
  - Si el permiso de activación es 0, la salida se desactiva.
- Una vez realizados los pasos anteriores, se inicia la comunicación mediante el bus de comunicaciones, siguiendo el siguiente protocolo:
1. Se leen los datos de la estructura en la que se almacenan los datos recibidos por el ESP32 encargado de leer los sensores del nunchuk.
  2. Se llaman a las funciones que se encargan de procesar los datos recibidos, asignándoles las entradas correspondientes al estado de los sensores.
  3. Se guarda el valor del tiempo actual desde que se ha iniciado la comunicación en una variable, para posteriormente poder calcular la latencia de la comunicación.
  4. Se procede a realizar la llamada a las funciones encargadas de procesar los datos de entrada de los sensores para formar la trama de 11 bits necesaria para el protocolo de comunicación.
  5. La conexión empieza a establecerse.
    - Se llama a la función de reloj para que se active y se desactive transcurrido un tiempo determinado (por defecto su salida tiene el valor 0)
    - Una vez que se desactiva la señal de reloj, comienza la transmisión de datos.
  6. Conexión establecida, envío de datos

- Se establece un tiempo de oscilación de la señal de reloj, que definirá la velocidad de la comunicación entre los dispositivos
- Se activa la variable de permiso de activación del reloj, para que este pueda realizar el cambio de la variable de salida según el tiempo configurado.
- La información a enviar se encuentra alojada en un vector bidimensional, cuyos valores de tamaño se corresponden con el número de paquetes a enviar y con el tamaño de bits de los mismos, respectivamente.
- Se declaran dos variables de tipo entero, las cuales representan la posición que ocupan en el vector bidimensional antes mencionado y la posición del bit que se está transmitiendo actualmente.
- Dichas variables se incrementan a medida que cada uno de los bits individuales de cada uno de los datos de los sensores son enviados.
- A medida que transcurre el programa, se asignan estas variables al vector que guarda los datos de los bits de los sensores.
- Cuando se termina de enviar la trama completa, el pin digital de datos se configura como entrada para prepararse para una señal de confirmación del dispositivo receptor (Arduino UNO);
- Al recibir la confirmación del dispositivo receptor, se vuelve a declarar el pin de datos como salida para poder transmitir información en el siguiente ciclo. Este comienza de nuevo.

A continuación, se muestra un diagrama de flujo que representa la secuencia del programa antes mencionada:

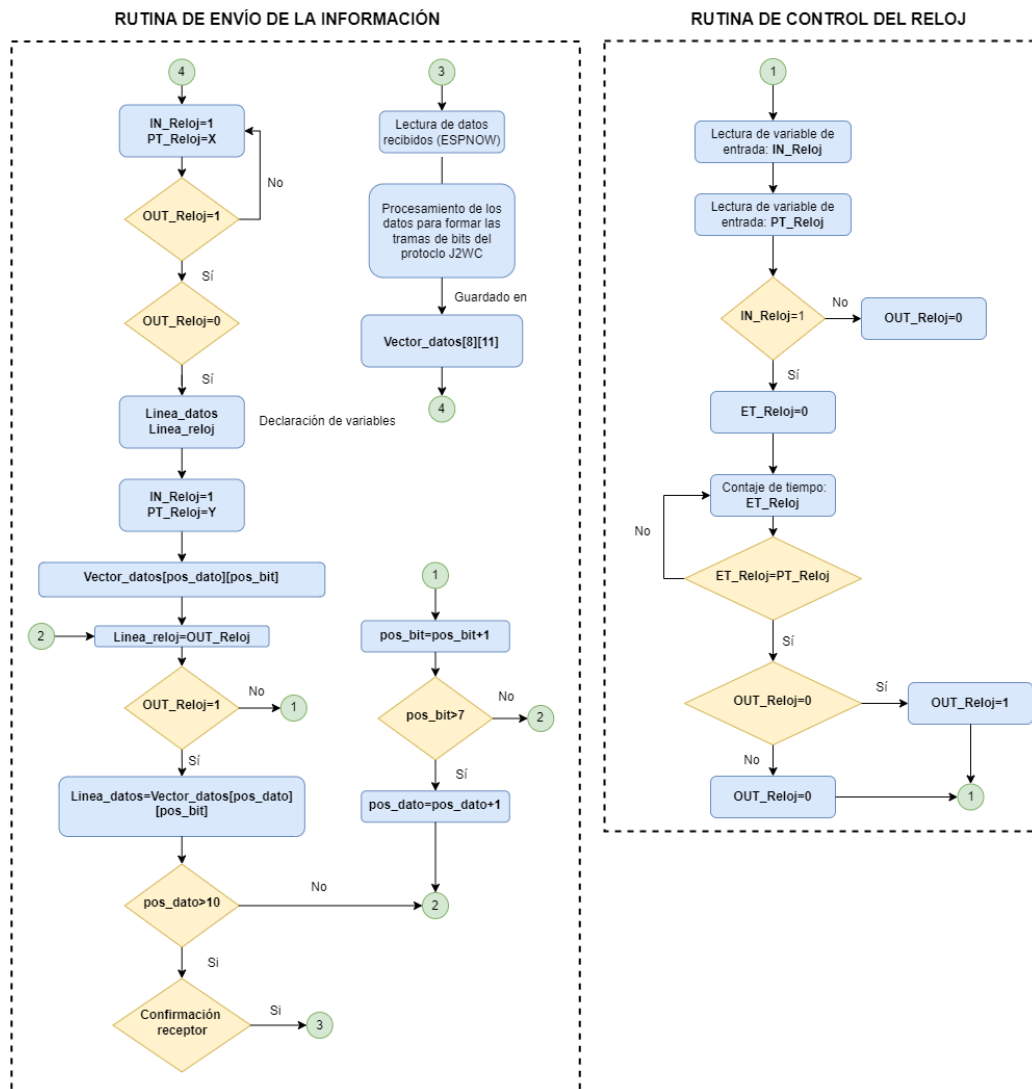


Diagrama de bloques que muestra el funcionamiento de la secuencia de envío de la información

**2. La información del estado de la comunicación se representará mediante Leds y una pantalla LCD 16x2:**

➤ **Leds:**

- **Led rojo:** Representará el estado de la línea de datos.
- **Led azul:** Mostrará el estado de la línea de reloj.

➤ **Pantalla LCD 16x2**

En ella se representarán:

- La latencia de la conexión con el Arduino y con el ESP32 que envía los datos del nunchuk.
- La velocidad de transmisión de los datos.
- El voltaje de la batería conectada al sistema de control manual inalámbrico.

**3. La velocidad de transmisión de los datos será regulada en función de la lectura de un potenciómetro.**

- Se realizará una lectura del valor analógico leído en uno de los pines de entrada de la placa.
- Se escalará el valor de la lectura para regular la velocidad de oscilación de reloj empleada para la transmisión de datos.

#### 4.4.4 Programación del sistema de control de recepción y transmisión de datos (Arduino UNO)

##### 4.4.4.1 Dispositivos que se incluyen en la instalación

- Microcontrolador Arduino UNO.
- 1 led rojo con su correspondiente resistencia.
- 1 led azul con su correspondiente resistencia.
- 1 led verde con su correspondiente resistencia.
- Un módulo Ethernet ENC28J60.
- Una placa de prototipo.
- Cableado entre los componentes mediante cable tipo *jumper*.
- Un cable ethernet.

##### 4.4.4.2 Protocolos de comunicación utilizados

- Modbus TCP/IP
- J2WC

##### 4.4.4.3 Programación del sistema

- Implementación del protocolo J2WC para recibir los datos desde el microcontrolador ESP32 y almacenarlos en un vector que contendrá el valor de los sensores y variables de estado enviadas por dicho microcontrolador

**Los datos se reciben haciendo uso del protocolo J2WC con el siguiente procedimiento:**

- Se utiliza una estructura de tipo *case* que tiene asociada una variable: *stepindex* que se va incrementando secuencialmente según se terminan de realizar las tareas que se incluyen en cada uno de los pasos.
- Se inicia la comunicación mediante el bus de comunicaciones, siguiendo el siguiente protocolo:



1. Se espera a que la señal de reloj esté apagada.
2. Al detectar un flanco positivo en la señal de reloj, se pasa al siguiente paso.
3. Tras detectar un flanco negativo en la señal de reloj, tanto el dispositivo emisor como el receptor estarán preparados para comenzar con la transmisión de la información.
4. Conexión establecida, comienza la recepción de datos  
Cada vez que se detecte un flanco positivo en la señal de reloj, se guardará el estado lógico en el que se encuentre la línea de datos y se almacenará en el vector encargado de recoger la información (vector unidimensional formado por 11 datos de tipo booleano) en la posición correspondiente, que estará determinada por un contador que se incrementará en una unidad cada vez que se detecte un flanco positivo en la señal de reloj.
  - Al superar este contador el valor de 10, significará que se ha terminado de recibir la información de uno de los datos enviados por el ESP 32 emisor. Se procede a:
    - Llamar a la función encargada de procesar este vector, descrita en el siguiente apartado.
    - Incrementar en 1 unidad el contador de datos recibidos que alcanzará su valor máximo (8) cuando se terminen de recibir todos los datos.
  - Cuando el contador de datos recibidos alcance el valor numérico 8, significará que se han terminado de recibir todos los datos, por lo que se procederá a:
    - Mostrar la información de los datos recibidos por el monitor serial.
    - Enviar la información de los datos recogidos al PLC simulado mediante el protocolo Modbus.
    - Enviar una señal de confirmación al dispositivo emisor, para indicarle que está listo para volver a recibir de nuevo los datos.

## Los datos recibidos por el protocolo J2WC son procesados

Cuando la función encargada de recibir los datos comunicados por el ESP32 emisor recibe los primeros 11 bits y los almacena en él vector, llama a la función que se describe a continuación. El papel desempeñado por esta es el de procesar dicho vector para asignarle a una de las variables que se enviarán por Modbus su valor correspondiente. Esta rutina será llamada un total de 8 veces por cada envío completo de datos por parte del ESP32 emisor. A continuación, se describe la estructura y las funcionalidades de esta:

➤ Como entradas recibe:

Un vector formado por 11 datos de tipo booleano.

➤ Realiza el siguiente procesamiento:

### Para formar el número entero

- Lee los valores de los últimos 6 bits del vector de entrada (4-10).
- Mediante un bucle *for*, multiplica cada bit por la potencia de 2 correspondiente. Los resultados de esta multiplicación se van sumando hasta formar el número deseado.

### Para asignar el signo al número entero

- Lee el valor del tercer bit del vector de entrada.
- El estado del bit de signo representa el signo del número entero (0 si es positivo y 1 si es negativo). Se realiza una comparación lógica con el mismo y multiplica por “-1” el número entero formado en el procesamiento anterior si dicho bit tiene como valor lógico un “1”.

### Para detectar el identificador

- Lee los valores de los 3 primeros bits del vector de entrada.
- Mediante un bucle *for*, multiplica cada bit por la potencia de 2 correspondiente. Los resultados de esta multiplicación se van sumando hasta formar el número deseado.

➤ Como salidas devuelve:

- El número entero procesado con su signo.
- El identificador correspondiente a dicho número.

Posteriormente, esta función llama a otra que tiene el papel de formar el vector de datos procesados, al que se accederá para enviarlos por el protocolo Modbus haciendo uso de las rutinas pertinentes.

La función llamada tiene la siguiente estructura:

- Como entradas recibe:
  - El valor del número entero.
  - El identificador de dicho número.
- Como salidas devuelve:
  - Un vector de datos unidimensional de 8 posiciones (correspondientes al número de datos recibidos).
  - Los índices de las posiciones del vector se asignan según el identificador del número que se guarda en dicha posición del vector.

Dicho vector contiene la información de todos los valores que ha enviado el ESP32 emisor.

- **Se procede a inicializar la conexión Modbus TPC/IP con el PLC simulado, y a transmitir los datos que se han recibido utilizando las funciones antes descritas.**
  - Se crea una instancia de un objeto Modbus, en la que realiza un *setup* inicial de la comunicación mediante una función de la librería.
  - Se declaran los registros de salida a los que el cliente Modbus accederá para realizar la lectura de los datos (8 registros de salida con direcciones 50 a 57).
  - Cuando las funciones encargadas de recibir los datos provenientes del ESP32 emisor terminen de enviar dichos datos, se escribirán en los registros que se han declarado anteriormente. Acto seguido, la función Modbus encargada de establecer la comunicación, se encargará de transmitir los datos al cliente alojado en el TIA Portal haciendo uso del módulo ethernet ENC28J60, que se conectará a través del switch virtual creado por el PLCSIM Advanced.

A continuación, se muestra un diagrama de flujo que representa de forma esquemática cómo se realiza la comunicación entre los dispositivos:



### **La información del estado de la comunicación se representará mediante Leds:**

- **Led rojo:** Representará que el sistema se encuentra alimentado correctamente, permaneciendo encendido si es así.
- **Led azul:** Parpadeará cada vez que se realice un envío completo de una secuencia de datos por parte del ESP32 emisor (8 datos).
- **Led verde:**
  - Apagado: no se ha establecido conexión con el cliente Modbus.
  - Encendido fijo: conexión establecida con el cliente Modbus, pero los datos no están siendo enviados.
  - Parpadeando: la conexión con el cliente se ha establecido de manera satisfactoria y además los datos están siendo enviados.

#### 4.4.5 Programación del sistema en TIA Portal.

Este programa será el encargado de recibir los datos transmitidos por los otros sistemas, además de procesarlos con el fin de poder controlar los movimientos del sistema robótico simulado. Se detallan a continuación algunas de sus características principales y funciones que desempeñan sus programas.

##### 4.4.5.1 Protocolos de comunicación utilizados

- Modbus TCP/IP
- Protocolo S7 propio de Siemens para realizar la comunicación entre las variables del PLC y las del sistema SCADA

##### 4.4.5.2 Funcionalidades principales

- Realizar la conexión Modbus a través de un cliente que se ejecuta en el PLC para leer las variables alojadas en los registros de la placa Arduino.
- Procesar los datos recibidos para que puedan ser correctamente interpretados por las rutinas de control del sistema robótico simulado a través de:
  - Una rutina de procesamiento de datos
  - Una librería de tipo “*objeto eje*” a través de la cual se consiguen controlar los 6 ejes del robot individualmente haciendo uso de 6 instancias de esta librería, pudiendo ajustar los parámetros de configuración de cada uno de ellos por separado para adaptarlos a las necesidades y restricciones que ofrecen los ejes del robot.
  - Realizar un guardado de varias de las posiciones actuales de los ejes, para que posteriormente se pueda mover el sistema robótico a las posiciones que previamente se han guardado.
- Implementar una función que se encargue de calcular el tiempo de latencia entre las conexiones establecidas entre los sistemas con los que se comunica:
  - La placa Arduino UNO.
  - El sistema robótico.

#### 4.4.5.3 Comunicación Modbus TCP/IP

- **Implementación:**

Mediante el uso de un bloque perteneciente a las librerías propias de Siemens, se llama desde las rutinas de proceso principales.

- **Configuración:**

Para establecer la conexión de manera satisfactoria, previamente se ha de crear un DB (*Data block*), que se asignará como entrada en la función Modbus. Los parámetros a configurar serán los siguientes:

- **REQ:** Señal de tipo booleana que permitirá que la conexión empiece a ejecutarse.
- **MB\_MODE:** Parámetro entre 0 y 1 que configura a la función para leer o escribir registros de entrada o salida. En este proyecto se le da el valor 0, ya que se procederá a leer los datos alojados en los registros del Arduino.
- **MB\_DATA\_ADDR:** Número que indica la dirección de memoria por la que se va a empezar a leer los datos. En este proyecto se utilizará la función 4 (leer registros de entrada) por lo que empieza a leer registros desde la dirección 30000. En este proyecto, sin embargo, la dirección de inicio de lectura de datos es de 30051, ya que en la placa Arduino los registros empiezan por el valor 51 (representa el offset que se suma a la dirección 30000).
- **MB\_DATA\_LEN:** Número de registros de entrada que deberá de leer la función. Tomará el valor de 8, que será el número de datos leídos en los registros de la placa Arduino.
- **MB\_DATA\_PTR:** Recibe como entrada la dirección del DB de datos en el que se guardarán las variables recibidas en la comunicación.
- **CONNECT:** Se le asocia un DB de datos en los que se incluye una variable de tipo *TCON\_IP\_v4*. Es una estructura que contiene los datos necesarios para realizar la comunicación necesaria con el dispositivo receptor, entre ellos:
  - **Interface ID:** número de identificación de hardware del PLC en cuestión (64).
  - **ID:** Identificador del dispersivo receptor (3).
  - **ActiveEstablished:** Mediante él se configura si la conexión va a ser pasiva o activa. Se configura en este caso con el valor 1 (conexión activa).

- **RemoteAddress:** Dirección IP del dispositivo desde el cual se procederá a leer los datos, en este caso de la placa Arduino UNO.

La IP configurada será la siguiente: 192.168.1.25

- **Remote Port:** Puerto mediante el cual se establecerá la comunicación, por defecto toma el valor de 502.

- **Estado de la conexión:**

La función Modbus representa el estado de la comunicación mediante salidas. Se hará uso del valor de las mismas para visualizar el estado de la comunicación en los dispositivos que intervienen en el proyecto, en concreto:

- Status: representa un código en hexadecimal.
- Error: salida digital que se activa si existe algún tipo de error en la comunicación.

#### 4.4.5.4 Librería para el control de los ejes.

La funcionalidad de esta librería es de permitir controlar individualmente a cada uno de los ejes del robot. Para ello, en la función de procesamiento de datos, se crearán seis instancias individuales de esta función, permitiendo dotar a cada eje de una configuración y modo de control personalizado.

- **Estructura de la función:**

- **Entradas**

- Permiso de activación: si está activa, la función podrá actuar sobre el control de los ejes.
- Permiso de giro a la derecha y a la izquierda: en función del estado de estas variables, el ángulo calculado para controlar al eje del robot se irá incrementando o decrementando.
- Ángulos de giro mínimos y máximos: son variables que limitan el ángulo máximo y mínimo de giro calculado por la rutina. Estos valores son necesarios debido a que el robot tiene ciertas restricciones en el movimiento de sus ejes.



- Tiempo de paso: variable que será responsable de definir cada cuanto tiempo se ha de incrementar la posición del eje en un valor determinado, para limitar la velocidad con la que este se mueve.
- Offset del ángulo: define cuanto ha de incrementarse el ángulo de giro del robot, por cada ciclo de programa.

➤ **Salidas**

- La función devuelve el ángulo calculado por la función en grados sexagesimales.

• **Programación de la función:**

Se llama a un temporizador, que empezará a contar un tiempo hasta que alcance el valor de la variable “tiempo de paso” antes mencionada. Activará su salida y permitirá que el ángulo del eje se incremente o decremente (según las variables de “permiso a la derecha e izquierda”) una determinada cantidad (definida por la variable “Offset del ángulo”).

Se procede a resetear el valor del temporizador para prepararlo para el siguiente control de movimiento.

Se limitarán los valores de los ángulos calculados según las variables (“Ángulos de giro mínimos y máximos”) para evitar que superen los valores permitidos.

#### 4.4.5.5 Función de procesamiento de datos y de control del robot

Encargada procesar la información recibida por Modbus, que se corresponden con los datos de los sensores del *nunchuk*, entre otros.

Tras el procesamiento, llama a las instancias creadas de cada uno de los ejes del robot, controlando a cada uno por separado mediante las variables correspondientes. Finalmente, según los datos de salida proporcionados por las rutinas de eje, prepara variables que serán comunicadas con el sistema SCADA para poder visualizar diversos parámetros que se detallarán a continuación.

- **Estructura de la función:**

- **Entradas (datos recibidos por Modbus)**

- Valor del valor de posición del joystick del nunchuk (ejes X e Y).
- Valor del estado del giroscopio del nunchuk (ejes X e Y).
- Valor del estado de los dos botones del *nunchuk* (C y Z).
- Tiempo de latencia de conexión entre el dispositivo de control manual autónomo y el ESP32 receptor.
- Tiempo de latencia de conexión entre el ESP32 receptor y la placa Arduino UNO.
- Tiempo de latencia de la conexión Modbus entre el cliente y el servidor (Arduino UNO).
- Tiempo de latencia de la conexión OPC entre el sistema SCADA y el Robot Studio.
- Variables comunicadas a través de la conexión S7 con el sistema SCADA al PLC:
  - Estado de las posiciones de los ejes en coordenadas cartesianas.
  - Estado de botones de control implementados en el SCADA para controlar variables de proceso dentro de la rutina.
  - Valor de los campos de entrada del SCADA que parametrizan el tiempo de paso la velocidad de movimiento de los ejes.

- **Salidas**

La función devuelve los siguientes valores.

- Valores procesados de las posiciones de los ejes en ángulos sexagesimales.
- Estado del par de ejes controlados.
- Velocidad de movimiento de los ejes.
- Estado de la activación de la ventosa neumática del sistema robótico
- Estado de los tiempos de latencia entre los sistemas.

Estas variables serán comunicadas al sistema SCADA para que puedan ser representadas.

- **Programación de la función**

- Se realiza la lectura de las variables de entrada y se guardan en variables auxiliares de programa para poder procesarlas y modificar sus valores.
- Se les asignan valores a los parámetros de configuración del control de los ejes.
- Se declaran y se activan los temporizadores encargados de calcular los tiempos de latencia entre las comunicaciones de los dispositivos.
- Procesado y filtrado de los valores de los sensores recibidos para realizar el control del sistema robótico de manera correcta:
  - Se establecen mediante parámetros, valores que determinan los valores máximos y mínimos numéricos que tienen los sensores, para poder hacer un correcto escalado de ellos para incrementar o decrementar los ángulos de los sensores de manera controlada.
  - De la misma forma, haciendo uso de estos parámetros de configuración, se determinan unos rangos de margen máximos y mínimos, de manera que el valor de los ángulos de los ejes del robot no se modifique si los datos del sensor no superan dichos valores.

Esta consideración es necesaria, ya que, aunque no se esté actuando físicamente en el dispositivo de control (*nunchuk*) los valores de los sensores cambian debido a la sensibilidad de estos a los cambios.

En esta imagen se muestra un ejemplo de lo descrito anteriormente:

Valor mínimo	Estado de reposo del sensor			Valor máximo
Fijo	Oscilación negativa	Valor ideal en reposo	Oscilación positiva	Fijo
-250	-17	0	35	200

*Representación esquemática de los valores del sensor*

La función encargada de realizar el escalado de los valores del sensor, ignorará las oscilaciones en el valor del mismo, que ocurren, aunque no se esté manipulando el controlador.

- Una vez realizado el escalado numérico correcto de los sensores, se emplea el valor de este escalado para discernir si los ejes han de incrementar o decrementar su valor de posición y a la velocidad a la que han de hacerlo.
- El programa está diseñado para controlar sólo dos ejes del robot de manera simultánea. Para ir cambiando entre los ejes a controlar, se leerá el valor de uno de los valores del *nunchuk* (botón C), que irá incrementando una variable que dará permiso a los ejes correspondientes para que puedan realizar su movimiento.
- Implementa dos modos de control diferentes para el sistema robótico:
  - Haciendo uso del joystick del *nunchuk*.
  - Mediante el giroscopio del *nunchuk*.

Se podrá cambiar entre estos modos de control bien desde un botón situado en el HMI o bien desde el propio *nunchuk*, realizando una determinada combinación de botones y movimientos con el mismo.
- Tras realizar el procesamiento de todas las variables, se llamarán a las rutinas encargadas de realizar el control de los ejes, asignándoles a las entradas los comandos necesarios y guardando el valor de sus salidas, que se corresponden a la posición individual de cada uno de los ejes, en variables internas de programa.
- Se llama a una función encargada de gestionar unas posiciones de guardado de los ejes y de enviarlos a dichas posiciones en el caso de que le sea indicado.
- Por último, se escriben en el HMI:
  - Valores y posiciones actuales de los ejes.
  - Grupo de ejes controlados.
  - Velocidad de movimiento de los ejes.
  - Estado de la ventosa neumática.
  - Tiempo de latencia existente entre los dispositivos que intervienen en la comunicación.

#### 4.4.5.6 Función de guardado de posiciones del robot

Función encargada de almacenar cinco posiciones actuales del robot, con el objetivo de poder enviarlas a la rutina encargada de gestionar el control de los ejes para dirigir al robot a las posiciones que se han guardado cuando el usuario decida.

Se crea un tipo de datos de usuario que almacenará una estructura formada por los siguientes elementos:

- Posiciones individuales de los seis ejes.
- Posición de la ventosa neumática en coordenadas cartesianas.
- Descripción de la posición de guardado.

En la función se crea un vector de cinco posiciones en las cuales se alojan los datos pertenecientes a las estructuras que se han descrito anteriormente.

- **Estructura de la función:**

- **Entradas**

- Comandos de guardado y de envío a la posición guardada leídos desde variables HMI
    - Estado actual de las posiciones de los ejes.
    - Descripción de la posición de guardado, leída desde una variable de tipo *String* de un campo de texto también situado en el HMI.

- **Salidas**

- Posición de guardado de los ejes.
    - Descripción de la posición de guardado.

- **Programación de la función**

- En el caso de recibir el comando de guardado de posición, almacenará en el vector antes descrito los valores de:
    - Las posiciones de los ejes en grados sexagesimales.
    - La posición de la ventosa en coordenadas cartesianas.
    - La descripción de la posición de guardado introducida por el usuario en el HMI.
  - Al recibir la orden de enviar al robot a la posición de guardado determinada, se ejecutarán las siguientes instrucciones:

- Se leerá de la posición correspondiente del vector de guardado de posiciones, los datos almacenados.
- La función accederá a las variables que controlan el ángulo de los ejes situadas en la función de procesamiento de datos y control del robot.
- Actualizará los valores del ángulo actual de cada una de las seis instancias de los objetos de *tipo eje*, para que puedan manipular las posiciones de dichos ejes a partir de esa posición de guardado.

#### 4.4.6 WinCC Advanced

Es un sistema que se encuentra implementado en el software TIA Portal, sin embargo, se ha decidido separar este del apartado anterior, ya que cumple con funciones distintas.

##### 4.4.6.1 Funciones principales que desempeña

- Leer las variables alojadas en las tablas de variables del PLC para poder visualizarlas en el sistema HMI.
- Crear un cliente OPC para poder acceder al servidor OPC creado por el programa ABB IRC5 OPC Server, en el que se alojarán las variables compartidas por la controladora virtual del sistema robótico simulado.
- Visualizar todos los datos y estados actuales tanto del programa como del sistema robótico mediante campos de salida.
- Proporcionar, mediante botones y campos de salida, unas entradas de control que serán leídas por las funciones encargadas de realizar el procesamiento de los datos y el control de los ejes del robot.

##### 4.4.6.2 Configuración del cliente OPC

- Se habilitará la opción de crear un cliente OPC en el apartado de configuración del *runtime*.
- Se creará una conexión de tipo OPC, que conectará el cliente OPC alojado en el sistema con el servidor OPC creado por el programa ABB IRC5 OPC server.
- Se hará uso de una tabla de variables mediante la cual se accederá a las variables que compartirá la controladora virtual a través del servidor OPC.

- A dichas variables se les asignará en un apartado de la tabla la conexión OPC creada anteriormente, para que puedan ser actualizadas de manera correcta a medida que la controladora va actualizando sus variables.

#### 4.4.6.3 Configuración de las tablas de variables

Se crearán tres tablas, encargadas de guardar los valores de estado de las variables pertinentes:

- **Tabla de datos PLC**

En ella se almacenarán las variables de las funciones de control del robot, compartidas mediante una conexión personal de Siemens (conexión S7).

- **Tabla de datos del guardado de posiciones**

En ella se almacenarán las variables de la función encargada, de gestionar las posiciones guardadas de los ejes del robot. Estas se compartirán también mediante la conexión S7 antes mencionada.

- **Tabla de datos OPC**

En esta tabla se almacenarán las variables compartidas entre el cliente y el servidor OPC, que se actualizarán mediante la conexión OPC creada en la configuración antes descrita.

#### **Actualización de los datos de las variables entre las tablas:**

Las variables que hay en las tablas, han de ser actualizadas periódicamente para satisfacer las necesidades de control y de representación de datos necesarias para que la implementación de todos los sistemas pueda funcionar de manera correcta.

Para ello es necesario que las variables que necesiten ser leídas por la tabla de datos OPC puedan actualizar sus valores con las variables de la tabla de datos PLC, y viceversa.

Sin embargo, desde el PLC no se puede acceder a la tabla de datos OPC debido a que las variables que se encuentran en ella no están compartidas mediante la conexión S7 de Siemens, sino a través de la conexión OPC. Para solventar este problema, se ha hecho uso de una funcionalidad que incorporan las tablas de variables:

- **Uso del evento que se ejecuta cada vez que la variable correspondiente cambia de valor:**

Cada vez que se modifica el valor de una variable, se produce un evento al que se le pueden asignar las funciones que sean necesarias que se ejecuten cada vez que esto ocurra. Entre estas funciones, existe una llamada “DefinirVariable”, que realiza el siguiente procedimiento cada vez que se activa:

- Lee el valor de cualquier variable situada en alguna de las tablas de variables HMI.
- Asigna dicho valor a otra variable en cualquiera de las tablas.

Al permitir esta función el acceso a cualquier tabla se solventa el problema antes mencionado.

Sin embargo, para que esto funcione, es necesario incorporar un botón en la pantalla en la que se visualizan los datos con los mismos eventos con las funciones asignados a las variables anteriores. No hace falta que se actúe sobre él para que se actualicen las variables, con su mera presencia en la pantalla es suficiente.



#### 4.4.6.4 Pantalla de visualización de datos

Desde esta pantalla se podrán tanto visualizar como controlar algunos de los parámetros de funcionamiento del sistema:



Imagen de visualización de los datos y de los elementos de control del programa

- **Visualización y representación de datos:**

- Posiciones actuales de los ejes (en grados sexagesimales).
- Posiciones de la ventosa neumática (en coordenadas cartesianas).
- Velocidades de movimiento de los ejes (en grados sexagesimales por segundo)
- Pareja de ejes activados.
- Estado de la ventosa neumática (activada o desactivada).
- Modo de control empleado para manejar los ejes (a través del joystick o el giroscopio).
- Estado de los tiempos de latencia entre los dispositivos que forman el sistema.
- Estado de la posición actual de guardado.

- **Configuración de parámetros y elementos de control del proceso**

- Configuración de la velocidad máxima de movimiento de los ejes (en grados sexagesimales por segundo).
- Configuración del tiempo de paso mediante un campo de entrada, que determina cada cuanto tiempo se modifica el valor de posición de los ejes (en milisegundos).
- Selección de la latencia entre los dispositivos que se quiere visualizar.
- Configuración del modo de control de los ejes (a través del joystick o el giroscopio).
- Comando de guardado de posición actual del robot .
- Comando de ir a la posición guardada.
- Selección del número de la posición sobre la que se quiere guardar o leer datos.
- Restablecimiento del valor de los ejes, (todos toman el valor de 0°).

#### 4.4.7 Servidor OPC (ABB IRC5 OPC Server)

Programa de ABB encargado de establecer la comunicación con la controladora virtual del sistema robótico simulado y de compartir las variables alojadas en ella mediante un servidor OPC.

- Mediante una interfaz, localiza la controladora virtual del programa Robot Studio, y permite añadirla al Servidor OPC asignándole alguno de los siguientes parámetros para que pueda ser localizada por el cliente:
  - Nombre de la controladora virtual.
  - Dirección de la controladora virtual
  - ID del sistema.
  - ID del controlador.
  - Nombre del sistema.

Para la configuración del proyecto, se detecta automáticamente la controladora y se añade al servidor OPC identificándola mediante el nombre que tiene.

- Ajustes de la comunicación:
  - El programa dispone de distintos parámetros de configuración para la conexión:
    - Encriptación de la misma mediante usuario y contraseña.
    - Idioma utilizado para mostrar los mensajes de diagnóstico.

- Tiempo de actualización de las variables OPC (entre 10 y 60000 ms).

La ejecución de los programas y la gestión de los recursos del programa Robot Studio se ven afectados por este tiempo de actualización, a medida que este es más pequeño, enlentece la ejecución de la simulación y de las funciones del programa.

Para este proyecto se le ha asignado un tiempo de actualización de 50ms, que ha proporcionado un buen tiempo de respuesta en la actualización de las variables compartidas y no ha enlentecido la ejecución de la simulación del brazo robótico.

- Ejecución del servidor:  
Mediante un botón de inicio y de parada, se gestiona el estado de ejecución del programa. Cada vez que se incluyen variables nuevas en la controladora virtual que se desea que sean compartidas, hay que reiniciar el servidor.

#### 4.4.8 Programación en Robot Studio

Programa encargado de realizar la simulación del sistema robótico. A continuación, se describe como se ha configurado el sistema, compuesto principalmente por dos elementos:

##### 4.4.8.1 Configuración de la estación

La estación es el entorno en el que se encuentran tanto el sistema robot con sus correspondientes herramientas, como objetos y elementos físicos que tienen un comportamiento dinámico cuando se está ejecutando la simulación.

A continuación, se detallan los componentes que forman la estación:

- **Elementos físicos:**

- Sistema robótico utilizado: IRB1600\_6\_120\_02.
- Herramienta del robot: ventosa neumática.
- Dos cubos de distinto tamaño que tendrán un comportamiento dinámico en la simulación, para que puedan ser manipulados por el brazo robótico.
- Superficie sobre la que irán apoyados los cubos, que tendrá un comportamiento estático en la simulación, no podrá ser manipulado.

- **Elementos de software**

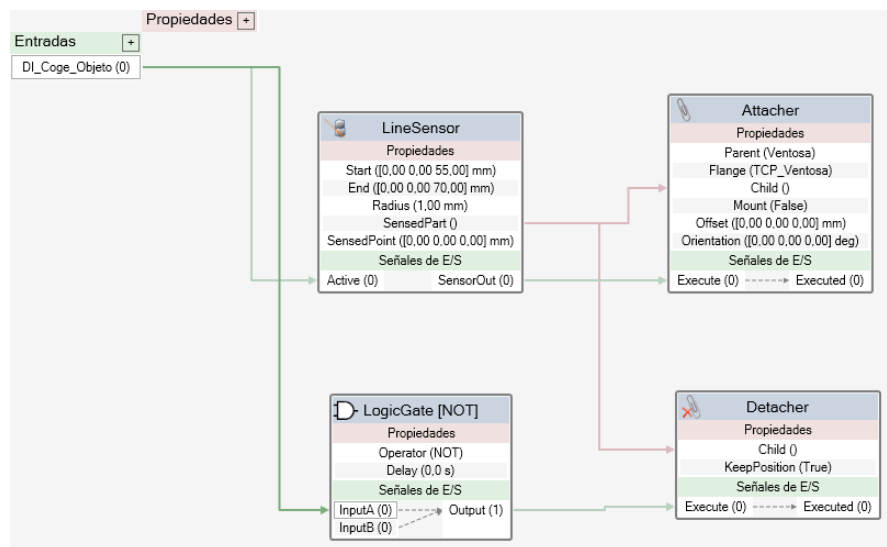
Se ha hecho uso de un recurso del programa llamado *smart component*, que permite asociarles a los objetos físicos que participan en la simulación propiedades para bien hacer que se comporten de una manera determinada o para leer información acerca del estado de los mismos, pudiendo posteriormente realizar una lógica de programación que permita interactuar con ellos desde la programación de la controladora.

Para este proyecto en concreto, se ha utilizado un *smart component* con las siguientes características:

- Empleado para interactuar con la herramienta del robot, una ventosa neumática.
- Mediante una entrada física de control, se actúa sobre ella para que entre ella y el objeto con el que está en contacto, se forme un enlace físico, de manera que dicho objeto permanezca unido a la ventosa y pueda seguir los movimientos del robot, participando en la simulación física.

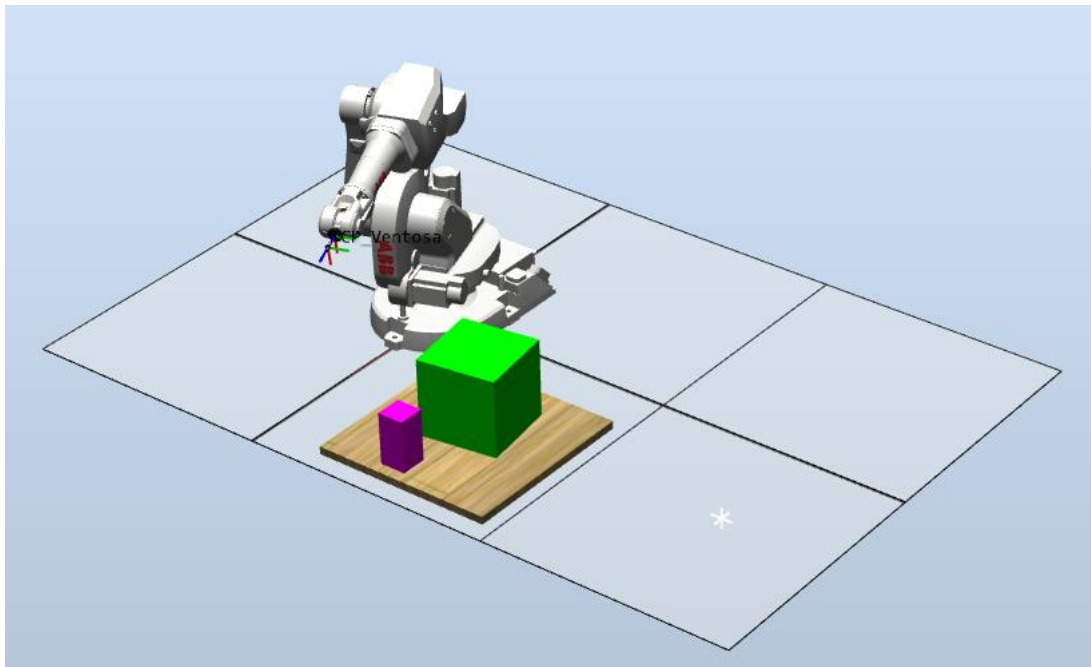
- Cuenta con una estructura interna de programación mediante bloques, entre ellos:
  - Un bloque que se encarga de realizar la unión entre la ventosa y el objeto físico al que se conecta.
  - Un bloque que deshace dicha unión.
  - Un detector de barrera que detecta cuando hay un objeto cerca del extremo de la ventosa.
  - Una puerta lógica de tipo *not*, que se encarga de controlar la activación y la desactivación del acople de la ventosa con el objeto en cuestión.

A continuación, se muestra un esquema de conexionado entre los bloques que se han definido anteriormente:



*Representación de la conexión lógica entre los bloques del smart component*

Todos los elementos descritos anteriormente, quedan dispuestos de la siguiente manera en la estación:



*Disposición física de los elementos de la estación*

#### 4.4.8.2 Programación de la controladora

La controladora es el sistema que se encarga de controlar los movimientos del robot, en este proyecto estará simulada.

- **Funciones principales que desempeña:**

- Ejecutar el programa cargado en su memoria, para poder controlar el estado del robot.
- Almacenar variables, las cuales serán compartidas mediante OPC haciendo uso del programa ABB IRC5 OPC.

- **Estructura de la programación:**

- Una función principal (*main*), encargada de ejecutar las siguientes subrutinas:
- Función de control de los ejes del robot.
- Función encargada de calcular la posición de la herramienta en coordenadas cartesianas.
- Función responsable de devolver el valor del contador de vida recibido por el servidor OPC a través de este.

- **Entradas**

- Comandos para las posiciones de los seis ejes del robot.
- Comando para activar o desactivar la ventosa neumática.
- Contador de vida del PLC.

- **Salidas**

- Posición de la ventosa en coordenadas cartesianas en los tres ejes (x, y, z).
- Contador de vida.

- **Programación de las funciones**

- **Función de control de los ejes**

Recibirá los valores de entrada de los comandos de las posiciones recibidas por OPC, y se los asignará a cada una de las variables que leerá la controladora para enviar al eje del robot al ángulo correspondiente.

Ejecutará una función que recibirá como entrada los valores de los ángulos de los ejes, y realizará el control de los mismos hasta llevarlos a dicho ángulo.

➤ **Función para leer la posición de los ejes**

Ejecutará una función propia de RAPID, que almacenará en ella los datos de las posiciones cartesianas de la posición de la ventosa en una variable.

Posteriormente, se almacenarán en las variables de salida los valores almacenados en la variable de retorno de la función, accediendo individualmente al correspondiente valor de la coordenada cartesiana almacenado en la estructura de dicha variable de retorno.

➤ **Función para calcular la latencia.**

Recibe como entrada el valor del contador de vida enviado por el PLC a vía OPC a la controladora, y asignará ese mismo valor a una variable de salida para comunicarlo a través del servidor OPC para que el PLC pueda recibirla.

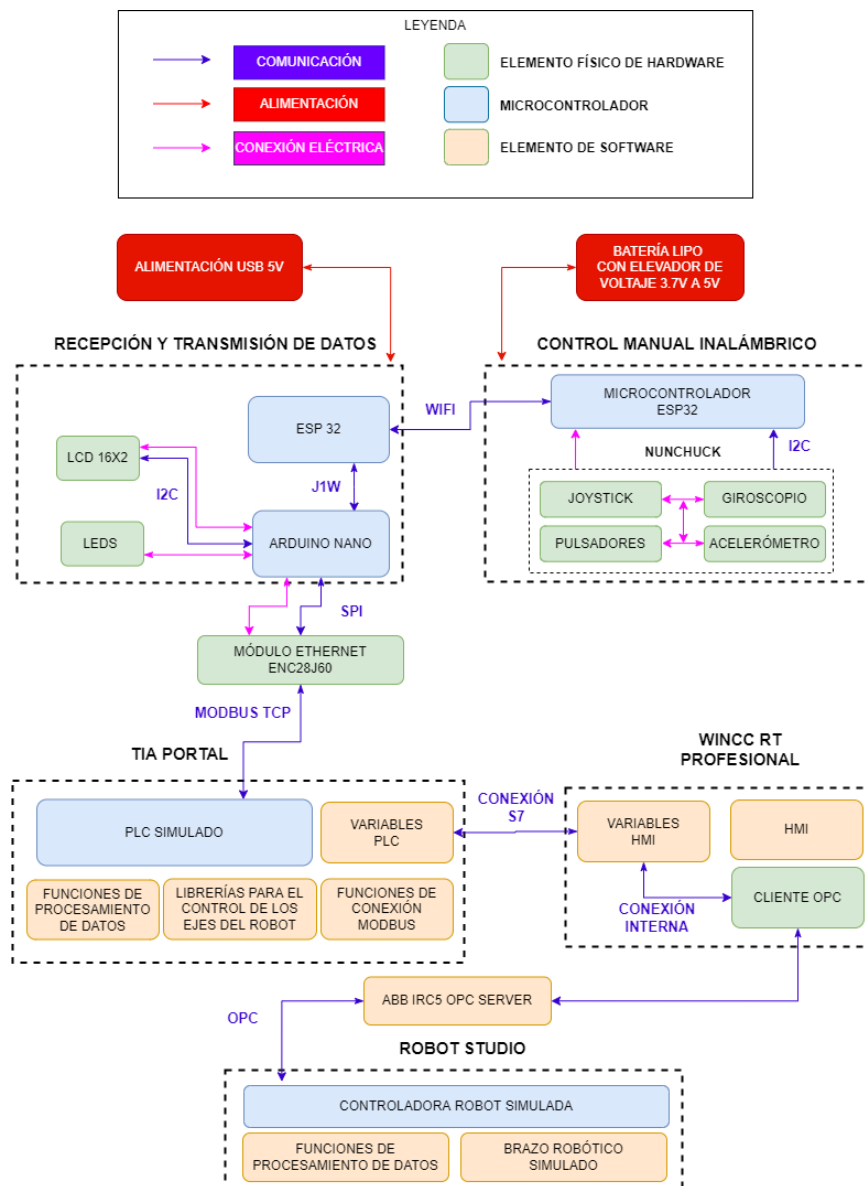
El tiempo de latencia será calculado por el PLC mediante un temporizador, que calculará el tiempo que tarda en recibir el mismo valor del contador de vida que ha enviado.



## 5. Planos y esquemas

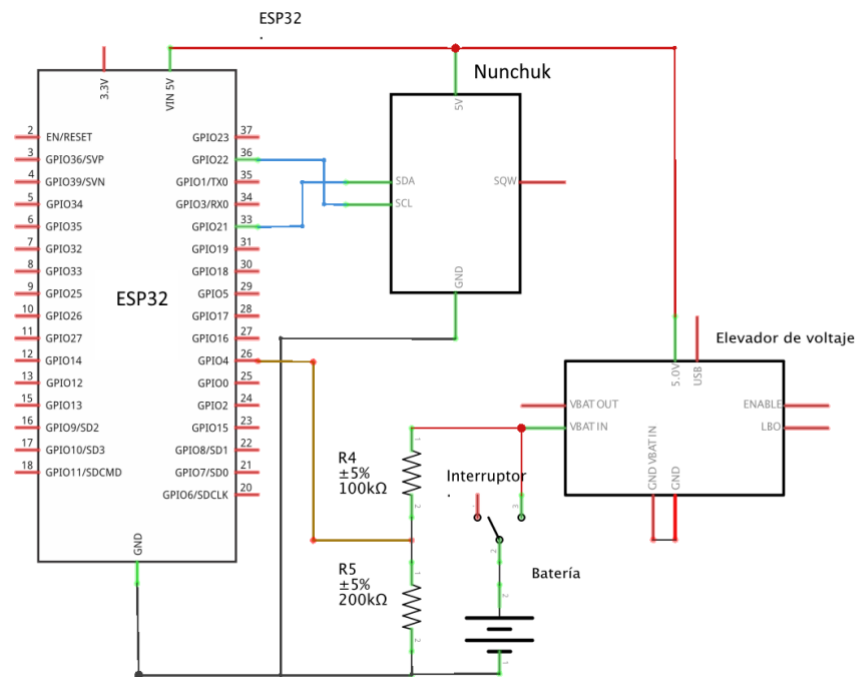
### 5.1 Esquema general de los sistemas involucrados

El siguiente esquema muestra de manera esquemática todos los dispositivos que conforman la instalación, así como las conexiones y relaciones que hay entre ellos:

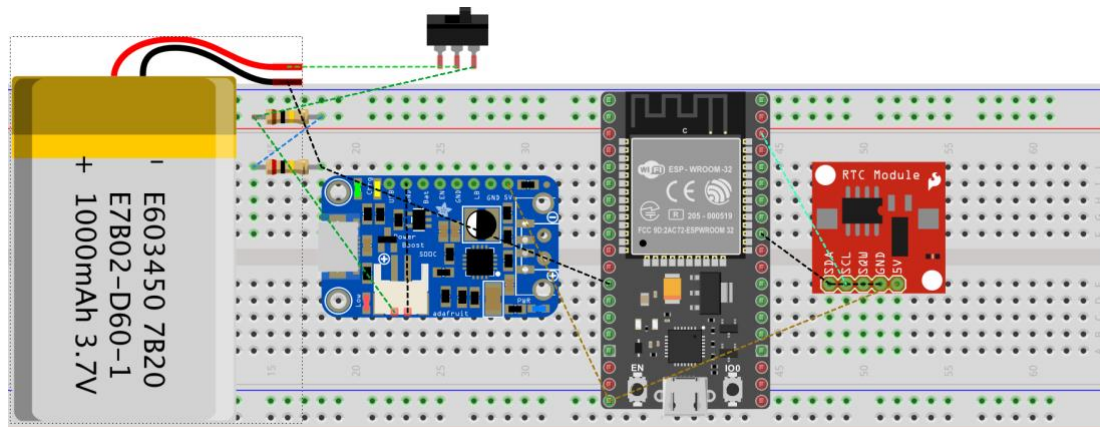


Esquema que muestra los dispositivos que componen la instalación y cómo están comunicados

## 5.2 Dispositivo de control manual inalámbrico

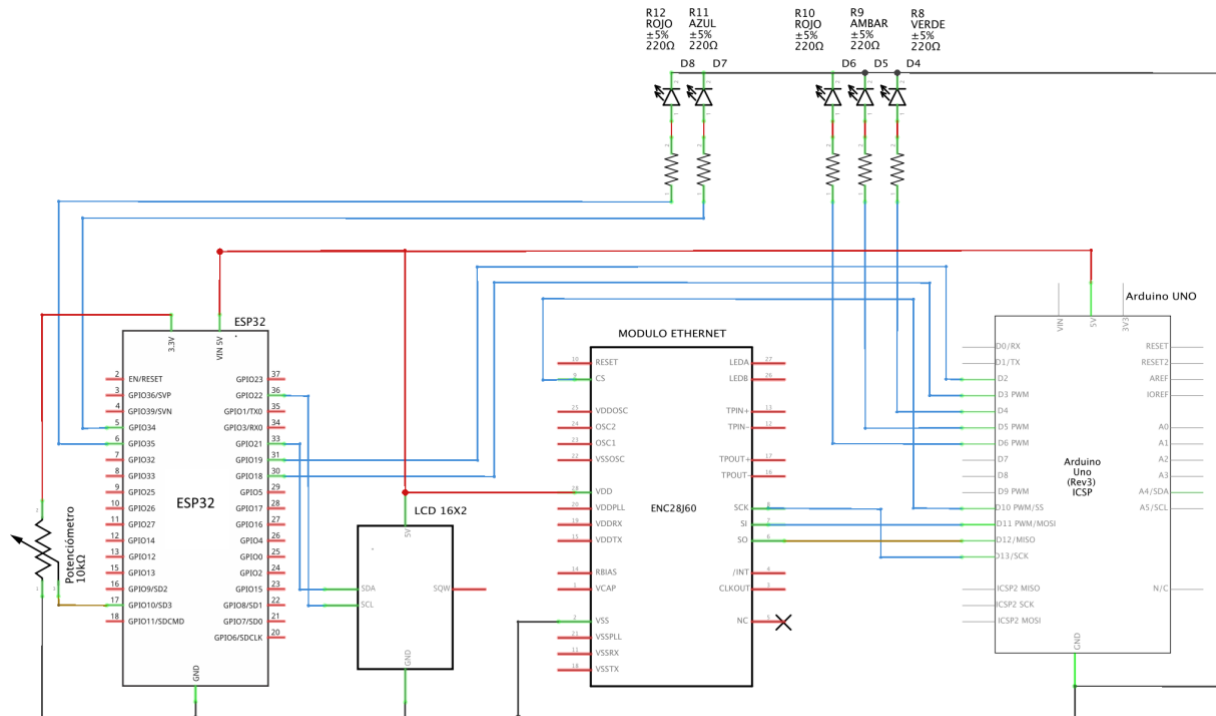


Esquema de conexionado eléctrico entre los dispositivos

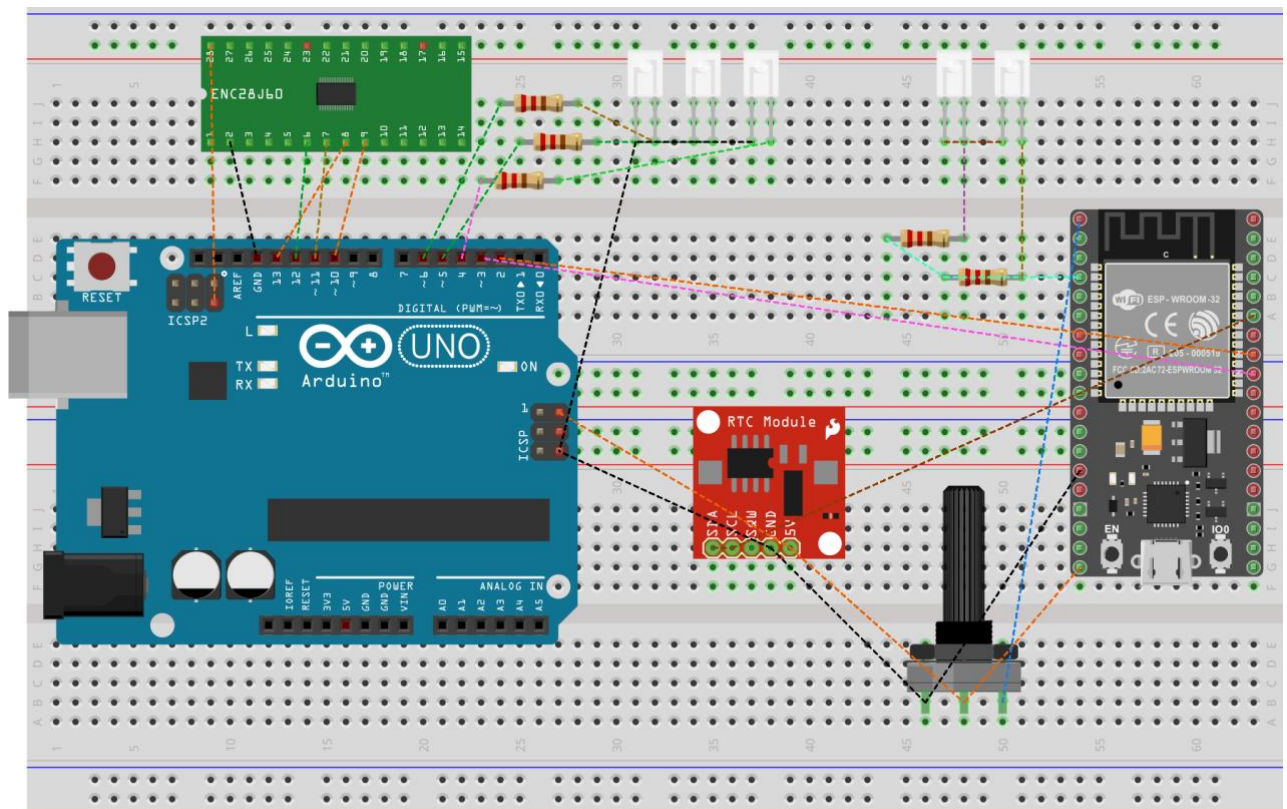


Disposición de los elementos en la placa de prototipo

## 5.3 Dispositivo de recepción y transmisión de datos



*Esquema de conexión eléctrica entre los dispositivos*



*Disposición de los elementos en la placa de prototipo*

## 6. Presupuesto

A continuación, se muestra una tabla en la que se incluyen todos los componentes de hardware que forman parte del proyecto. Se han omitido en lo que al software se refiere, el precio de las licencias requeridas para utilizar algunos de los programas.

PRESUPUESTO DEL PROYECTO					
ID	DESCRIPCIÓN	CANTIDAD	PRECIO	FUENTE	SUBTOTAL
1	Microcontrolador ESP32	2	11,99 €	Amazon	23,98 €
2	Microcontrolador Arduino UNO	1	19,49 €	Amazon	19,49 €
3	Controlador remoto Wii ( <i>nunchuk</i> )	1	10,50 €	Amazon	10,50 €
4	Batería LIPO 220mAh	1	10,99 €	Amazon	10,99 €
5	Circuito elevador de voltaje	1	1,39 €	Amazon	1,39 €
6	Interruptor	1	0,79 €	Amazon	0,79 €
7	Led	7	0,08 €	Amazon	0,56 €
8	Pantalla LCD 16x2	1	6,49 €	Amazon	6,49 €
9	Módulo I2C para pantalla LCD 16x2	1	2,66 €	Amazon	2,66 €
10	Módulo Ethernet ENC28J60	1	10,49 €	Amazon	10,49 €
11	Placas de prototipo	3	3,83 €	Amazon	11,49 €
12	Cables de tipo Jumper	1	1,57 €	Amazon	1,57 €
13	Cable Ethernet	1	7,15 €	Amazon	7,15 €
14	Potenciómetro 10kΩ	1	0,57 €	Amazon	0,57 €
15	Ordenador PC Latitude 3410	1	930,53 €	PC componentes	930,53 €
16	Mano de obra	60	14,99 €	-	899,40 €
				<b>Precio sin IVA</b>	<b>1.938,05 €</b>
				<b>Precio con IVA (21%)</b>	<b>2.345,04 €</b>

*Tabla que muestra los componentes utilizados, su precio y el precio total de los mismos*

## 7. Trabajos futuros

En este apartado se hablará de las posibles mejoras que se pueden y se quieren implementar en el proyecto, con el objetivo de hacerlo más completo y funcional y dotarlo de un carácter más profesional.

### 7.1 Mejorar el protocolo J2WC

- **Para que transmita la información de manera más rápida:**

- Aumentando la velocidad a la que se realiza la comunicación.
- Empaquetando los datos de manera que ocupen menos espacio
- Ignorando los datos que no son necesarios de enviar:  
Si por ejemplo el valor de un sensor no ha cambiado en el transcurso de un ciclo de programa a otro, no sería necesario enviar esa información, ya que sería redundante.
- Hacer uso de los registros del Arduino para leer las entradas digitales en vez de emplear la función "*digitalRead()*".

- **Implementar un sistema de detección de errores en la comunicación:**

Además de enviar la información de los sensores, se enviará una trama de bits, comúnmente llamada *checksum*, en cuyo contenido habrá información resumida de la información previamente enviada, haciendo uso por ejemplo de bits de paridad o de otros métodos.

El receptor, al comparar la información recibida con el *checksum*, y detectar el error en el proceso de transmisión de los datos, avisará al emisor para que vuelva a enviar la información.

### 7.2 Mejorar el conexionado eléctrico de los componentes de hardware

Se han hecho uso de placas de prototipo y cables de tipo *jumper* para realizar la conexión de los equipos, es funcional y fácil de implementar, pero no muestra profesionalidad y la estética es mejorable. Para corregir estos aspectos se proponen dos soluciones posibles:

- **Montar los dispositivos en placas de circuito impreso y realizar el conexionado mediante soldaduras**

Se evita así el tener que utilizar los cables *jumper* para conectar los dispositivos entre sí, y se reduce a su vez el tamaño que ocupan todos los componentes conectados.

- **Utilizar un software de diseño de placas de circuito impreso para crear una placa personalizada.**

Existen gran variedad de programas con los que se pueden realizar diseños propios de placas PCB. Una vez diseñadas, se pueden “imprimir” de manera casera mediante el uso de productos químicos, o enviar el diseño a empresas que se encargan precisamente de realizar esta función.

Aplicando este método, se consigue:

Que la integración de los componentes en el sistema sea compacta y cuente con una mejor estética, evitando además el uso de las soldaduras para realizar la unión entre los componentes.

La posibilidad de poder realizar múltiples veces la integración de los componentes en proyectos diferentes, ya que las placas están diseñadas.

### 7.3 Rediseñar la estructura de la estación del programa Robot Studio.

Tal y como se ha presentado el proyecto, la estación cuenta con una instalación simple y con no muy buena estética. Se propone como mejora el rediseñar la estética de los objetos haciendo que se asemejen a elementos reales, y que la estación cuente en general con una apariencia que se corresponda con la de una instalación real.

## 8. Conclusiones

En este apartado se presentan las conclusiones a las que se ha llegado desde el comienzo del planteamiento del proyecto hasta la finalización del mismo.

### 8.1 La importancia y relevancia de las comunicaciones

En la actualidad, existe una gran variedad de dispositivos de distintos fabricantes, los cuales, bien por cuestiones de marketing o de gestión de recursos, implementan sus propias estrategias y buscan soluciones para resolver los problemas y las necesidades con las que tienen que lidiar.

Esto implica que, al intentar realizar la integración de múltiples dispositivos de distintos fabricantes, surjan problemas y dificultades en lo que al tema de las comunicaciones entre los mismos se refiere, por lo que en algunos casos la comunicación directa entre ellos se complica.

Este proyecto tiene la finalidad de transmitir el siguiente mensaje:

**Se ha de recurrir a las funciones, recursos y a las tareas que es capaz de realizar cada uno de los dispositivos de manera individual, y usarlas con la finalidad de compensar las carencias y limitaciones que puedan tener el resto de los sistemas con el objetivo de que todos sean partícipes en la tarea o el objetivo final que se desea alcanzar.**

Es por ello que para poder cumplir con los objetivos que se han propuesto en el proyecto, se haya hecho uso de varios protocolos de comunicaciones. Recordemos que para que exista la comunicación, han de estar presentes un emisor, un receptor, el mensaje y un medio mediante el cual este pueda ser transmitido.

Sin embargo, aun estando presentes todos los elementos anteriores, no sirve de mucho si el receptor, aun recibiendo la información, no sabe cómo interpretarla. Un ejemplo clásico es la situación en la que dos personas se están intentando comunicar, pero no hablan el mismo lenguaje, o si lo hacen, no lo conocen al completo por lo que no consiguen comunicar lo que desean. No por ello desisten en su propósito, e intentan mediante gestos o utilizando otros recursos verbales o no verbales el poder cumplir su objetivo.

Un protocolo de comunicaciones, no es más que un idioma o un conjunto de instrucciones para que dos o más dispositivos puedan entenderse entre ellos. En este proyecto, “el entenderse” hace referencia a que la información transmitida por el



emisor, que puede ser el valor de un sensor o el estado de una variable de proceso interna alojada en la programación del mismo, llegue al dispositivo receptor y que este pueda procesarla haciendo uso de los recursos y las funciones con la que está programado, sin necesidad de que otro elemento intervenga en el proceso.

Por todo lo anterior, he querido integrar en este proyecto mi propio protocolo de comunicaciones, que se ha diseñado teniendo en cuenta las problemáticas antes descritas, y con el objetivo final de crear una herramienta propia para poder integrar en cualquier proyecto personal que quiera desarrollar la posibilidad de comunicar de manera satisfactoria a los dispositivos que lo componen. Sólo han de contar con la electrónica necesaria que permita leer y procesar entradas y actuar sobre sus salidas.

Ha sido un gran placer para mí el poder cumplir con los objetivos planteados en el proyecto, han sido muchos los obstáculos y problemas con los que me he encontrado, para los cuales he tenido que buscar soluciones a las que no ha sido siempre fácil llegar. Es de agradecer el poder contar y disponer de los recursos aportados por una gran comunidad de personas para cumplir las metas que uno se propone, con el objetivo y propósito final de utilizar dichos recursos para construir algo nuevo, que sirva en el futuro de inspiración, ayuda o motivación a otras personas para que puedan desarrollar sus proyectos personales.

## 9. Bibliografía

En este apartado se hace referencia a todos los recursos y fuentes a las que he acudido para poder realizar el proyecto.

- **Comunicación Modbus entre un PLC y Arduino**  
<https://www.electroallweb.com/index.php/2020/03/24/como-establecer-comunicacion-entre-un-arduino-y-un-plc-a-traves-del-protocolo-modbus-enviar-y-recibir-datos/>
- **Comunicación OPC entre WinCC y Robot Studio**  
[https://www.youtube.com/watch?v=rYYVDdOm2QE&ab\\_channel=JuanCarlosMart%C3%ADnCastillo](https://www.youtube.com/watch?v=rYYVDdOm2QE&ab_channel=JuanCarlosMart%C3%ADnCastillo)
- **Utilización del módulo ethernet ENC28J60 con Arduino:**  
[https://naylorlampmechatronics.com/blog/17\\_tutorial-modulo-ethernet-enc28j60-y-arduino.html](https://naylorlampmechatronics.com/blog/17_tutorial-modulo-ethernet-enc28j60-y-arduino.html)
- **Fundamentos teóricos de los protocolos de comunicación**
  - I2C:  
<https://hetpro-store.com/TUTORIALES/i2c/#:~:text=I2C%20es%20un%20puerto%20y,de%20comunicaci%C3%B3n%20SDA%20y%20SCL>.
  - SPI:  
<http://panamahitek.com/como-funciona-el-protocolo-spi/>
  - ESPNOW:  
<https://www.espressif.com/en/products/software/esp-now/overview#:~:text=ESP%20NOW%20is%20yet%20another,often%20deployed%20in%20wireless%20mouses>.  
[https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp\\_now.html](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html)

- Modbus TCP:  
<https://www.rtautomation.com/technologies/modbus-tcpip/>  
[https://www.csimn.com/CSI\\_pages/Modbus101.html](https://www.csimn.com/CSI_pages/Modbus101.html)
- **Datos técnicos de los componentes empleados:**
  - ESP32:  
[http://www.edu.xunta.gal/centros/ieslaxeiro/system/files/ESP-32%20Dev%20Kit%20C%20V2\\_EN.pdf](http://www.edu.xunta.gal/centros/ieslaxeiro/system/files/ESP-32%20Dev%20Kit%20C%20V2_EN.pdf)
  - Arduino UNO:  
<https://store-usa.arduino.cc/products/arduino-uno-rev3>
  - LCD 16x2:  
<https://www.watelectronics.com/lcd-16x2/>
  - Módulo I2C para LCD 16x2  
[http://www.handsontec.com/dataspecs/module/I2C\\_1602\\_LCD.pdf](http://www.handsontec.com/dataspecs/module/I2C_1602_LCD.pdf)
  - Nunchuk:  
<https://bootlin.com/labs/doc/nunchuk.pdf>
  - Módulo ethernet ENC28J60  
<https://www.openimpulse.com/blog/products-page/product-category/enc28j60-ethernet-module/>
  - Batería LIPO:  
<https://www.amazon.es/bater%C3%ADa-Cargador-Eachine-Quadcopter-Repuestos/dp/B07D3864QN>
  - Circuito elevador de tensión para la batería LIPO:  
[https://www.amazon.es/DollaTek-Converter-Circuito-elevable-Ajustable/dp/B081JNVGFZ/ref=sr\\_1\\_2?\\_mk\\_es\\_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crd=LA4UY0EE1ZNT&keywords=elevador+de+voltage](https://www.amazon.es/DollaTek-Converter-Circuito-elevable-Ajustable/dp/B081JNVGFZ/ref=sr_1_2?_mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crd=LA4UY0EE1ZNT&keywords=elevador+de+voltage)

[+3.3-5v&qid=1654585636&srefix=elevador+de+voltaje+3.3-5%2Caps%2C236&sr=8-2](#)

➤ Placa de prototipo:

[https://www.globalspec.com/industrial-directory/breadboard\\_specifications](https://www.globalspec.com/industrial-directory/breadboard_specifications)

➤ Ordenador PC Latitude 3410:

<https://www.dell.com/es-es/shop/port%C3%A1tiles-dell/port%C3%A1til-latitude-3410-para-empresas/spd/latitude-14-3410-laptop>

- **Páginas web consultadas para conocer los precios de los componentes y elaborar el presupuesto**

➤ <https://www.amazon.es/>

➤ [https://www.pccomponentes.com/?gclid=CjwKCAjwy\\_aUBhACEiwA2IHHQPI\\_26WPLE5mNPNLOOfgUYB-W3PtZ3\\_uh7mLGVVH7IxeHU-if14hCbhoCFXEQA\\_vD\\_BwE](https://www.pccomponentes.com/?gclid=CjwKCAjwy_aUBhACEiwA2IHHQPI_26WPLE5mNPNLOOfgUYB-W3PtZ3_uh7mLGVVH7IxeHU-if14hCbhoCFXEQA_vD_BwE)

- **Programa empleado para realizar los esquemas eléctricos**

➤ <https://fritzing.org/>

- **Programa utilizado para elaborar los esquemas funcionales**

➤ <https://app.diagrams.net/>

## Anexos

En esta sección se adjuntarán imágenes que contendrán el código con el que se ha programado cada uno de los dispositivos haciendo uso del lenguaje de programación correspondiente.

### Anexo I. Programación del sistema de control manual inalámbrico

- Se incluyen las librerías y se declaran las variables de proceso

```
//Se incluyen las librerías necesarias para el control del programa
#include <esp_now.h>
#include <WiFi.h>

#include <Wire.h>
#include "Nunchuk.h"

// Dirección MAC del dispositivo receptor
uint8_t broadcastAddress[] = {0x3C, 0x61, 0x05, 0x3E, 0x25, 0x7C};

//Estructura de datos a enviar
typedef struct struct_message {
    int pot_x_raw;
    int pot_y_raw;
    int gir_x_raw;
    int gir_y_raw;
    int c;
    int z;
    int contador_vida;
} struct_message;

// Creación de una instancia de la estructura
struct_message myData;

esp_now_peer_info_t peerInfo;

//Función que envía los datos
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
    Serial.print("\r\nLast Packet Send Status:\t");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}
```

- Función de configuración, se ejecuta sólo una vez

```
void setup() {  
  // Inicialización del monitor serial  
  Serial.begin(9600);  
  
  // Modo de configuración del chip WIFI de la placa  
  WiFi.mode(WIFI_STA);  
  
  // Mensaje de aviso  
  if (esp_now_init() != ESP_OK) {  
    Serial.println("Error en la inicialización");  
    return;  
  }  
  
  // Cuando se inicializa la comunicación, se envían los datos  
  esp_now_register_send_cb(OnDataSent);  
  
  // Se registra el dispositivo receptor  
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);  
  peerInfo.channel = 0;  
  peerInfo.encrypt = false;  
  
  // Mensaje de aviso  
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {  
    Serial.println("Receptor no configurado");  
    return;  
  }  
  Wire.begin();  
  nunchuk_init();  
}
```

- Bucle que ejecuta el programa de forma cíclica

```
void loop() {

//Se envían los datos a la placa receptora al cuando se leen todos los datos del nunchuck
if(nunchuk_read())
{
    enviar_datos();
}
//Incremento del contador de vida
myData.contador_vida++;
if(myData.contador_vida>100) myData.contador_vida=0;

// Envío del mensaje via ESP-NOW
esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));

//Muestra el estado del envío por el monitor serial
if (result == ESP_OK) {
    Serial.println("Enviado");
}
else {
    Serial.println("No enviado");
}

}
```

- Función que guarda los datos

```
void enviar_datos()
{
//Función que lee los datos del nunchuck y los envía a la estructura de comunicaciones
myData.pot_x_raw=(int)nunchuk_joystickX();
myData.pot_y_raw=(int)nunchuk_joystickY();
myData.gir_x_raw=(int)nunchuk_accelX();
myData.gir_y_raw=(int)nunchuk_accelY();
myData.c=(int)nunchuk_buttonC();
myData.z=(int)nunchuk_buttonZ();

}
```

## Anexo II. Programación del sistema de recepción y transmisión de datos (ESP32)

- Se incluyen las librerías y se declaran las variables de proceso

```
////////*****EMISOR DE DATOS RECOGIDOS POR EL JOYSTICK VIA ESPNOW A LA PLACA ARDUINO*****////////
//Librerías para la comunicación ESPNOW
#include <esp_now.h>
#include <WiFi.h>

////////-----VARIABLES DE COMUNICACION J2WC-----////////

//Direcciones de los pines digitales de E/S
int pinDatos = 19;
int pinRelej = 18;
int pinLedDatos = 17;
int pinLedRelej = 16;
int pinBoton = 34;
bool estado_boton;

//Variables necesarias para el control del programa
unsigned long int tA_ciclo, tC_ciclo, tiempo_ciclo; //Para el cálculo de los tiempos de ciclo
unsigned long int tA_relej, tC_relej, OUT_relej;
bool salida_conversion[11];
bool informacion_salida[8][11];
int informacion_entrada[8];
int stepindex;
long int num_ciclos_relej = 0;
int contador_n_dato, contador_n_bit;
bool first_scan, check_relej_inicio, check_flanco_temporizador;

//Control sobre las líneas de datos
bool cmd_pin_datos, cmd_pin_relej;
bool cmd_IN_relej, cmd_forzar_relej;
int cmd_PT_relej;
int offset;
int tpc;
int pot;

////////-----VARIABLES DE COMUNICACION ESPNOW-----////////
typedef struct struct_message {
    int pot_x_raw;
    int pot_y_raw;
    int gir_x_raw;
    int gir_y_raw;
    int c;
    int z;
    int contador_vida;
} struct_message;
struct_message myData;
```



- Declaración de las funciones que se usarán en el programa

```
////////-----DECLARACIÓN FUNCIONES DEL PROGRAMA PARA COMUNICACIÓN J2WC-----////////

//Controla el estado de una salida que oscila entre 0 y 1 ciclicamente
void senal_reloj(bool IN, int PT);

//Forma la trama de bits a enviar correspondiente a uno de los sensores
void transformar_entero_a_byte(int n, int id, bool salida_conversion[]);

//Forma la trama de salida completa con todos los datos de los sensores
void formar_trama_salida(int num_datos);

//Asocia los valores de los sensores a una posición del vector para facilitar el manejo de los mismos
void datos_a_vector(int pot_x, int pot_y, int gir_x, int gir_y, int c, int z, int tciclo_EM, int tciclo_REC);

//Funciones de depuración para visualizar el valor de las tramas
void mostrar_datos_byte(int tam, bool entrada[]);
void mostrar_datos_trama_salida();

//Controla los pines de salida según los comandos generados por las funciones de control del programa
void escribir_pines(bool IN_reloj, bool IN_datos);

//Envía los datos a los sensores a la placa arduino receptora mediante el protocolo JW2
//Devuelve un 1 cuando termina de enviar toda la información
bool enviar_datos();
```

- Función de configuración, se ejecuta sólo una vez

```
void setup() {

    Serial.begin(9600); //Iniciación de la comunicación serial con el PC para diagnóstico y depuración
    //Declaración de los pines E/S
    pinMode(pinDatos, OUTPUT);
    pinMode(pinRelej, OUTPUT);
    pinMode(pinLedDatos, OUTPUT);
    pinMode(pinLedRelej, OUTPUT);
    pinMode(33, INPUT);
    pinMode(pinBoton, INPUT_PULLUP);

    //Setup para la comunicación ESPNOW
    //Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }
    esp_now_register_recv_cb(OnDataRecv);

    delay(3000);
}
```

- Bucle que ejecuta el programa de forma cíclica

```
void loop() {  
  
    //Llamada a la función reloj  
    senal_reloj(cmd_IN_reloj, cmd_PT_reloj);  
    cmd_pin_reloj = OUT_reloj; //El estado de la salida digital dependerá del valor de salida de la señal de reloj  
  
    //Llamada a la rutina de envío de datos  
    enviar_datos();  
  
    //Actuación sobre las salidas digitales en función de los comandos calculados en las funciones anteriores  
    escribir_pines( (cmd_pin_reloj or cmd_forzar_reloj), cmd_pin_datos);  
  
    //Funciones de depuración  
    Serial.println(stepindex);  
    //mostrar_datos_trama_salida();  
  
}
```

- Función que envía los datos (parte 1)

```
bool enviar_datos()  
{  
    switch (stepindex)  
    {  
        case 0:  
            // Variables para el cálculo del tiempo de latencia  
            tpc = int((tC_ciclo - tA_ciclo));  
            tA_ciclo = millis();  
            //Lectura del valor del potenciómetro para regular la velocidad de comunicación  
            int pot=map(analogRead(32),0,4000,0,100);  
  
            //Recoje el valor de los sensores y los introduce en el vector correspondiente  
            datos_a_vector(myData.pot_x_raw, myData.pot_y_raw, myData.gir_x_raw/10, myData.gir_y_raw/10, myData.c, myData.z, tpc/100, myData.contador_vida);  
            //Forma la trama de bits de salida  
            formar_trama_salida(8);  
            cmd_IN_reloj = 0; //Reseteo del reloj  
            tC_reloj = millis();  
  
            stepindex += 10; //Incremento del paso  
            break;  
        case 10:  
            //El reloj permanece 2s encendido y 2s apagado  
            cmd_PT_reloj = 10;  
            cmd_IN_reloj = 1;  
            if (OUT_reloj)  
            {  
                pinMode(pinDatos, OUTPUT);  
                check_reloj_inicio = 1;  
            }  
            else if (!OUT_reloj and check_reloj_inicio)  
            {  
                stepindex += 10; //Incremento del paso  
                cmd_IN_reloj = 0;  
                check_reloj_inicio = 0;  
            }  
            break;  
    }  
}
```

- Función que envía los datos (parte 2)

```
case 20:
    //Envío de los datos
    //Activación del reloj
    cmd_PT_reloj = 1;
    cmd_IN_reloj = 1;
    if (contador_n_bit > 10)
    {
        //Cuando se envían los 11 bits de datos correspondientes a el valor de un sensor,
        //se resetean los contadores, para poder leer el siguiente.
        contador_n_dato++;
        contador_n_bit = 0;
    }
    //Al terminar de enviar los datos recibidos por los sensores, vuelve a comenzar el ciclo
    //empezando por el primero
    if (contador_n_dato > 7)
    {
        //La secuencia ha terminado de enviarse
        contador_n_dato = 0;
        cmd_IN_reloj = 0;
        stepindex += 10; //Incremento del paso
        cmd_pin_datos=0;
    }
    if (OUT_reloj)
    {
        //Cuando la señal de reloj se activa, se incrementa en 1 el contador de bits
        if (!check_reloj_inicio)
        {
            contador_n_bit++;
            check_reloj_inicio = 1;
        }
    }
    else
    {
        //Aprovechando que la señal de reloj está a apagada, se prepara la línea de datos para enviar el valor
        //del siguiente bit
        cmd_pin_datos = informacion_salida[contador_n_dato][contador_n_bit];
        check_reloj_inicio = 0;
    }
    break;
```

- Función que envía los datos (parte 3)

```
case 30:
    cmd_IN_reloj = 0;
    check_reloj_inicio = 0;
    pinMode(pinDatos, INPUT); //Preparamos al pin para leer una señal del receptor
    stepindex += 5; //Se salta de paso para no configurar el pin como entrada continuamente
    break;
case 35:
    //Se espera a que el receptor le avise que ha terminado con su procesamiento
    pinMode(pinDatos, INPUT); //Preparamos al pin para leer una señal del receptor
    if (digitalRead(pinDatos))
    {
        tC_ciclo = millis();
        stepindex = 0;
    }
    break;
}
```

- Funciones secundarias del programa llamadas desde el bucle principal (parte 1)

```
void transformar_entero_a_byte(int n, int id, bool salida_conversion[])
{
    //Forma la parte entera
    if (abs(n) > 127)
    {
        n = 127;
    }
    int divisor = n;
    for (int j = 0; j < 7; j++)
    {
        salida_conversion[10 - j] = divisor % 2;
        divisor = divisor / 2;
    }
    //Coloca el signo (0 positivo, 1 negativo)
    salida_conversion[3] = (n < 0);

    //Forma los bits que representan el ID del dato (del 0 al 7)
    divisor = id;
    for (int j = 0; j < 3; j++)
    {
        salida_conversion[2 - j] = divisor % 2;
        divisor = divisor / 2;
    }
}

void mostrar_datos_byte(int tam, bool entrada[])
{
    //Muestra los datos de los bits individuales de uno de los sensores
    for (int j = 0; j < tam; j++)
    {
        Serial.print(entrada[j]);
    }
    Serial.println();
}
```

- Funciones secundarias del programa llamadas desde el bucle principal (parte 2)

```
void formar_trama_salida(int num_datos)
{
    //Forma la trama de salida, recorriendo los vectores formados por los bits individuales
    //de cada uno de los datos
    for (int i = 0; i < num_datos; i++)
    {
        transformar_entero_a_byte(informacion_entrada[i], i, salida_conversion);
        for (int j = 0; j < 11; j++)
        {
            informacion_salida[i][j] = salida_conversion[j];
        }
    }
}

void datos_a_vector(int pot_x, int pot_y, int gir_x, int gir_y, int c, int z, int tciclo_EM, int tciclo_REC)
{
    //Asigna los valores de los sensores a su correspondiente posición del vector
    informacion_entrada[0] = pot_x;
    informacion_entrada[1] = pot_y;
    informacion_entrada[2] = gir_x;
    informacion_entrada[3] = gir_y;
    informacion_entrada[4] = c;
    informacion_entrada[5] = z;
    informacion_entrada[6] = tciclo_EM;
    informacion_entrada[7] = tciclo_REC;

    for(int j=0;j<8;j++)
    {
        Serial.print(j);
        Serial.print("-");
        Serial.println(informacion_entrada[j]);
    }
}
```

- Funciones secundarias del programa llamadas desde el bucle principal (parte 3)

```
void mostrar_datos_trama_salida()
{
    //Muestra los datos completos de toda la trama de salida
    if (!first_scan)
    {

        for (int i = 0; i < 8; i++)
        {
            Serial.print("D");
            Serial.print(i);
            Serial.print(":");
            Serial.print(informacion_entrada[i]);
            Serial.print("-");
            for (int j = 0; j < 11; j++)
            {
                Serial.print(informacion_salida[i][j]);
            }
            Serial.println();
        }
    }
    first_scan = 1;
}

void senal_reloj(bool IN, int PT)
{
    //Función empleada para activar y desactivar una señal periódicamente
    tA_reloj = millis();

    if (IN)
    {
        if (tA_reloj - tC_reloj >= PT)
        {
            OUT_reloj = !OUT_reloj; //Invierte el valor de la salida cada vez que se cumple el tiempo
            tC_reloj = millis();
            num_ciclos_reloj++;
        }
    }
    else
    {
        OUT_reloj = 0;
    }
}
```

- Funciones secundarias del programa llamadas desde el bucle principal (parte 4)

```
int temporizador(bool IN)
{
    //Función empleada para calcular el tiempo transcurrido desde que se activa la señal de entrada
    tA_ciclo = millis();

    if (IN)
    {
        if (!check_flanco_temporizador)
        {
            tC_ciclo = millis();
            check_flanco_temporizador = 1;
        }
    }
    else
    {
        check_flanco_temporizador = 0;
    }

    return ( int(tA_ciclo - tC_ciclo) / 100);
}
```

## Anexo III. Programación del sistema de recepción y transmisión de datos (Arduino UNO)

- Se incluyen las librerías y se declaran las variables de proceso

```

////////*****RECEPTOR DE DATOS RECOGIDOS POR EL ESP32 *****//////////
////////*****ENVÍA LOS DATOS POR MODBUS AL PLC SIMULADO*****//////////

//Librerías necesarias para la comunicación Modbus
#include <EtherCard.h>
#include <Modbus.h>
#include <ModbusIP_ENC28J60.h>
//Declaración de registros para Modbus
const int POT_X_CAL = 50;
const int POT_Y_CAL = 51;
const int G_X_cal = 52;
const int G_Y_cal = 53;
const int BOT_C = 54;
const int BOT_Z = 55;
const int LATENCIA_J2W = 56;
const int LATENCIA_ESPNOW = 57;
ModbusIP mb;//Objeto Modbus

////////-----VARIABLES DE COMUNICACION JW2-----////////
//Direcciones de los pines digitales de E/S
int pinDatos = 2;
int pinReloj = 3;
int pinLedDatos = 5;
int pinLedReloj = 13;
int pinBoton = 34;
bool estado_boton;
//Variables necesarias para el control del programa
unsigned long int tA_ciclo, tC_ciclo, tiempo_ciclo; //Para el cálculo de los tiempos de ciclo
unsigned long int tA_reloj, tC_reloj, OUT_reloj;
bool salida_conversion[11];
int datos_sensores_entrada[8];
int stepindex;
long int num_ciclos_reloj = 0;
int contador_n_dato, contador_n_bit;
bool first_scan, check_reloj_inicio;
bool entrada_bits[11];
int datos_sensores[8];
//Control sobre las líneas de datos
bool cmd_pin_datos, cmd_pin_reloj;
bool cmd_IN_reloj;
int cmd_PT_reloj;
int offset;
//Lectura de las líneas de datos
bool sts_pin_datos, sts_pin_reloj, sts_flanco_reloj;

```



- Declaración de las funciones que se usarán en el programa

```
/////////-----DECLARACIÓN FUNCIONES DEL PROGRAMA-----/////////

//Funciones de depuración para visualizar el valor de las tramas
void mostrar_datos_byte(int tam, bool entrada[]);
void mostrar_datos_trama_entrada(int IN_trama[]);

//Lee los pines de entrada digitales
void leer_pines();

//Recibe los datos de los sensores a la placa ESP32 EMISORA mediante el protocolo JW2
void recibir_datos();

//Envía los datos recibidos por el ESP32 via J2W al PLC simulado vía Modbus
void enviar_datos_modbus(int entrada[]);

int pot_x, pot_y, gir_x, gir_y, c, z, D_latencia_J2W, D_latencia_ESPNOW;
bool check_flanco_reloj, check_flanco_temporizador;

//Asocia los bits de entrada al sensor correspondiente con su valor y signo
void transformar_byte_a_entero(bool entrada[]);
//Detecta un flanco positivo en la señal de reloj
bool flanco_reloj(bool IN_sts_reloj);

//Controla los pines de salida correspondientes
void escribir_pines(bool cmd_pin_reloj, bool cmd_pin_datos);
```

- Función de configuración, se ejecuta sólo una vez

```
void setup() {

    //Se le asigna una dirección MAC y una IP al Arduino para la comunicación por Modbus
    //dirección MAC
    byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
    //dirección IP
    byte ip [] = {192, 168, 1, 25};

    Serial.begin(9600); //Iniciación de la comunicación serial con el PC para diagnóstico y depuración
    //Declaración de los pines E/S
    pinMode(pinDatos, INPUT);
    pinMode(pinReloj, INPUT);
    pinMode(pinLedDatos, OUTPUT);
    pinMode(pinLedReloj, OUTPUT);
    //pinMode(pinBoton, INPUT_PULLUP);

    //Se añaden los registros
    mb.config (mac, ip); // se confirma las direcciones
    mb.addIreg (POT_X_CAL);
    mb.addIreg (POT_Y_CAL);
    mb.addIreg (G_X_cal);
    mb.addIreg (G_Y_cal);
    mb.addIreg (BOT_C);
    mb.addIreg (BOT_Z);
    mb.addIreg (LATENCIA_J2W);
    mb.addIreg (LATENCIA_ESPNOW);

}
```

- Bucle que ejecuta el programa de forma cíclica

```
void loop() {

    mb.task(); //Actualiza continuamente la conexión Modbus
    //enviar_datos_modbus();
    leer_pines(); //Lee los pines de entrada físicos
    sts_flanco_reloj = flanco_reloj(sts_pin_reloj);
    recibir_datos();

    //Llamada a funciones de depuración
    //Serial.println("a");
    /*
        Serial.print("Bit:");
        Serial.print(contador_n_bit);
        Serial.print("N:");
        Serial.print(contador_n_dato);
        Serial.print(" St:");
        Serial.println(stepindex);
    */
}
```

- Función que recibe los datos (parte 1)

```
void recibir_datos()
{
    switch (stepindex)
    {
        case 0:
            // Se espera a que la señal de reloj esté apagada
            if (!sts_pin_reloj) stepindex += 2; //Incremento del paso
            break;
        case 2:
            // Se espera a que la señal de reloj esté activa
            if (sts_pin_reloj)
            {
                digitalWrite(pinDatos, 0);
                pinMode(pinDatos, INPUT);
                stepindex += 2; //Incremento del paso
            }
            break;
        case 4:
            // Se detecta un flanco negativo en la señal de reloj=> Empieza a leer los datos de entrada
            if (!sts_pin_reloj) stepindex = 10; //Incremento del paso
            break;
    }
}
```

- Función que recibe los datos (parte 2)

```
case 10:
    //Recepción de datos

    //Cuando se reciben los 11 bits de datos correspondientes a el valor de un sensor,
    //se resetean los contadores, para poder leer el siguiente.

    if (sts_flanco_reloj)
    {
        //Cuando se detecta la activación de la señal de reloj, se guardan el estado de la entrada
        //que transmite la información
        entrada_bits[contador_n_bit] = sts_pin_datos;

        //Cuando la señal de reloj se activa, se incrementa en 1 el contador de bits
        contador_n_bit++;
    }
    if (contador_n_bit > 10)
    {
        //Envia los 11 bits recibidos a la función para procesarlos
        procesar_sensor_entrada(entrada_bits);
        contador_n_bit = 0;
        contador_n_dato++;
    }
    if (contador_n_dato > 7)
    {
        //Todos los datos de los sensores recibidos=>Se resetean las variables
        stepindex += 10; //Incremento del paso
        contador_n_dato = 0;
        contador_n_bit = 0;
    }
    break;
}
```

- Función que recibe los datos (parte 3)

```

case 20:
    mostrar_datos_trama_entrada(datos_sensores_entrada); //Muestra los datos de los sensores
    vector_a_datos();
    enviar_datos_modbus();
    pinMode(pinDatos, OUTPUT); //Preparamos al pin para enviar una señal al emisor
    //delay(10);
    stepindex += 5; //Se salta de paso para no configurar el pin como entrada continuamente

    break;

case 25:
    digitalWrite(pinDatos, 1); //Se escribe en el pin para indicar que se ha terminado de procesar los datos
    //delay(10);

    //El emisor nos confirma que ha recibido el mensaje
    //digitalWrite(pinDatos,0); //Se deja de mandar la confirmacion
    stepindex = 0;

    break;
}
}
}

```

- Funciones secundarias del programa llamadas desde el bucle principal (parte 1)

```

void leer_pines()
{
    //Lee los valores recibidos en los pines de entrada
    sts_pin_datos = digitalRead(pinDatos);
    sts_pin_reloj = digitalRead(pinReleoj);
}

bool flanco_reloj(bool IN_sts_reloj)
{
    if (IN_sts_reloj and !check_flanco_reloj)
    {
        check_flanco_reloj = 1;
        return (1);
    }
    else if (IN_sts_reloj and check_flanco_reloj)
    {
        return (0);
    }
    else if (!IN_sts_reloj)
    {
        check_flanco_reloj = 0;
        return (0);
    }
}

void procesar_sensor_entrada(bool entrada[])
{
    //Recibe los bits provenientes de un sensor y los almacena en la posición correspondiente según el ID que tenga
    transformar_byte_a_entero(entrada);
    datos_sensores_entrada[OUT_id_sensor] = OUT_valorDecimal;
}

```

- Funciones secundarias del programa llamadas desde el bucle principal (parte 2)

```
void vector_a_datos()
{
    //Asigna los valores de los sensores a su correspondiente posición del vector
    pot_x = datos_sensores_entrada[0];
    pot_y = datos_sensores_entrada[1];
    gir_x = datos_sensores_entrada[2];
    gir_y = datos_sensores_entrada[3];
    c = datos_sensores_entrada[4];
    z = datos_sensores_entrada[5];
    D_latencia_J2W = datos_sensores_entrada[6];
    D_latencia_ESPNOW = datos_sensores_entrada[7];
}

void transformar_byte_a_entero(bool entrada[])
{
    //Cálculo de la parte entera
    OUT_valorDecimal = 0;
    for (byte i = 4; i < 11; i++)
    {
        OUT_valorDecimal *= 2;
        OUT_valorDecimal += entrada[i] ;
    }
    //Asigna el signo correspondiente
    if (entrada[3])
    {
        OUT_valorDecimal = OUT_valorDecimal * (-1);
    }

    //Calcula el ID del dato
    OUT_id_sensor = 0;
    for (byte i = 0; i < 3; i++)
    {
        OUT_id_sensor *= 2;
        OUT_id_sensor += entrada[i] ;
    }
    //Serial.println(OUT_valorDecimal);
}
```

- Funciones secundarias del programa llamadas desde el bucle principal (parte 3)

```
void mostrar_datos_byte(int tam, bool entrada[])
{
    //Muestra los datos de los bits individuales de uno de los sensores
    for (int j = 0; j < tam; j++)
    {
        Serial.print(entrada[j]);
    }
    Serial.println();
}

void mostrar_datos_trama_entrada(int IN_trama[])
{
    //Muestra los datos completos de toda la trama de entrada
    for (int i = 0; i < 8; i++)
    {
        Serial.print("D");
        Serial.print(i);
        Serial.print("->");
        Serial.print(IN_trama[i]);
        Serial.println();
    }
    //Serial.print(tC_ciclo-tA_ciclo);
}

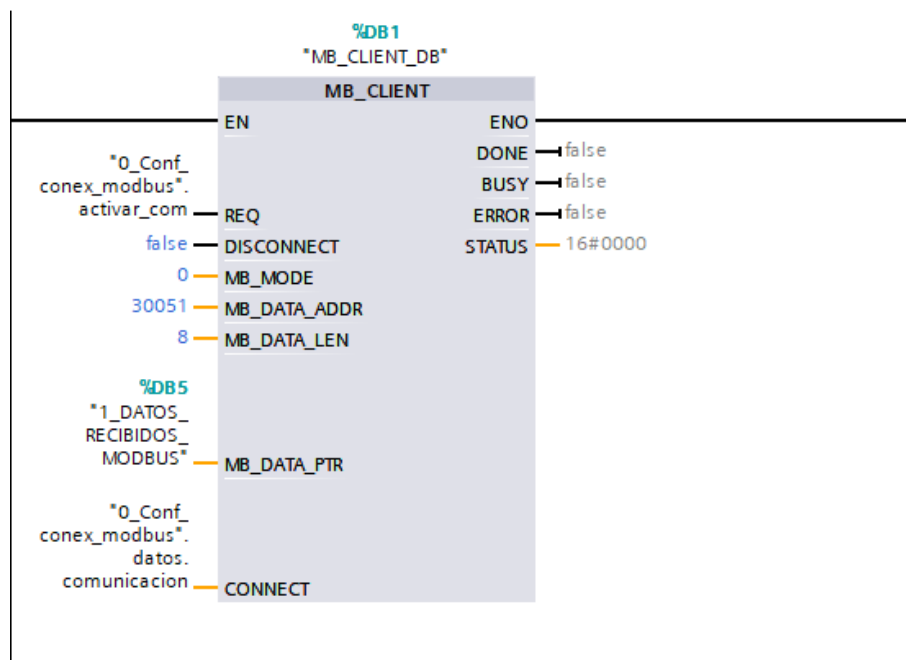
void escribir_pines(bool IN_reloj, bool IN_datos)
{
    //Escribe los valores de entrada en los pines correspondientes
    digitalWrite(pinLedDatos, IN_datos);
    digitalWrite(pinLedRelej, IN_reloj);
}

void enviar_datos_modbus()
{
    //Se escriben los datos en los registros Modbus
    mb.Ireg(POT_X_CAL, pot_x);
    mb.Ireg(POT_Y_CAL, pot_y);
    mb.Ireg(G_X_cal, gir_x);
    mb.Ireg(G_Y_cal, gir_y);
    mb.Ireg(BOT_C, c);
    mb.Ireg(BOT_Z, z);
    mb.Ireg(LATENCIA_J2W, D_latencia_J2W);
    mb.Ireg(LATENCIA_ESPNOW, D_latencia_ESPNOW);
}
```

## Anexo IV. Programación en TIA PORTAL

### CONFIGURACIÓN Y PROGRAMACIÓN DE LA COMUNICACIÓN MODBUS:

- Llamada a la función Modbus para establecer la comunicación con el servidor



- DB de configuración de la conexión

0_Conf_conex_modbus			
	Nombre	Tipo de datos	Valor de arranq...
1	Static		
2	datos	Struct	
3	comunicacion	TCON_IP_v4	
4	InterfaceId	HW_ANY	64
5	ID	CONN_OUC	16#3
6	ConnectionType	Byte	16#0B
7	ActiveEstablish...	Bool	TRUE
8	RemoteAddress	IP_V4	
9	ADDR	Array[1..4] of Byte	
10	ADDR[1]	Byte	192
11	ADDR[2]	Byte	168
12	ADDR[3]	Byte	1
13	ADDR[4]	Byte	25
14	RemotePort	UInt	502
15	LocalPort	UInt	0
16	VALOR_POT	Word	16#0
17	activar_com	Bool	1

- DB en el que se guardan los datos recibidos por Modbus

1_DATOS_RECIBIDOS_MODBUS			
	Nombre	Tipo de datos	Offset
1	Static		
2	POT_X_CAL	Int	0.0
3	POT_Y_CAL	Int	2.0
4	G_X_CAL	Int	4.0
5	G_Y_CAL	Int	6.0
6	BOT_C	Int	8.0
7	BOT_Z	Int	10.0
8	Latencia_J2W	Int	12.0
9	Latencia_ESPNOW	Int	14.0



## PROGRAMACIÓN DE LA LIBRERÍA PARA EL CONTROL DE LOS EJES

- Variables utilizadas

1_OBJETO_EJE					
	Nombre	Tipo de datos	Valor predet.	Remanencia	A
1	▼ Input				
2	IN_Activar	Bool	false	No remane...	
3	IN_Permitido_izquierda	Bool	false	No remane...	
4	IN_Permitido_derecha	Bool	false	No remane...	
5	IN_Max_angulo	Real	0.0	No remane...	
6	IN_Min_angulo	Real	0.0	No remane...	
7	IN_Tiempo_paso	Int	0	No remane...	
8	IN_Angulo_offset	Real	0.0	No remane...	
9	▼ Output				
10	<Agregar>				
11	▼ InOut				
12	<Agregar>				
13	▼ Static				
14	OUT_Angulo_Eje	Int	0	No remane...	
15	Proc_Permitido_Derecha	Bool	false	No remane...	
16	Proc_permitido_Izquierda	Bool	false	No remane...	
17	angulo_ant	Real	0.0	No remane...	
18	Angulo_actual	Real	0.0	No remane...	
19	temp_angulo	TON_TIME		No remane...	
20	in_temp	Bool	false	No remane...	
21	out_temp	Bool	false	No remane...	
22	pt_temp	Time	T#0ms	No remane...	
23	flanco_activacion	R_TRIG			
24	in_flanco	Bool	false	No remane...	
25	out_flanco	Bool	false	No remane...	
26	cmd_invertir_giro	Bool	false	No remane...	

- Programación de la función

```

2  #pt_temp:=INT_TO_TIME(#IN_Tiempo_paso);
3  #in_flanco := #IN_Activar;
4  IF #temp_angulo(IN := #in_temp,
5      PT := #pt_temp,
6      Q => #out_temp);
7  IF #flanco_activacion(CLK := #in_flanco,
8      Q => #out_flanco);
9  //GESTION TEMP ÁNGULO
10 IF (NOT #IN_Activar) THEN
11     #angulo_ant := #Angulo_actual;
12     #in_temp := 0;
13 ELSE
14     #in_temp := 1;
15 END_IF;
16 IF (#out_flanco) THEN
17     #Angulo_actual := #angulo_ant;
18 END_IF;
19 //GESTIÓN DE INVERSIÓN DE MOVIMIENTO EN CASO DE QUE SEA NECESARIO
20 IF (#cmd_invertir_giro) THEN
21     #Proc_Permiso_Derecha := NOT (#IN_Permiso_derecha);
22     #Proc_permiso_Izquierda := NOT (#IN_Permiso_izquierda);
23 ELSE
24     #Proc_Permiso_Derecha := (#IN_Permiso_derecha);
25     #Proc_permiso_Izquierda := (#IN_Permiso_izquierda);
26
27 END_IF;
28 //MOVIMIENTO HACIA DERECHA O IZQUIERDA
29 IF (#out_temp AND #IN_Activar) THEN
30     #in_temp := 0;
31     IF (#Proc_Permiso_Derecha = 0 AND #Proc_permiso_Izquierda = 1)
32     THEN
33         #Angulo_actual := #Angulo_actual + #IN_Angulo_offset;
34         IF (#Angulo_actual > #IN_Max_angulo) THEN
35             #Angulo_actual := #IN_Max_angulo;
36         END_IF;
37     ELSIF(#Proc_Permiso_Derecha = 1 AND #Proc_permiso_Izquierda = 0)
38     THEN
39         #Angulo_actual := #Angulo_actual - #IN_Angulo_offset;
40         IF (#Angulo_actual < #IN_Min_angulo) THEN
41             #Angulo_actual := #IN_Min_angulo;
42         END_IF;
43     END_IF;
44 END_IF;
45 #OUT_Angulo_Eje := #Angulo_actual*100; //ESCRITURA DEL ÁNGULO EN EL ROBOT
46







































```

## PROGRAMACIÓN DE LA RUTINA DE PROCESAMIENTO DE DATOS

- Variables utilizadas (parte 1)

2_PROCESAMIENTO_DATOS			
	Nombre	Tipo de datos	Valor predet.
7	▼ Static		
8	IN_1_POT_X	Int	0
9	IN_2_POT_Y	Int	0
10	IN_3_GIR_X	Int	0
11	IN_4_GIR_Y	Int	0
12	IN_5_BOT_C	Int	0
13	IN_6_BOT_Z	Int	0
14	▶ EJE_1	"1_OBJETO_EJE"	
15	▶ EJE_2	"1_OBJETO_EJE"	
16	▶ EJE_3	"1_OBJETO_EJE"	
17	▶ EJE_4	"1_OBJETO_EJE"	
18	▶ EJE_5	"1_OBJETO_EJE"	
19	▶ EJE_6	"1_OBJETO_EJE"	
20	▶ Flanco_cambio_ejes	R_TRIG	
21	Sts_modos_control	Bool	false
22	IN_flanco_cambio_ejes	Bool	false
23	Q_flanco_cambio_ejes	Bool	false
24	cfg_margen_pot_x	Int	10
25	cfg_margen_pot_y	Int	10
26	cfg_margen_gir_x	Int	50
27	cfg_margen_gir_y	Int	0
28	cfg_max_pot_x	Int	0
29	cfg_max_pot_y	Int	0
30	cfg_max_gir_x	Int	0
31	cfg_max_gir_y	Int	0
32	cfg_neutro_pot_x	Int	0
33	cfg_neutro_pot_y	Int	0
34	cfg_neutro_gir_x	Int	0
35	cfg_neutro_gir_y	Int	0
36	cfg_max_vel	Real	2.0
37	cfg_min_vel	Real	0.0
38	cfg_factor_x	Real	0.0
39	cfg_factor_y	Real	0.0
40	sts_vel_x	Real	0.0
41	sts_vel_y	Real	0.0
42	sts_derecha_x	Bool	false
43	sts_izquierda_x	Bool	false
44	sts_derecha_y	Bool	false
45	sts_izquierda_y	Bool	false

- Variables utilizadas (parte 2)

2_PROCESAMIENTO_DATOS				
		Nombre	Tipo de datos	Valor predet.
49		GLOBAL_angulo_min	Real	-90.0
50		GLOBAL_angulo_offs...	Real	0.0
51		GLOBAL_angulo_offs...	Real	0.0
52		OPC_OUT_POS_E1	Int	0
53		OPC_OUT_POS_E2	Int	0
54		OPC_OUT_POS_E3	Int	0
55		OPC_OUT_POS_E4	Int	0
56		OPC_OUT_POS_E5	Int	0
57		OPC_OUT_POS_E6	Int	0
58		OPC_OUT_ACTIVAR_V...	Int	0
59		HMI_OUT_P_E1	Real	0.0
60		HMI_OUT_P_E2	Real	0.0
61		HMI_OUT_P_E3	Real	0.0
62		HMI_OUT_P_E4	Real	0.0
63		HMI_OUT_P_E5	Real	0.0
64		HMI_OUT_P_E6	Real	0.0
65		HMI_OUT_VEL_X	Real	0.0
66		HMI_OUT_VEL_Y	Real	0.0
67		HMI_OUT_GRUPO_1	Bool	false
68		HMI_OUT_GRUPO_2	Bool	false
69		HMI_OUT_GRUPO_3	Bool	false
70		HMI_OUT_VENTOSA	Bool	false
71		HMI_IN_VEL_MAX	Real	0.0
72		HMI_IN_VEL_PASO	Int	0
73		HMI_IN_INICIO	Bool	false
74		HMI_IN_POS_EJE_X	Real	0.0
75		HMI_IN_POS_EJE_Y	Real	0.0
76		HMI_IN_POS_EJE_Z	Real	0.0
77		HMI_OUT_DISPARO	Bool	false
78		HMI_Actualizacion_Va...	Bool	false
79		Contador_vida_IN	Int	0
80		Contador_vida_OUT	Int	0
81		TON_Latencia_Conexi...	TON_TIME	
82		IN_TON_Latencia	Bool	false
83		OUT_TON_Latencia	Bool	false
84		ET_TON_Latencia	Tíme	T#0ms
85		Sts_contador_vida_OK	Bool	false
86		Sts_tiempo_latencia	Int	0

- Programación de la función (lectura de entradas y ajuste de parámetros)

```

1  //ENTRADAS
2  #IN_1_POT_X := "1_DATOS_RECIBIDOS_MODBUS".POT_X_CAL;
3  #IN_2_POT_Y := "1_DATOS_RECIBIDOS_MODBUS".POT_Y_CAL;
4  #IN_3_GIR_X := "1_DATOS_RECIBIDOS_MODBUS".G_X_CAL;
5  #IN_4_GIR_Y := "1_DATOS_RECIBIDOS_MODBUS".G_Y_CAL;
6  #IN_5_BOT_C := "1_DATOS_RECIBIDOS_MODBUS".BOT_C;
7  #IN_6_BOT_Z := "1_DATOS_RECIBIDOS_MODBUS".BOT_Z;
8
9  //LECTURA VALORES HMI
10 #cfg_max_vel := #HMI_IN_VEL_MAX;
11 #GLOBAL_tiempo_paso := #HMI_IN_VEL_PASO;
12
13 //CONFIGURACION PARAMETROS JOYSTICK
14 #cfg_margen_pot_x := 10;
15 #cfg_margen_pot_y := 10;
16 #cfg_max_pot_x := 100;
17 #cfg_max_pot_y := 100;
18 #cfg_neutro_pot_x := 0;
19 #cfg_neutro_pot_y := 0;
20
21 //CONFIGURACION PARAMETROS GIROSCOPIO
22 #cfg_margen_gir_x := 60;
23 #cfg_margen_gir_y := 100;
24 #cfg_max_gir_x := 200;
25 #cfg_max_gir_y := 150;
26 #cfg_neutro_gir_x := 40;
27 #cfg_neutro_gir_y := 0;
28
29 //#cfg_max_vel := 4;
30

```

- Programación de la función (parte 1)

```

35 REGION LATENCIA
36 //CÁLCULO DE LA LATENCIA DE LA CONEXIÓN ENTRE EL PLC Y ROBOT STUDIO
37 #TON_Latencia_Conexion_RS(IN := #IN_TON_Latencia,
38                             PT := T#100M,
39                             ET => #ET_TON_Latencia,
40                             Q => #OUT_TON_Latencia);
41
42 IF (#Contador_vida_IN = #Contador_vida_OUT AND (NOT #Sts_contador_vida_OK))
43 THEN
44     #Sts_contador_vida_OK := 1;
45     #Contador_vida_OUT += 1;
46     #IN_TON_Latencia := 1;
47 ELSIF (#Contador_vida_IN = #Contador_vida_OUT AND #Sts_contador_vida_OK)
48 THEN
49     #Sts_contador_vida_OK := 0;
50     #Sts_tiempo_latencia := TIME_TO_INT(IN := #ET_TON_Latencia);
51     #IN_TON_Latencia := 0;
52     RESET_TIMER(#TON_Latencia_Conexion_RS);
53
54 END_IF;
55 IF (#Contador_vida_OUT > 30000)
56 THEN
57     #Contador_vida_OUT := 0;
58 END_IF;
59 END_REGION
60
61 //ENVIAR BRAZO A POSICION INICIAL
62 IF (#HMI_IN_INICIO) THEN
63     #EJE_1.Angulo_actual := 0;
64     #EJE_2.Angulo_actual := 0;
65     #EJE_3.Angulo_actual := 0;
66     #EJE_4.Angulo_actual := 0;
67     #EJE_5.Angulo_actual := 0;
68     #EJE_6.Angulo_actual := 0;
69 END_IF;

```

- Programación de la función (parte 2)

```

88 //CONTROL SENTIDO DE GIRO DE LOS EJES
89 IF (#Sts_modo_control = 0)
90 THEN
91     REGION CONTROL POR JOYSTICK
92     //SE CALCULA EL SENTIDO DE GIRO
93     IF (#IN_1_POT_X > (#cfg_neutro_pot_x + #cfg_margen_pot_x)) THEN
94         #sts_izquierda_x := 0;
95         #sts_derecha_x := 1;
96     ELSIF (#IN_1_POT_X < (#cfg_neutro_pot_x - #cfg_margen_pot_x)) THEN
97
98         #sts_izquierda_x := 1;
99         #sts_derecha_x := 0;
100     ELSE
101         #sts_izquierda_x := 0;
102         #sts_derecha_x := 0;
103     END_IF;
104     IF (#IN_2_POT_Y > (#cfg_neutro_pot_y + #cfg_margen_pot_y)) THEN
105         #sts_izquierda_y := 0;
106         #sts_derecha_y := 1;
107     ELSIF (#IN_2_POT_Y < (#cfg_neutro_pot_y - #cfg_margen_pot_y)) THEN
108         #sts_izquierda_y := 1;
109         #sts_derecha_y := 0;
110     ELSE
111         #sts_izquierda_y := 0;
112         #sts_derecha_y := 0;
113     END_IF;
114     //CÁLCULO DE VELOCIDAD DE MOVIMIENTO EN GRADOS ANGULARES/PASO (°/P)
115     //HACIENDO EL ESCALADO DE LOS SENSORES
116     #cfg_factor_x := (#cfg_max_vel / #cfg_max_pot_x);
117     #cfg_factor_y := (#cfg_max_vel / #cfg_max_pot_y);
118     #sts_vel_x := ABS((#IN_1_POT_X * 1.0) * #cfg_factor_x);
119     #sts_vel_y := ABS((#IN_2_POT_Y * 1.0) * #cfg_factor_y);
120     #GLOBAL_angulo_offset_x := #sts_vel_x;
121     #GLOBAL_angulo_offset_y := #sts_vel_y;
122     #EJE_2.cmd_invertir_giro := 1;
123     #EJE_3.cmd_invertir_giro := 1;
124     #EJE_4.cmd_invertir_giro := 1;
125     #EJE_5.cmd_invertir_giro := 1;
126
127     END_REGION
128 END_IF;
--

```

- Programación de la función (parte 3)

```

131 IF (#Sts_modos_control = 1)
132 THEN
133     REGION CONTROL POR GIROSCOPIO
134     //SE CALCULA EL SENTIDO DE GIRO
135     IF (#IN_3_GIR_X > (#cfg_neutro_gir_x + #cfg_margen_gir_x)) THEN
136         #sts_izquierda_x := 0;
137         #sts_derecha_x := 1;
138     ELSIF (#IN_3_GIR_X < (#cfg_neutro_gir_x - #cfg_margen_gir_x)) THEN
139         #sts_izquierda_x := 1;
140         #sts_derecha_x := 0;
141     ELSE
142         #sts_izquierda_x := 0;
143         #sts_derecha_x := 0;
144     END_IF;
145     IF (#IN_4_GIR_Y > (#cfg_neutro_gir_y + #cfg_margen_gir_y)) THEN
146         #sts_izquierda_y := 0;
147         #sts_derecha_y := 1;
148     ELSIF (#IN_4_GIR_Y < (#cfg_neutro_gir_y - #cfg_margen_gir_y)) THEN
149         #sts_izquierda_y := 1;
150         #sts_derecha_y := 0;
151     ELSE
152         #sts_izquierda_y := 0;
153         #sts_derecha_y := 0;
154     END_IF;
155     //CÁLCULO DE VELOCIDAD DE MOVIMIENTO EN GRADOS ANGULARES/PASO (°/P)
156     //HACIENDO EL ESCALADO DE LOS SENSORES
157     #cfg_factor_x := (#cfg_max_vel / #cfg_max_gir_x);
158     #cfg_factor_y := (#cfg_max_vel / #cfg_max_gir_y);
159     #sts_vel_x := ABS((#IN_3_GIR_X * 1.0) * #cfg_factor_x);
160     #sts_vel_y := ABS((#IN_4_GIR_Y * 1.0) * #cfg_factor_y);
161     #GLOBAL_angulo_offset_x := #sts_vel_x;
162     #GLOBAL_angulo_offset_y := #sts_vel_y;
163     #EJE_2.cmd_invertir_giro := 1;
164     #EJE_3.cmd_invertir_giro := 1;
165     #EJE_4.cmd_invertir_giro := 1;
166     #EJE_5.cmd_invertir_giro := 1;
167     END_REGION
168 END_IF;

```



- Programación de la función (parte 4)

```

172 REGION LLAMADA A LAS RUTINAS DE CONTROL DE LOS EJES
173 #EJE_1(IN_Activar := (#sts_grupo_ejes_activos = 1) ,
174       IN_Permiso_derecha := #sts_derecha_x,
175       IN_Permiso_izquierda := #sts_izquierda_x,
176       IN_Max_angulo := #GLOBAL_angulo_max,
177       IN_Min_angulo := #GLOBAL_angulo_min,
178       IN_Tiempo_paso := #GLOBAL_tiempo_paso,
179       IN_Angulo_offset := #GLOBAL_angulo_offset_x);
180 #EJE_2(IN_Activar := (#sts_grupo_ejes_activos = 1),
181       IN_Permiso_derecha := #sts_derecha_y,
182       IN_Permiso_izquierda := #sts_izquierda_y,
183       IN_Max_angulo := #GLOBAL_angulo_max,
184       IN_Min_angulo := #GLOBAL_angulo_min,
185       IN_Tiempo_paso := #GLOBAL_tiempo_paso,
186       IN_Angulo_offset := #GLOBAL_angulo_offset_y);
187 #EJE_3(IN_Activar := (#sts_grupo_ejes_activos = 2),
188       IN_Permiso_derecha := #sts_derecha_y,
189       IN_Permiso_izquierda := #sts_izquierda_y,
190       IN_Max_angulo := #GLOBAL_angulo_max,
191       IN_Min_angulo := #GLOBAL_angulo_min,
192       IN_Tiempo_paso := #GLOBAL_tiempo_paso,
193       IN_Angulo_offset := #GLOBAL_angulo_offset_y);
194 #EJE_4(IN_Activar := (#sts_grupo_ejes_activos = 2),
195       IN_Permiso_derecha := #sts_derecha_x,
196       IN_Permiso_izquierda := #sts_izquierda_x,
197       IN_Max_angulo := #GLOBAL_angulo_max,
198       IN_Min_angulo := #GLOBAL_angulo_min,
199       IN_Tiempo_paso := #GLOBAL_tiempo_paso,
200       IN_Angulo_offset := #GLOBAL_angulo_offset_x);
201 #EJE_5(IN_Activar := (#sts_grupo_ejes_activos = 3) ,
202       IN_Permiso_derecha := #sts_derecha_y,
203       IN_Permiso_izquierda := #sts_izquierda_y,
204       IN_Max_angulo := #GLOBAL_angulo_max,
205       IN_Min_angulo := #GLOBAL_angulo_min,
206       IN_Tiempo_paso := #GLOBAL_tiempo_paso,
207       IN_Angulo_offset := #GLOBAL_angulo_offset_y);
208 #EJE_6(IN_Activar := (#sts_grupo_ejes_activos = 3),
209       IN_Permiso_derecha := #sts_derecha_x,
210       IN_Permiso_izquierda := #sts_izquierda_x,
211       IN_Max_angulo := #GLOBAL_angulo_max,
212       IN_Min_angulo := #GLOBAL_angulo_min,
213       IN_Tiempo_paso := #GLOBAL_tiempo_paso,
214       IN_Angulo_offset := #GLOBAL_angulo_offset_x);

```

- Programación de la función (parte 5)

```

217 //ESCRITURA VALORES OPC
218 #OPC_OUT_POS_E1 := #EJE_1.OUT_Angulo_Eje;
219 #OPC_OUT_POS_E2 := #EJE_2.OUT_Angulo_Eje;
220 #OPC_OUT_POS_E3 := #EJE_3.OUT_Angulo_Eje;
221 #OPC_OUT_POS_E4 := #EJE_4.OUT_Angulo_Eje;
222 #OPC_OUT_POS_E5 := #EJE_5.OUT_Angulo_Eje;
223 #OPC_OUT_POS_E6 := #EJE_6.OUT_Angulo_Eje;
224 #OPC_OUT_ACTIVAR_VENTOSA := #IN_6_BOT_Z;
225
226
227 //ESCRITURA VALORES HMI
228 #HMI_OUT_GRUPO_1 := (#sts_grupo_ejes_activos = 1);
229 #HMI_OUT_GRUPO_2 := (#sts_grupo_ejes_activos = 2);
230 #HMI_OUT_GRUPO_3 := (#sts_grupo_ejes_activos = 3);
231
232 #HMI_OUT_P_E1 := #EJE_1.Angulo_actual;
233 #HMI_OUT_P_E2 := #EJE_2.Angulo_actual;
234 #HMI_OUT_P_E3 := #EJE_3.Angulo_actual;
235 #HMI_OUT_P_E4 := #EJE_4.Angulo_actual;
236 #HMI_OUT_P_E5 := #EJE_5.Angulo_actual;
237 #HMI_OUT_P_E6 := #EJE_6.Angulo_actual;
238
239 #HMI_OUT_VEL_X := #sts_vel_x / (#GLOBAL tiempo_paso * 1.0 / 1000);
240 #HMI_OUT_VEL_Y := #sts_vel_y / (#GLOBAL tiempo_paso * 1.0 / 1000);
241 #HMI_OUT_VENTOSA := (#OPC_OUT_ACTIVAR_VENTOSA = 1);
242
243
244 //LLAMADA A LA RUTINA QUE GESTIONA LAS POSICIONES
245 "3_Gestion_Posiciones_DB"();

```

## Anexo V. Programación del sistema robótico en Robot Studio

### PROGRAMACIÓN DE LA CONTROLADORA EN RAPID

- Se declaran las variables de proceso

```
MODULE Module1

]   PERS NUM IN_POS_E1;
    PERS NUM IN_POS_E2;
    PERS NUM IN_POS_E3;
    PERS NUM IN_POS_E4;
    PERS NUM IN_POS_E5;
    PERS NUM IN_POS_E6;
    PERS NUM IN_Contador_Vida;
    PERS NUM OUT_Contador_Vida;
    PERS NUM ENTRADA_PR;
    PERS NUM Pos_Eje_X;
    PERS NUM Pos_Eje_Y;
    PERS NUM Pos_Eje_Z;
    VAR NUM LEER;
    VAR robtarget pTemp;
    VAR jointtarget jt_eje1:=[[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR speeddata IN_velocidad:=v600;
    VAR Num coeficiente_escalado:=100;
```

- Bucle que ejecuta el programa de forma cíclica

```
PROC main()
    WHILE (TRUE) DO
        !LLAMADA A LAS FUNCIONES DEL PROGRAMA
        control_ejes_def;
        mostrar_info_ejes;
        latencia;
        LEER:=IN_PRUEBA_FISICA;
    ENDWHILE
```

- Funciones secundarias del programa llamadas desde el bucle principal

```
PROC control_ejes_def()
    !RUTINA QUE LEE LOS VALORES RECIBIDOS POR OPC Y CONTROLA LOS EJES
    jt_eje1.robax.rax_1:=IN_POS_E1/coeficiente_escalado;
    jt_eje1.robax.rax_2:=IN_POS_E2/coeficiente_escalado;
    jt_eje1.robax.rax_3:=IN_POS_E3/coeficiente_escalado;
    jt_eje1.robax.rax_4:=IN_POS_E4/coeficiente_escalado;
    jt_eje1.robax.rax_5:=IN_POS_E5/coeficiente_escalado;
    jt_eje1.robax.rax_6:=IN_POS_E6/coeficiente_escalado;
    MoveAbsj jt_eje1,IN_velocidad,fine,tool0;
ENDPROC

PROC mostrar_info_ejes()
    !RUTINA QUE LEE EL VALOR DE LA POSICIÓN DE LA VENTOSA Y ENVÍA LOS VALORES POR OPC
    pTemp := CRobT();
    Pos_Eje_X:=pTemp.trans.x/1000;
    Pos_Eje_Y:=pTemp.trans.y/1000;
    Pos_Eje_Z:=pTemp.trans.z/1000;
ENDPROC

PROC latencia()
    !FUNCIÓN PARA REALIZAR EL CÁLCULO DE LA LATENCIA DE LA CONEXIÓN EN EL PLC SIMULADO
    OUT_Contador_Vida:=IN_Contador_Vida;
ENDPROC
```