UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

## Escuela Técnica Superior de Ingeniería Industrial

# Diseño de un banco de ensayos de máquinas de imanes permanentes de técnica senoidal (servoaccionamiento PMSM Brushless AC) de 2,2kW operados mediante autómata programable y con interfaz HTML basada en ESP32.

Trabajo Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

AUTOR/A: Soriano Cuesta, Ángela

Tutor/a: Martínez Román, Javier Andrés

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

# DISEÑO DE UN BANCO DE ENSAYOS DE MÁQUINAS DE IMANES PERMANENTES DE TÉCNICA SENOIDAL (SERVOACCIONAMIENTO PMSM BRUSHLESS AC) DE 2,2kW OPERADOS MEDIANTE AUTÓMATA PROGRAMABLE Y CON INTERFAZ HTML BASADA EN ESP32

AUTOR: Ángela Soriano Cuesta

TUTOR: Javier Andrés Martínez Román

**Curso Académico: 2022/23**

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

# ÍNDICE DE DOCUMENTOS

# RESUMEN

Este trabajo trata el diseño de un banco de ensayos de máquinas de imanes permanente de técnica senoidal de manera que el usuario sea capaz de controlarlo desde cualquier dispositivo mediante una interfaz HTML basada en ESP32. El banco está formado por dos servomotores de la empresa B&R controlados por un autómata programable que es capaz de comunicarse mediante TCP/IP con un microcontrolador ESP32. El primer motor trabaja en modo control de velocidad y el segundo en modo control de par.

Por otro lado, el microcontrolador ESP32 se caracteriza por ofrecer conectividad Wifi y Bluetooth, siendo capaz de actuar en una amplia esfera de aplicaciones. En este trabajo se encarga de diseñar la interfaz de usuario de manera que permita mostrar y modificar el estado de los servomotores, además de representar gráficamente en tiempo real el parámetro de velocidad. Para ello utilizará el entorno Arduino que hace uso del lenguaje C++ y que es una de las plataformas más comunes debido a su facilidad, rapidez y versatilidad.

B&R es una empresa líder en el campo de la automatización que cuenta con numerosos productos, entre ellos destacan sus servomotores Brushless capaces de controlar su posición y velocidad. Además, posee su propia herramienta de programación, Automation Studio, que contiene una gran variedad de recursos, librerías y lenguajes de programación que facilitan la configuración y estructura del proyecto. Mediante este programa se controla el movimiento de los motores y el sistema de comunicación de datos entre el autómata y el ESP32.

**Palabras clave**: Interfaz de usuario WEB, comunicación TCP, PLC, ESP32, banco de ensayos de motores

# RESUM

Aquest treball tracta el disseny d'un banc d'assajos de màquines d'imants permanent de tècnica senoidal de manera que l'usuari siga capaç de controlar-ho des de qualsevol dispositiu mitjançant una interfície HTML basada en ESP32. El banc està format per dos servomotors de l'empresa B&R controlats per un autòmat programable que és capaç de comunicar-se mitjançant TCP/IP amb un microcontrolador ESP32. El primer motor treballa en manera control de velocitat i el segon en manera control de parell.

D'altra banda, el microcontrolador ESP32 es caracteritza per oferir connectivitat Wifi i Bluetooth, sent capaç d'actuar en una àmplia esfera d'aplicacions. En aquest treball s'encarrega de dissenyar la interfície d'usuari de manera que permeta mostrar i modificar l'estat dels servomotors, a més de representar gràficament en temps real el paràmetre de velocitat. Per a això utilitzarà l'entorn Arduino que fa ús del llenguatge C++ i que és una de les plataformes més comunes a causa de la seua facilitat, rapidesa i versatilitat.

B&R és una empresa líder en el camp de l'automatització que compta amb nombrosos productes, entre ells destaquen els seus servomotors Brushless capaços de controlar la seua posició i velocitat. A més, posseeix la seua pròpia eina de programació, Automation Studio, que conté una gran varietat de recursos, llibreries i llenguatges de programació que faciliten la configuració i estructura del projecte. Mitjançant aquest programa es controla el moviment dels motors i el sistema de comunicación de dades entre l'autòmat i l'ESP32.

**Paraules clau**: Interfície d'usuari WEB, comunicació TCP, PLC, ESP32, banc d'assajos de motors

# ABSTRACT

This project deals with the design of a test bench for permanent magnet machines of sinusoidal technique so that the user is able to control it from any device through an HTML interface based on ESP32. The bench is made up of two B&R company servomotors controlled by a programmable controller that is capable of communicating via TCP/IP with an ESP32 microcontroller. The first motor works in speed control mode and the second in torque control mode.

On the other hand, the ESP32 microcontroller is characterized by offering Wi-Fi and Bluetooth connectivity, being capable of acting in a wide range of applications. In this project, it oversees the design of the user interface in such a way that allows showing and modifying the status of the servomotors, as well as graphically representing the speed parameter. For this, it will use the Arduino environment that makes use of the C++ language and that is one of the most common platforms due to its ease, speed and versatility.

B&R is a leading company in the field of automation that has numerous products, including its Brushless servomotors capable of controlling their position and speed. In addition, it has its own programming tool, Automation Studio, which contains a wide variety of resources, libraries and programming languages that facilitate the configuration and structure of the project. Through this program the movement of the motors and the data communication system between the PLC and the ESP32 are controlled.

**Keywords**: WEB user interface, TCP communication, PLC, ESP32, motor test bench

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

# MEMORIA

*Diseño de un banco de ensayos de máquinas de imanes permanentes de técnica senoidal (servoaccionamiento PMSM Brushless AC) de 2,2kW operados mediante autómata programable y con interfaz HTML basada en ESP32.*

Ángela Soriano Cuesta

Curso 2022/23

# ÍNDICE DE LA MEMORIA

# ÍNDICE DE FIGURAS

# ÍNDICE DE CÓDIGOS

# CAPÍTULO 1: INTRODUCCIÓN, ANTECEDENTES Y MOTIVACIÓN

### *1.1. Introducción*

El objetivo principal de este proyecto se basa en el diseño y control de un banco de ensayos de máquinas de imanes permanentes de técnica senoidal mediante una interfaz HTML creada por un microcontrolador ESP32, de manera que el usuario sea capaz de mandar órdenes y recibir datos desde cualquier dispositivo que soporte clientes HTML (navegadores). El banco de ensayos está formado por dos servomotores que actúan de manera diferente, ya que uno trabaja en modo control de velocidad y el otro en modo control de par. Estos estarán controlados por un autómata programable (PLC) y comunicados con el ESP32 mediante ese mismo PLC programado con ayuda de la aplicación Automation Studio. El autómata no solo será capaz de leer los datos de velocidad y par de los servomotores, sino que también podrá enviar y recibirlos del usuario mediante el ESP32, programado con el software Arduino IDE.

Así, se podrá dividir el proyecto en dos partes de mayor importancia y diferenciadas: la programación del PLC mediante Automation Studio y del ESP32 con Arduino IDE.
En cuanto al Automation Studio, se pretende controlar tanto el movimiento de ambos servomotores, como el envío y recepción de datos. De esta manera, el objetivo de este programa en el proyecto es ser capaz de comunicarse mediante TCP/IP con el ESP32, realizar las acciones que el usuario solicite y leer los datos de velocidad.
Por otra parte, con el software de Arduino IDE, los objetivos claros son la conexión a Wifi necesaria para utilizar el servidor web, el diseño de la página web que servirá de interfaz de usuario y la comunicación con el autómata, tanto para enviarle las órdenes de los usuarios como para recibir los datos de los servomotores y mostrarlos en tiempo real.

### *1.2. Antecedentes: Banco de ensayos basado en Servomotores Brushless*

Un servomotor es un motor que genera una elevada potencia cuyos parámetros de velocidad, aceleración y posición angular se pueden controlar mediante un sensor de posición. En su interior contiene un encoder que actúa como sensor proporcionando retroalimentación de posición para controlar su velocidad de rotación y su posición. Los servomotores son utilizados para numerosas aplicaciones específicas donde se requieren variaciones continuas de velocidad, especialmente en el sector de la automatización. Funcionan mediante diferentes y muy variados sistemas de control de la tensión que se comunican con el motor para establecer su par y con él, la velocidad y posición del eje, siendo la longitud de ese punto la que determina hasta dónde debe girar el motor. Así, cuando se le da una orden de movimiento al servomotor, este se moverá a la posición deseada y se mantendrá ahí. (Aula21, s.f)

En este proyecto, se han utilizado dos servomotores Brushless de técnica senoidal de la empresa B&R. B&R es una empresa austríaca líder en el campo de la automatización industrial que potencia la innovación y la calidad de sus productos. (B&R, 2017) Forma parte del grupo ABB, el cual busca soluciones más sostenibles y eficientes para optimizar procesos y recursos. (ABB, 2020) Además, cuenta con una amplia gama de productos que permiten la automatización de máquinas y sistemas, además de un software propio capaz de programar y configurar sus productos, el Automation Studio. En el caso de sus motores de imán permanente, estos se caracterizan por su elevado rendimiento, dinamismo y adaptabilidad. (B&R, 2020)

Actualmente, los servomotores Brushless son los más utilizados en la industria y se trata de motores de corriente alterna sin escobillas. Estos son capaces de desarrollar una mayor potencia para menor volumen en comparación con los motores tradicionales, ya que están formados por un estator segmentado donde se ha rellenado el espacio del doble de cobre. También es importante mencionar la elevada precisión en cuanto a la velocidad y a la posición que estos servomotores son capaces de proporcionar.

Por otro lado, el sistema de regulación y control de los servomotores utilizados sigue un control senoidal. Esta técnica senoidal se considera el método más complejo a la hora de regular la posición y la velocidad, pero también es el que proporciona mejores prestaciones al sistema de regulación. A partir de éste, se conoce la posición absoluta del rotor del motor de manera precisa y se puede controlar de forma continua. (Quimel, 2011)
El sistema de fábrica de los motores utilizados en este proyecto permite operar los servomotores mediante un modo de test, el cual requiere que se siga un procedimiento específico para el correcto funcionamiento del banco. Éste es un proceso poco intuitivo e incómodo para trabajar especialmente cuando se desea realizar distintos ensayos, debido a que requiere su conocimiento previo y que puede dar lugar a diversos errores.

### *1.3. Motivación*

La principal meta de este banco de ensayos es que sea una herramienta de aprendizaje para el alumnado que lo utilice para realizar ensayos de accionamientos. Es por eso que el proceso de funcionamiento debe ser sencillo de entender y los resultados fáciles de obtener, sin embargo, el modo de test del programa no cumple ninguno.
Debido a la dificultad de la forma de operación del banco, se quiere encontrar una manera más fácil de operar en forma conjunta los motores. De esta manera, la interfaz que se le muestra al usuario deberá ser más sencilla y centrada en los resultados a la hora de realizar ensayos. Por ello, se ha planteado una interfaz que contenga tan solo los diversos botones que sigan el orden del funcionamiento de los motores y que además muestre gráficamente los datos de velocidad en tiempo real.
Así, a lo largo de este trabajo se ha diseñado una interfaz HTML a la que los alumnos puedan conectarse desde cualquier dispositivo que admita navegadores, ya sea su teléfono móvil o su ordenador propio. Desde esta serán capaces de poner en marcha los motores, además de poder mandar el dato de velocidad o de par que se desee. También existirá otra pestaña que represente esos valores, facilitando el estudio del banco de ensayos.

### *1.4. Objetivos de desarrollo sostenible*

Los Objetivos de Desarrollo Sostenible (ODS) son una serie de metas aprobadas por la Organización de Naciones Unidas en 2015 que pretenden mejorar la vida de todos para el año 2030 en cuanto a pobreza, educación, igualdad y medio ambiente. La siguiente tabla representa el grado de relación de cada uno de estos objetivos con este proyecto.

| **Objetivos de Desarrollo Sostenibles** | **Alto** | **Medio** | **Bajo** | **No procede** |
|---|---|---|---|---|
| ODS 1. **Fin de la pobreza** | | | X | |
| ODS 2. **Hambre cero** | | | X | |
| ODS 3. **Salud y bienestar** | | | X | |
| ODS 4. **Educación de calidad** | X | | | |
| ODS 5. **Igualdad de género** | | | X | |
| ODS 6. **Agua limpia y saneamiento** | | | X | |
| ODS 7. **Energía asequible y no contaminante** | | X | | |
| ODS 8. **Trabajo decente y crecimiento económico** | | X | | |
| ODS 9. **Industria, innovación e infraestructuras** | | | X | |
| ODS 10. **Reducción de las desigualdades** | | | X | |
| ODS 11. **Ciudades y comunidades sostenibles** | | | X | |
| ODS 12. **Producción y consumo responsables** | | | X | |
| ODS 13. **Acción por el clima** | | X | | |
| ODS 14. **Vida submarina** | | | X | |
| ODS 15. **Vida de ecosistemas terrestres** | | | X | |
| ODS 16. **Paz, justicia e instituciones sólidas** | | | X | |
| ODS 17. **Alianzas para lograr objetivos** | | | X | |

En concreto el objetivo número cuatro busca una educación inclusiva, equitativa y de calidad, además de oportunidades de aprendizaje para todos. Este proyecto va dirigido al aprendizaje de las personas, potenciando especialmente este objetivo. Una de las metas específicas para 2030 se trata de aumentar el número de jóvenes y adultos que tienen las competencias técnicas y profesionales para acceder al empleo, el trabajo decente y el emprendimiento. (Unidas, s.f.)

Los objetivos de energía asequible y no contaminante y de acción por el clima se han considerado que tienen un grado medio de relación con el proyecto, ya que en este se utilizan motores eléctricos, utilizando una energía potencialmente menos contaminante. Mientras que el objetivo de trabajo decente y crecimiento económico se puede considerar que se relaciona con el trabajo, al tratarse este de un banco de ensayos. Estos ensayos buscan optimizar el funcionamiento de los productos, logrando niveles de eficiencia y productividad mayores.

### *1.5. Resumen de la memoria*

En la siguiente memoria se encuentran una serie de capítulos que explican el procedimiento que se ha seguido para conseguir un correcto funcionamiento del banco de ensayos mediante una interfaz de usuario más sencilla y clara.

En primer lugar, se trata el diseño básico de las órdenes que debe ejecutar el autómata para manejar las operaciones del banco, así como el funcionamiento del intercambio de información entre PLC y ESP32. Por un lado, se explica el orden del procedimiento para poner en marcha los motores, así como la manera en la que el usuario comprueba que se está realizando correctamente. Por otro lado, se diferencian los tipos de datos que el usuario puede pedirle al banco y se explica cómo se codifican los programas del autómata y del microcontrolador a la hora de comunicarse y distinguir esos datos.

A continuación, se redacta el proceso de programación del autómata mediante la aplicación Automation Studio de manera que cumpla las siguientes tres funciones: ejecutar las distintas operaciones de los motores, comunicarse con el microcontrolador para enviar y recibir datos y relacionar los datos con las operaciones. Para ello, se hará una introducción al Automation Studio y se explicará cómo se debe configurar e instalar el proyecto. Después se detallarán los diferentes módulos que ejecutan cada función.

Por último, se explica el procedimiento seguido para realizar el diseño de la interfaz HTML que se le presentará al usuario mediante el microcontrolador ESP32 y el entorno Arduino. Éste es el encargado de, además de diseñar la página web, convertir y transferir los valores entre usuario y autómata. Para ello, se detallará cada parte del código de Arduino IDE que ejecuta las funciones necesarias para realizar lo planteado.

# CAPÍTULO 2: NORMATIVA

En este proyecto intervienen distintas normas que regulan los diversos elementos que participan en la correcta ejecución del trabajo.

En primer lugar, la norma aplicada a los a los autómatas programables (PLC) y a sus periféricos asociados (IEC-61131). Esta no solo define e identifica las características principales de los PLC, sino que también especifica sus requisitos mínimos, condiciones de servicio, aspectos constructivos y su seguridad general. Además, cabe destacar el apartado de la normativa IEC-61131-3, donde se define las especificaciones de los lenguajes de programación de los autómatas, incluyendo el modelo de software y la estructura del lenguaje. (GENIA, 2006)

Por otro lado, para la programación del ESP32 se utiliza un lenguaje basado en C++, el cual se define en la norma ISO/IEC 14882:2020, donde también se especifican los requisitos para su implementación. C++ es un lenguaje de programación basado en el lenguaje C, como se describe en ISO/IEC 9899:2018. Ambos documentos establecen la representación de programas y de datos, así como las reglas y las limitaciones de cada uno de los lenguajes de programación. (ISO, 2020)

Por último, cabe mencionar la normativa que describe el lenguaje HTML, utilizado para la realización de la página web que funcionará como interfaz de usuario. La última versión actualizada del estándar se encuentra en la norma ISO/IEC 15445:2000. (ISO, 2003)

# CAPÍTULO 3: PLANTEAMIENTO GENERAL DEL DISEÑO: INTRODUCCIÓN A LOS PROGRAMAS DEL AUTÓMATA Y DEL MICROCONTROLADOR

Como se ha explicado previamente, el objetivo principal de este trabajo es controlar el movimiento de dos servomotores Brushless a partir de la comunicación entre un autómata y un microcontrolador ESP32 que implementará una interfaz de usuario basada en servidor HTML. De esta manera, primeramente, se deberá explicar cómo funcionan los servomotores y seguidamente cómo es capaz de comunicarse el ESP32 con el PLC para realizar correctamente esa puesta en marcha de los motores.

### *3.1. Modo de funcionamiento*

En primer lugar y partiendo de un estado de apagado, se deberá encender el interruptor diferencial, permitiendo que entre corriente en el banco de ensayos y se inicie y prepare el programa del autómata que controla el banco. Esta operación es manual y se mantendrá así una vez se haya implementado la nueva interfaz de operación del banco. A partir de este punto se describe la secuencia de operación normal para presentar las diferentes órdenes que se deben emitir desde la interfaz de operación del banco. Una vez conectada la alimentación de potencia y preparado e iniciado el autómata, habrá que permitir o habilitar *(Enable)* el funcionamiento del servomotor correspondiente que se quiere poner en marcha. Cada servoamplificador tiene conectada su entrada de habilitación a una salida digital del autómata que habrá que habilitar mediante el botón *Enable* de la interfaz de usuario*,* el servomotor 1 está conectado a la tercera salida y el servomotor 2 a la cuarta de la tarjeta.

Llegados a este punto, los servoamplificadores requieren la orden de encendido para poder ejecutar el resto de operaciones, por lo que se les suministrará una vez se aprete al botón *Power* de la interfaz de usuario*.* Previamente a la puesta en marcha del motor correspondiente, existe un paso intermedio necesario y típico del control de servoaccionamientos que se trata de la función *Homing* (con el mismo nombre en la interfaz de usuario) o búsqueda del punto de referencia, que permite que el control localice la posición de referencia del eje y permita su funcionamiento posterior ante cualquier orden de posicionamiento o de movimiento. Finalmente, ya solo faltaría iniciar el movimiento del servomotor (*on* en la interfaz de usuario*)*.

El procedimiento a la hora del apagado del motor sería justo el inverso, sin ser necesario pasar por el *Homing*. Se pararía el movimiento del motor (*off* en la interfaz de usuario), se apagaría el servoamplificador (paso a su estado de reposo en el que no está permitido ejecutar órdenes de movimiento) volviendo a apretar el botón de *Power* y se deshabilitaría su funcionamiento pasando a nivel bajo la salida correspondiente del autómata mediante el botón *Disable* de la interfaz de usuario. Si no se siguiera este orden de manera exacta, se pasaría a un estado de

error que impediría la puesta en marcha del motor. Para volver a poner en funcionamiento el motor, se necesitaría apretar el botón Error (reconocimiento de error) que avisa de que el usuario conoce que ha realizado un error y permite resetearlo para así permitir continuar con la secuencia de encendido. Seguidamente habría que repetir los pasos explicados anteriormente.

Ahora ya se conoce el modo de funcionamiento, sin embargo, una persona que pretende poner en marcha un servomotor no es capaz de saber con certeza si ha realizado la puesta en marcha en el orden correcto o si se encuentra en el estado de error. Para ello se ha implementado una parte del código que active una salida asociada a cada uno de los estados comentados apretando la orden que se solicite desde la interfaz de usuario. Así, el usuario es capaz de observar si las salidas están encendidas o no de manera visual en el propio autómata programable y comprobar que está realizando el proceso de paso de un estado de operación al siguiente correctamente.
En primer lugar, la salida número cinco se ha vinculado con el botón *Power*, de manera que cuando este se apriete, la salida se encenderá. Por el contrario, cuando se quiera apagar el botón, la salida también lo hará. Cuando el autómata se encuentre en el estado de *Homing,* la salida siete se encenderá y cuando se haya realizado la acción con éxito, se apagará. Cabe mencionar que este proceso se hace de manera bastante rápida de modo que esta salida tan solo parpadeará, pero será suficiente para informar al usuario de que puede continuar con el proceso de puesta en marcha.
La siguiente salida, la número siete, se ha enlazado con el inicio y el paro de movimiento del servomotor. Cuando se ejecute una orden de movimiento, la salida se encenderá y cuando se deje de mover por haberse emitido la orden de paro, se apagará. Finalmente, mientras que el PLC se encuentre en estado de error, la salida 8 estará encendida. Solo cuando el usuario apriete al botón de error y así el autómata salga de este estado, la salida 8 dejará de estar encendida.

### *3.2. Códigos de comunicación*

A la hora de programar el ESP32 para que controle la puesta en marcha de los servomotores, este debe de ser capaz de enviarle un valor al autómata y que éste reconozca la acción o el dato al cual se corresponde. Además, esta información debe ser enviada como órdenes binarias, por lo que será importante traducir las órdenes y los valores de velocidad y par a números binarios para su correspondiente envío.
Primeramente, podemos distinguir dos tipos de valores que se van a enviar: acciones que participan en el funcionamiento de los servomotores y datos de velocidad y par. Como se está trabajando con datos binarios de 16 bits, se ha decidido utilizar el bit de mayor peso para realizar la distinción de tipo de dato. De esta manera, si el bit es 1 se tratará de un nuevo dato de velocidad o par, y si es 0, será un valor de acción. El resto de bits serán los que representarán el propio valor de velocidad y par o el significado de la acción correspondiente.

#### *3.2.1. Datos de acción*

A continuación, se mostrará una tabla que representa las diferentes acciones y su valor en binario que se ha relacionado. De esta manera a la hora de programar tanto el ESP32 como el autómata, cada número describa la acción que se pretende.

| Acción | Código (binario) | Número (decimal) |
|---|---|---|
| *Enable 1* | 0000000000000001 | 1 |
| *Enable 2* | 0000000000000010 | 2 |
| *Disable 1* | 0000000000000011 | 3 |
| *Disable 2* | 0000000000000100 | 4 |
| *Power 1* | 0000000000000101 | 5 |
| *Power 2* | 0000000000000110 | 6 |
| *Homing 1* | 0000000000000111 | 7 |
| *Homing 2* | 0000000000001000 | 8 |
| *Error 1* | 0000000000001001 | 9 |
| *Error 2* | 0000000000001010 | 10 |
| *Off 1* | 0000000000001011 | 11 |
| *On 1* | 0000000000001100 | 12 |
| *On 2* | 0000000000001101 | 13 |
| *Off 2* | 0000000000001110 | 14 |

Por ejemplo, cuando el usuario quiera habilitar el primer servomotor, el ESP32 reconocerá la acción *Enable 1* y le enviará un 1 al autómata. Éste recibirá ese dato y distinguirá que se trata de una acción, al tener el bit de mayor peso a cero, y además sabrá que debe habilitar la salida del servomotor 1 y lo ejecutará.

### *3.2.2. Datos de velocidad y de par*

Cuando se trate de datos de velocidad y de par, el usuario le pedirá un valor concreto al ESP32 y éste tendrá que convertirlo para que cuando le llegue al autómata, sea capaz de reconocer que se trata de un dato de velocidad o par y de qué valor se trata. Para ello, en el código realizado mediante Arduino IDE se deberá modificar el valor que se da por el cliente para que el bit de mayor peso sea 1, y el resto de bits representen el valor correcto. De la misma manera, se ha codificado el programa del autómata mediante Automation Studio para que cuando le llegue este valor y reconozca que es un dato nuevo de velocidad o de par, almacene el valor sin contar el bit de mayor peso.

Además, también se deberá distinguir si es un dato de velocidad o de par, por lo que se ha vuelto a utilizar un bit, en este caso el segundo bit de mayor peso, para realizar la distinción. Si el bit se trata de un 1, se trataría de un valor nuevo de par, mientras que, si el bit es 0, sería un valor de velocidad. Se codificaría tanto el programa de Arduino IDE como el Automation Studio, igual que para distinguir entre dato de acción y velocidad o par, pero con el número de bit cambiado.

Por otro lado, los valores de velocidad y de par que el usuario pida por la interfaz, son datos analógicos, pero a la hora de trabajar con comunicación TCP/IP entre microcontrolador y autómata, se utilizarán datos digitales. Para pasar de un valor analógico dado por el cliente al valor digital que mandará el ESP32 al PLC, se ha seguido los pasos correspondientes a la codificación digital de valores. Se supone que, tanto para el par como para la velocidad, el valor mínimo que puede solicitar el usuario es 0 y el valor máximo será establecido como el límite que puede soportar el propio servomotor. Por tanto, cada valor de velocidad o par tendrá asociado

un valor digital siguiendo la línea de transferencia correspondiente. Trabajando con 14 bits, ya que el decimoquinto y decimosexto tienen otra función más allá de mandar información, se podrán escribir hasta $2^n$-1 valores, es decir, 16383. A partir de una simple regla de tres, el ESP32 es capaz de mandar el valor digital que representa la velocidad en revoluciones por minuto o el par en Newtons metro, y el autómata es capaz de leer ese dato y transformarlo de vuelta a la escala de cada uno de esos parámetros en las órdenes de operación del servoamplificador.

# CAPÍTULO 4: AUTOMATION STUDIO: DISEÑO DEL CÓDIGO DEL AUTÓMATA

### *4.1. Introducción al Automation Studio*

Automation Studio es la herramienta de programación que utilizan los equipos de la empresa B&R, que incluye el autómata, el *motion* o sistema de control y operación de movimiento, la seguridad y la visualización. Permite estructurar y configurar un proyecto de manera sencilla, además de contar con una gran variedad de herramientas y librerías que permiten trabajar de forma eficiente. Cabe destacar también la posibilidad que tiene el programa de elegir entre diferentes lenguajes de programación y opciones de simulación que facilitan la realización del proyecto. (B&R, 2018)

Se ha utilizado el Texto Estructurado como lenguaje de programación en este proyecto, ya que se trata de un lenguaje de alto nivel que es altamente utilizado especialmente en el campo de la automatización. Se caracteriza por permitir una programación rápida y eficiente, gracias a la facilidad en el uso de construcciones estandarizadas. Además, es un lenguaje flexible, ya que utilizada numerosas herramientas de otros lenguajes como variables, operadores, funciones y elementos que ayudan a controlar el proyecto. (B&R, 2017)

En este trabajo, se ha utilizado el Automation Studio para servir de enlace entre el ESP32, encargado de recibir las órdenes del usuario, y el propio motor. Así, el microcontrolador manda los diferentes datos que desea el usuario y se los transmite al autómata mediante comunicación TCP/IP. Este es capaz de leer las órdenes y ejecutarlas, además de enviarle a la vez al ESP32 y al servidor web los datos en tiempo real de velocidad de los servomotores.

### *4.2. Concepto básico de Automation Studio*

Una vez se abre el proyecto a realizar mediante el Automation Studio, se puede observar distintas áreas de trabajo que permiten ejecutar diferentes funciones.

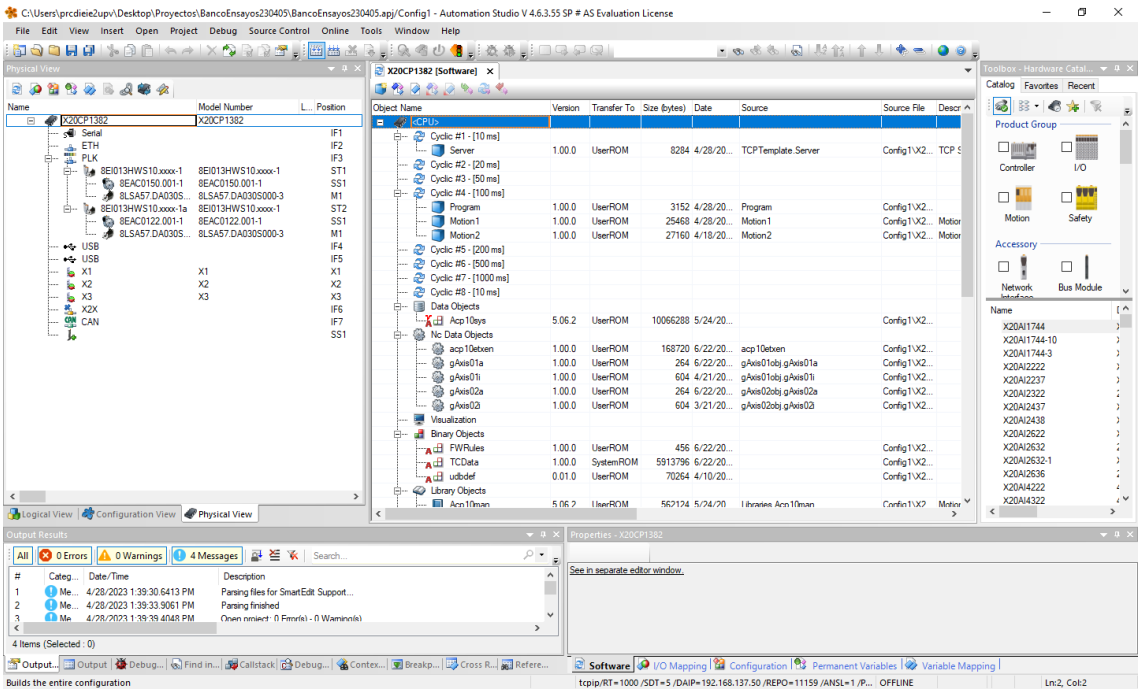


*Figura 1. Programa Automation Studio. Pantalla de inicio.*

El elemento central de la interfaz es el explorador del proyecto, que se encuentra a la izquierda de la pantalla y el cual se divide, a su vez, en tres pestañas distintas. La *Logical View* es la encargada de organizar el software y gestionar los objetos del proyecto. Todos los elementos del software necesarios se organizan en paquetes y se ordenan en forma de árbol en la *Logical View*.

En concreto, esta ventana organizará los distintos módulos del proyecto y contendrá las librerías necesarias para ejecutarlos, además de las variables y los datos globales que intervienen en su funcionamiento.

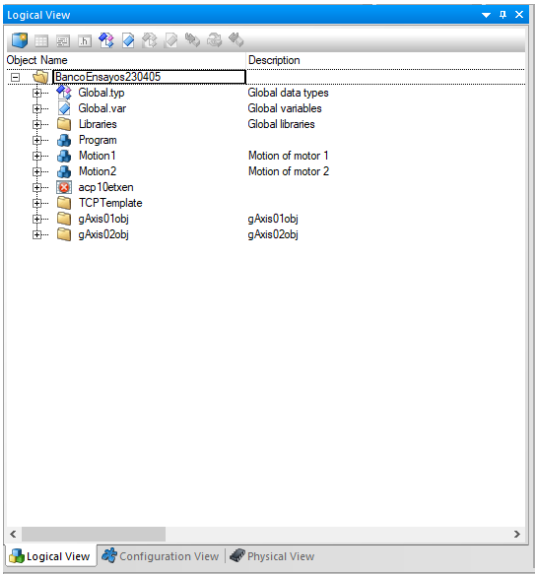



*Figura 2. Programa Automation Studio. Logical View.*

En segundo lugar, se encuentra la *Configuration View*, capaz de gestionar distintas configuraciones de una misma máquina. Cada configuración será diferente al resto debido a las variaciones en el software o en el hardware utilizado, sin embargo, solo una podrá estar activada a la vez. Para este proyecto no ha sido necesario crear más de una configuración.

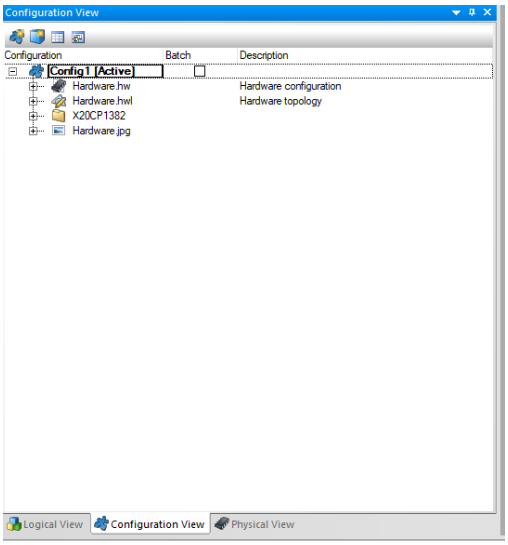



*Figura 3. Programa Automation Studio. Configuration View.*

Por último, los elementos del hardware configurado se organizan de manera jerárquica y se muestran al usuario en la *Physical View*. En esta, se podrá configurar y modificar los módulos, la red y el comportamiento del sistema.

Para este proyecto, se tiene como elemento principal el PLC, del cual se pueden desglosar sus distintas conexiones, entre ellas los servoaccionamientos y los canales de entrada y salida. Desde esta ventana se podría realizar la configuración de cada uno de los elementos que conforman este proyecto.

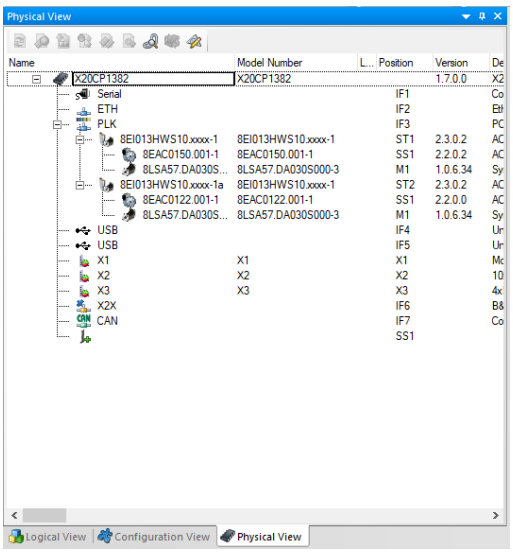



*Figura 4. Automation Studio. Physical View.*

En el centro de la pantalla se encuentra el área de trabajo que presenta el documento con el que se está trabajando en ese momento. En este caso se muestra el Software del proyecto en el que se puede destacar los cuatro módulos que se utilizarán para el funcionamiento de éste: *Server*, *Program*, *Motion 1* y *Motion 2*.

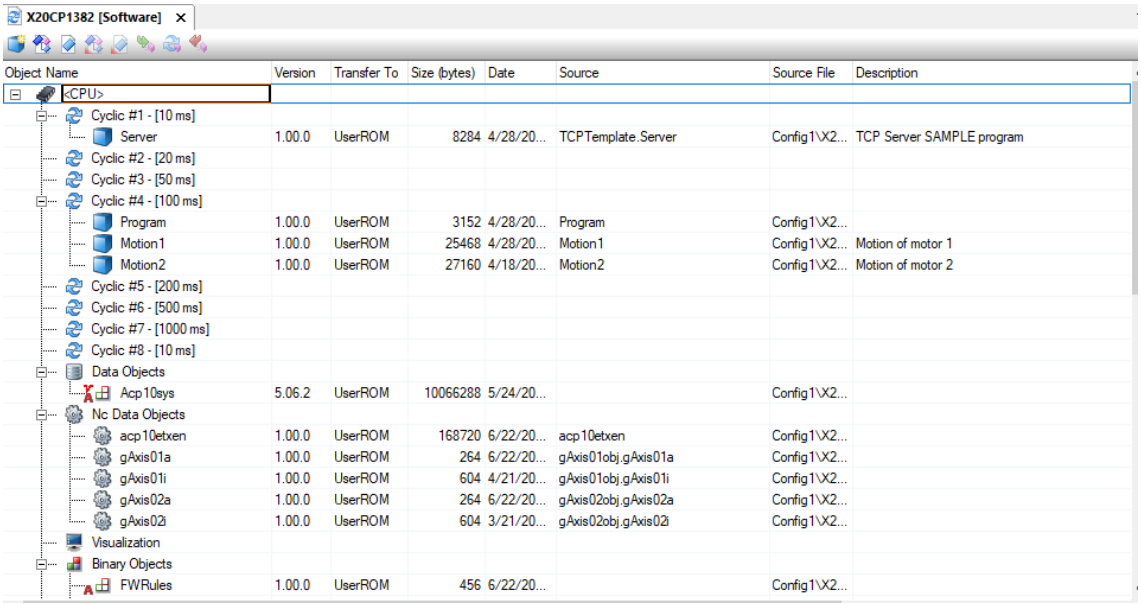



*Figura 5. Programa Automation Studio. Software del proyecto.*

A la derecha, se muestra el catálogo de objetos, *Toolbox,* el cual permite añadir elementos al proyecto. Cada vez que se quiera añadir un elemento al trabajo que se desea realizar, se buscará en el catálogo y se añadirá al proyecto.

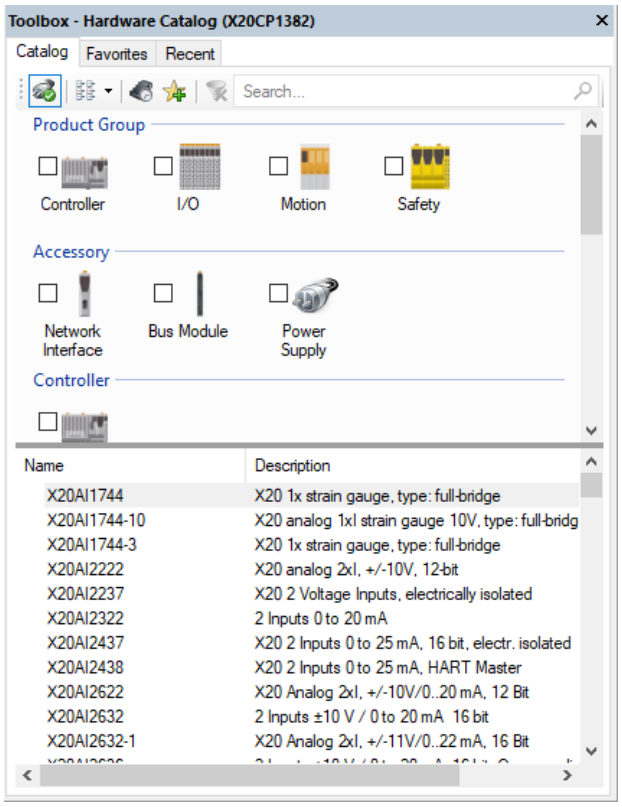



*Figura 6. Programa Automation Studio. Catálogo de objetos.*

Mediante éste, se crea el banco de ensayo en la propia aplicación, el cual se representa de manera gráfica en la vista *System Designer*. De esta forma se puede observar y organizar el sistema desde el programa, tal y como se haría en la vida real.

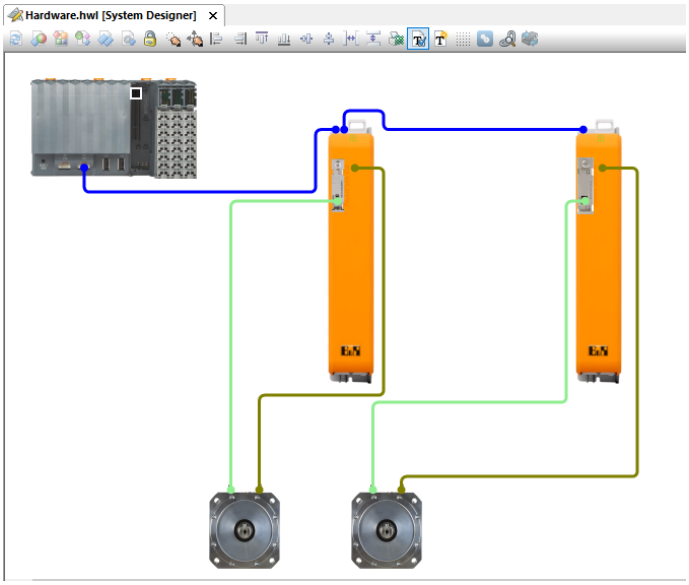



*Figura 7. Programa Automation Studio. System Designer.*

Los resultados tras compilar el programa se mostrarán en la *Output Window*, situada en la parte inferior izquierda de la pantalla. Si existiera algún error, advertencia o simplemente información, se mostraría por esta ventana.

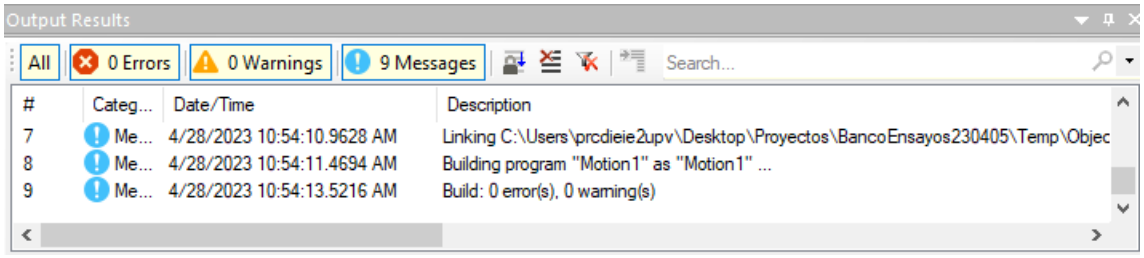



*Figura 8. Programa Automation Studio. Output Window.*

Finalmente, a la derecha de la pantalla de salida podrá aparecer un panel de propiedades o *Properties Window* que como su nombre indica, mostrará las propiedades del elemento seleccionado. Si se deseara, en esta ventana también sería posible modificar las propiedades de ese objeto.

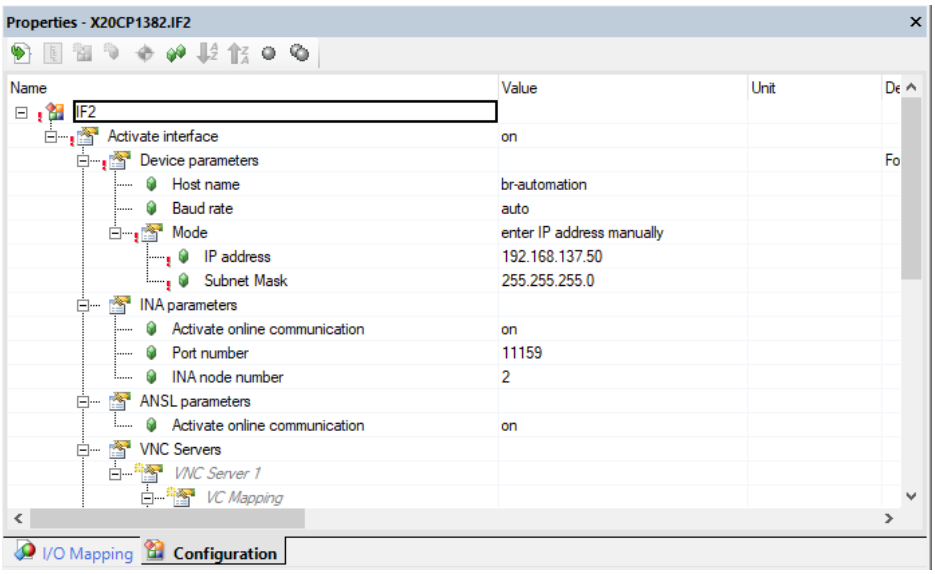



*Figura 9. Programa Automation Studio. Properties Window.*

### *4.3. Configuración del programa*

A continuación, se explicará en detalle ciertas partes del proceso de configuración del proyecto en la aplicación que se consideran importantes para la compresión de éste.

#### *4.3.1. Insertar módulos de entrada/salida*

Los módulos de entrada/salida requeridos en el banco de ensayo se añadirán de la misma manera que el resto de los elementos. Se escogerán desde el *Hardware Catalog* y se añadirán en la *Physical View*. Una vez añadidos al proyecto, el programa permite asignar variables a cada entrada/salida seleccionando "I/O Mapping" en el canal correspondiente dentro de la *Physical View.*

A continuación, se muestra la pestaña "I/O Mapping" del canal que contiene las salidas conectadas a los servomotores de este trabajo. A cada una de las salidas conectadas a un motor se le ha asignado una variable que será utilizada para habilitar su salida correspondiente.

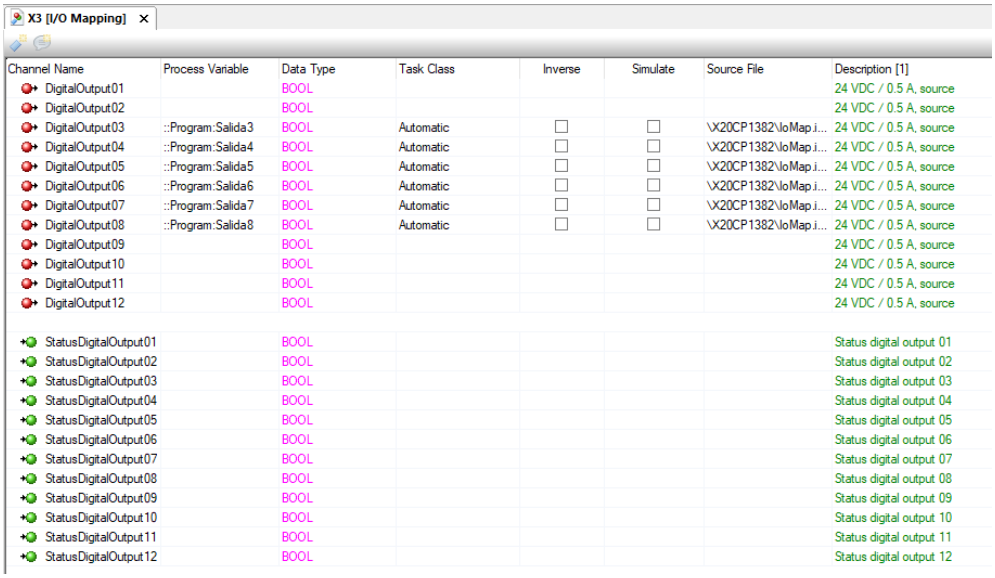



*Figura 10. Programa Automation Studio. I/O Mapping del canal de entrada/salida que contiene los servomotores.*

### *4.3.2. Dirección IP*

Dentro de la *Physical View*, también se requiere configurar la interfaz de red de control, ya que el Automation Studio necesita una red para la comunicación con el autómata. Será la interfaz Ethernet la que se encargue de la relación entre PC y autómata, debiendo modificar su configuración para que la dirección IP de ambas sea igual.

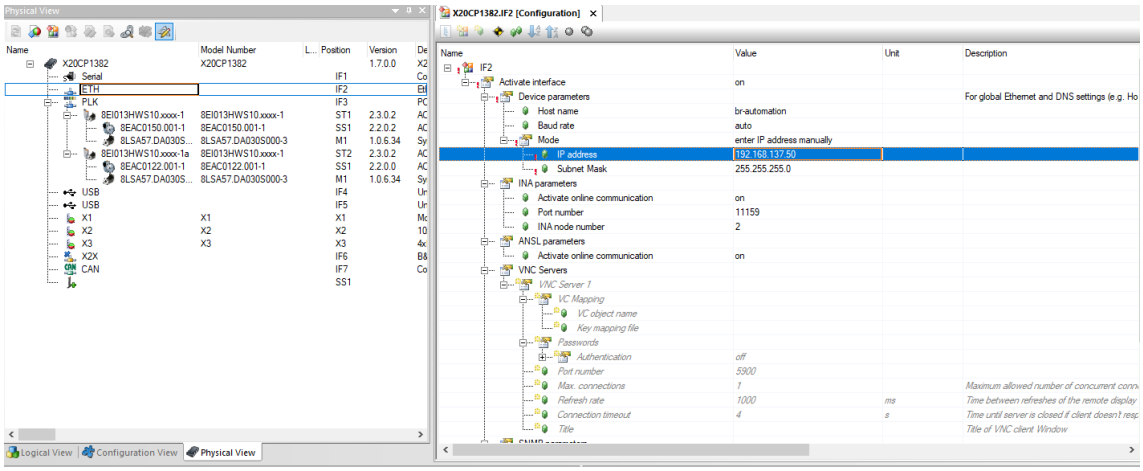



*Figura 11. Programa Automation Studio. Modificación del parámetro de la dirección IP.*

La dirección IP para este proyecto se ha modificado respecto a la utilizada por defecto, debido a la necesidad de tener conexión a internet para generar las gráficas en el servidor web, dándole el valor de 192.168.137.50 y una Subnet Mask de 255.255.255.0. Esta IP se ha escogido para que coincidiese con la del ordenador que hace de puente entre la intranet UPV y la LAN del banco de ensayo, la cual también hemos modificado previamente al crear una red compartida desde el ordenador.

### *4.3.3. Conexión entre PC y autómata*

Una vez configurada la interfaz de red de control, se requerirá establecer conexión con el PLC. Para ello, en el menú en la parte superior del programa se entrará a “Online” y después a “Settings”. En esta ventana, se podrá añadir una nueva conexión especificando sus parámetros o utilizar la función de búsqueda en la red y seleccionar la CPU con la que se quiere trabajar.

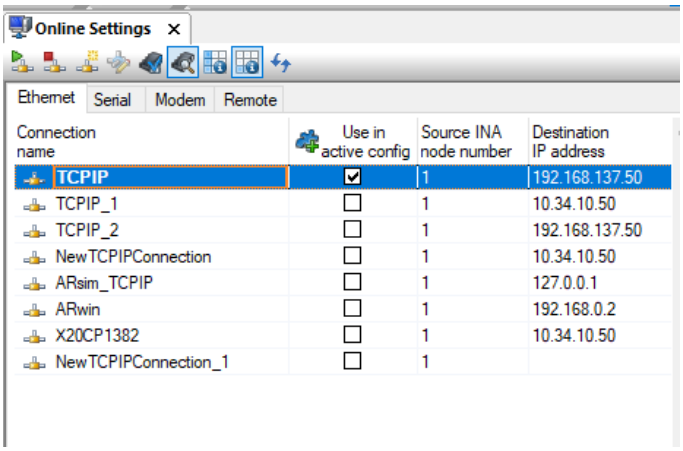



*Figura 12. Programa Automation Studio. Online Settings.*

Como se puede observar, en este proyecto aparecen la conexión con la dirección IP por defecto, la conexión con el parámetro modificado y una serie de conexiones que son posibles utilizarlas si se quiere simular el proyecto. Se ha seleccionado la que contiene la IP que coincide con la que se ha configurado previamente. Finalmente, se puede comprobar que se realiza la conexión entre PC y autómata, ya que su estado se muestra en la barra de estado situada en la parte inferior derecha de la pantalla.

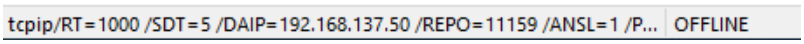



*Figura 13. Programa Automation Studio. Estado de desconexión entre PC y autómata.*

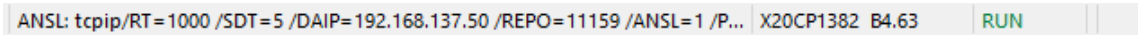



*Figura 14. Programa Automation Studio. Estado de conexión entre PC y autómata.*

### *4.3.4. Instalación del proyecto*

El último paso de la configuración del proyecto simplemente consiste en transferir este al autómata. Dentro de "Project" en el menú superior, se elegirá "Project Installation" y se seleccionará "Transfer to Target". También es más sencillo apretar en su icono correspondiente en la parte superior de la ventana principal.

*Figura 15. Programa Automation Studio. Menú de iconos.*

En primer lugar, el programa se compilaría y si existiera algún error o advertencia la mostraría por la ventana de salida, *Output Window*. Si se compila con éxito, tras unos segundos, el programa ya estaría listo para el comienzo de la comunicación e intercambio de datos con el usuario para la ejecución de las diferentes acciones de los servomotores.

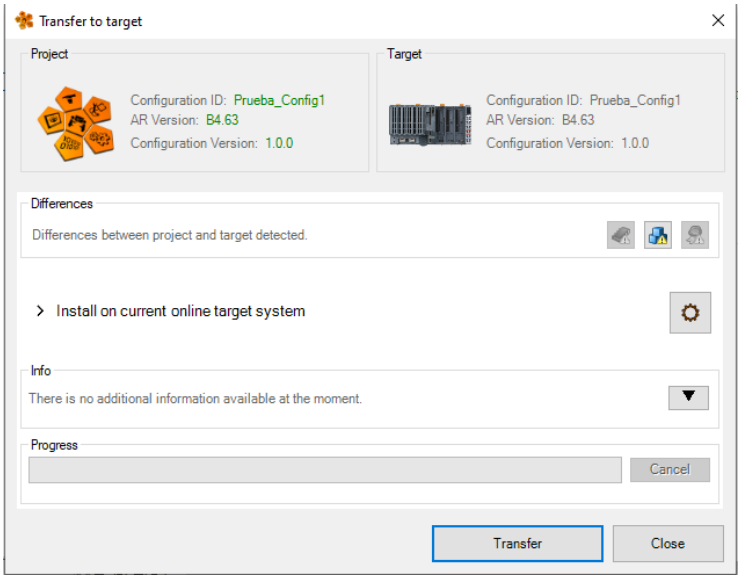



*Figura 16. Programa Automation Studio. Transferencia del proyecto al autómata.*

### *4.4. Funcionamiento básico del Automation Studio*

El Automation Studio es capaz de trabajar de forma modular para facilitar el trabajo e incluso hacerlo de una manera más eficiente. Es por eso por lo que, para facilitar la realización de este proyecto, se ha dividido en cuatro módulos distintos, es decir, cuatro códigos diferenciados que se encargan de hacer las diferentes funciones que se han planteado en los objetivos para el Automation Studio. En primer lugar, están el *Motion 1* y el *Motion 2,* que se basan en la realización de las acciones de cada servomotor respectivamente. También existe un módulo *Server* que describe el funcionamiento del autómata ante la comunicación TCP/IP con el cliente y, por último, un programa principal que enlaza el resto.

Cada módulo está formado por tres programas distintos: *init*, *cyclic* y *exit*. El primero se hará una sola vez al iniciarse el módulo correspondiente, en el cual se suele inicializar las variables. El programa *cyclic* constará de las diferentes acciones que se desea realizar, las cuales serán ejecutadas de manera cíclica, una y otra vez. El módulo también podrá contar con un código de salida, *exit*, el cual se hará una sola vez, cuando se quiera terminar de ejecutar. A continuación, se explicará cada uno detalladamente.

### *4.5. Motion 1*

Cabe destacar que la mayor parte de los dos programas *motion* han sido provistos por el fabricante, en este caso la empresa B&R. Por tanto, solo se explicará con detalle las partes que participan activamente en el funcionamiento del proyecto.

En primer lugar, el módulo *motion 1* es el encargado de ejecutar las operaciones del primer motor, el cual trabaja en modo control de velocidad. En el programa de inicialización simplemente se da valores a los parámetros de control de velocidad, aceleración y deceleración. Serán a los que funcionará el servomotor si no se reemplaza por nuevos datos de velocidad. Además, se inicializará la variable *AxisStep* al estado de espera.

```
PROGRAM _INIT

// Axis reference:
Axis1Obj := ADR(gAxis01);

AxisStep := STATE_WAIT; (* start step *) //estado inicial de espera

//Inicialización de parámetros que más tarde pueden ser modificados
BasicControl.Parameter.Velocity             := 1000;  //velocity for movement
BasicControl.Parameter.Acceleration         := 5000;  //acceleration for movement
BasicControl.Parameter.Deceleration         := 5000;  //deceleration for movement
BasicControl.Parameter.JogVelocity          := 400;   //velocity for jogging
END_PROGRAM
```

*Código 1. Programa Automation Studio. Inicialización del Motion 1.*

Una vez se entra al programa *cyclic,* se distinguen tres fases. La primera fase se basa en la lectura del estado del motor, en esta es especialmente importante la lectura de la velocidad, ya que se guardará el valor de la velocidad a la que se mueve el motor en ese momento. Este valor será el que, más tarde, el autómata convertirá y enviará al ESP32 para que lo muestre al usuario mediante una gráfica en el servidor web. El código siguiente se basa en esa lectura de la velocidad actual, se lee el valor mediante la función *MC_ReadActualVelocity* y, si es válido, se guarda en la variable *ActualVelocity*.

```
(******************** MC_READACTUALVELOCITY **********************)
MC_ReadActualVelocity_0.Enable := (NOT(MC_ReadActualVelocity_0.Error));
MC_ReadActualVelocity_0.Axis := Axis1Obj;
MC_ReadActualVelocity_0();
IF(MC_ReadActualVelocity_0.Valid = TRUE)THEN
    BasicControl.Status.ActVelocity := MC_ReadActualVelocity_0.Velocity;
    ActualVelocity := BasicControl.Status.ActVelocity;
    //le mandamos el valor actual de la velocidad a la variable ActualVelocity
END_IF
```

*Código 2. Programa Automation Studio. Lectura de la velocidad actual del servomotor 1.*

En una segunda parte, se define cada posible estado del motor mediante los parámetros necesarios. Mediante una estructura CASE, se utiliza la variable *AxisStep* para conocer el estado en el que se encuentra el autómata y pasar al siguiente cuando corresponda. A continuación, se explicará cada uno de los que normalmente se utilizan en el orden para el correcto funcionamiento del servomotor.

Al principio de este módulo se ha inicializado la variable *AxisStep* al estado de espera. Este se basa en reiniciar todos los parámetros que actúan en el movimiento del motor, de manera que estén a 0 cuando se enciende por primera vez el motor. El autómata se quedará continuamente en este estado hasta que se apriete el botón de POWER, por lo que la variable *BasicControl.Command.Power* se hará verdadera y se pasará al estado de POWER-ON.

```
CASE AxisStep OF

    (******************** WAIT *************************)
    STATE_WAIT:  (* STATE: Wait -> AxisStep=0; *) //
        IF (BasicControl.Command.Power = TRUE) THEN
            AxisStep := STATE_POWER_ON;   // Si le mandamos la orden POWER, se pasará al estado de POWER ON
        ELSE
            MC_Power_0.Enable := FALSE;
        END_IF

        (* reset all FB execute inputs we use *)
        MC_Home_0.Execute := FALSE;
        MC_Stop_0.Execute := FALSE;
        MC_MoveAbsolute_0.Execute := FALSE;
        MC_MoveAdditive_0.Execute := FALSE;
        MC_MoveVelocity_0.Execute := FALSE;
        MC_Halt_0.Execute := FALSE;
        MC_ReadAxisError_0.Acknowledge := FALSE;
        MC_Reset_0.Execute := FALSE;

        (* reset user commands *)
        BasicControl.Command.Stop := FALSE;
        BasicControl.Command.Halt := FALSE;
        BasicControl.Command.Home := FALSE;
        BasicControl.Command.MoveJogPos := FALSE;
        BasicControl.Command.MoveJogNeg := FALSE;
        BasicControl.Command.MoveVelocity := FALSE;
        BasicControl.Command.MoveAbsolute := FALSE;
        BasicControl.Command.MoveAdditive := FALSE;

        BasicControl.Status.ErrorID := 0;
```

*Código 3. Programa Automation Studio. Estado de espera del servomotor 1.*

Una vez en el segundo estado, si el motor está encendido correctamente, pasará automáticamente al estado de READY, el cual está preparado para recibir la orden del usuario y generar el movimiento que este desee. Si hubiera algún error durante la ejecución, se pasaría al estado de error.

```
    (********************* POWER ON ***********************)
STATE_POWER_ON:  (* STATE: Power on -> AxisStep=1 *)
    MC_Power_0.Enable := TRUE;
    IF (MC_Power_0.Status = TRUE) THEN
        AxisStep := STATE_READY;//Verifica que el motor esté encedido y nos manda al estado de READY
    END_IF
    (* if a power error occured go to error state *)
    IF (MC_Power_0.Error = TRUE) THEN
        BasicControl.Status.ErrorID := MC_Power_0.ErrorID;
        AxisStep := STATE_ERROR;
    END_IF
```

*Código 4. Programa Automation Studio. Estado de POWER ON del servomotor 1.*

El estado READY, es aquel donde el autómata ya está preparado para recibir la orden y ejecutar la acción que el usuario desde su interfaz envía. De esta manera, se quedará en este estado hasta que se active el botón de la acción que se desee, momento en el cual pasará al estado que se encarga de realizar el movimiento correspondiente.

```
    (********************* READY ***********************)
STATE_READY:  (* STATE: Waiting for commands ->AxisStep=10 *)

    IF (BasicControl.Command.Home = TRUE)THEN
        BasicControl.Command.Home := FALSE;
        AxisStep := STATE_HOME;

    ELSIF (BasicControl.Command.Stop = TRUE) THEN
        AxisStep := STATE_STOP;

    ELSIF (BasicControl.Command.MoveJogPos = TRUE) THEN
        AxisStep := STATE_JOG_POSITIVE;

    ELSIF (BasicControl.Command.MoveJogNeg = TRUE) THEN
        AxisStep := STATE_JOG_NEGATIVE;

    ELSIF (BasicControl.Command.MoveAbsolute = TRUE) THEN
        BasicControl.Command.MoveAbsolute := FALSE;
        AxisStep := STATE_MOVE_ABSOLUTE;

    ELSIF (BasicControl.Command.MoveAdditive = TRUE) THEN
        BasicControl.Command.MoveAdditive := FALSE;
        AxisStep := STATE_MOVE_ADDITIVE;

    ELSIF (BasicControl.Command.MoveVelocity = TRUE) THEN
        BasicControl.Command.MoveVelocity := FALSE;
        AxisStep := STATE_MOVE_VELOCITY;

    ELSIF (BasicControl.Command.Halt = TRUE) THEN
        BasicControl.Command.Halt := FALSE;
        AxisStep := STATE_HALT;
    ELSIF (BasicControl.Command.PowerOff) THEN
        BasicControl.Command.PowerOff := FALSE;
        AxisStep := STATE_POWER_OFF;
    END_IF
```

*Código 5. Programa Automation Studio. Estado READY del servomotor 1.*

Llegados a este punto, para que funcione el servomotor es necesario pasar por un estado de HOME, o búsqueda de la posición de referencia, apretando al botón de homing en el servidor web por parte del usuario. En este estado simplemente se ejecutará el *HomingMode* y una vez este termine, volverá al estado de READY para esperar a recibir la siguiente orden de movimiento.

```
    (******************** HOME **********************)
STATE_HOME:  (* STATE: start homing process -> AxisStep=2*)
    MC_Home_0.Position := BasicControl.Parameter.HomePosition;
    MC_Home_0.HomingMode := BasicControl.Parameter.HomeMode;
    MC_Home_0.Execute := TRUE;
    IF (MC_Home_0.Done = TRUE) THEN
        MC_Home_0.Execute := FALSE;
        AxisStep := STATE_READY;//Volverá automaticamente al estado de READY una vez ejecutado
    END_IF
    (* if a homing error occured go to error state *)
    IF (MC_Home_0.Error = TRUE) THEN
        MC_Home_0.Execute := FALSE;
        BasicControl.Status.ErrorID := MC_Home_0.ErrorID;
        AxisStep := STATE_ERROR;
    END_IF
```

*Código 6. Programa Automation Studio. Estado HOME del servomotor 1.*

En este momento, el cliente apretaría al botón de ON, el cual inicia el *BasicControl.Command.MoveVelocity*, pasando al estado de movimiento de velocidad. Aquí, se da los valores necesarios a los parámetros y se ejecuta el movimiento, que en este caso es que se empiece a mover el motor a la velocidad que previamente le hemos impuesto. Seguirá moviéndose hasta que apretemos al botón que lo para, pasando al estado de HALT. Al igual que en otros estados, si surgiera algún error durante la puesta en marcha del servomotor, se pasaría al estado de error.

```
    (******************** START VELOCITY MOVEMENT **********************)
STATE_MOVE_VELOCITY:  (* STATE: Start velocity movement -> AxisStep=16 *)
    MC_MoveVelocity_0.Velocity      := BasicControl.Parameter.Velocity;
    MC_MoveVelocity_0.Acceleration  := BasicControl.Parameter.Acceleration;
    MC_MoveVelocity_0.Deceleration  := BasicControl.Parameter.Deceleration;
    MC_MoveVelocity_0.Direction     := BasicControl.Parameter.Direction;
    MC_MoveVelocity_0.Execute := TRUE;
    (* check if commanded velocity is reached *)
    IF (BasicControl.Command.Halt) THEN
        //Si le llega la orden de parar, dejará de moverse el motor y pasará al estado de HALT
        BasicControl.Command.Halt := FALSE;
        MC_MoveVelocity_0.Execute := FALSE;
        AxisStep := STATE_HALT;
    ELSIF (MC_MoveVelocity_0.InVelocity = TRUE) THEN
        MC_MoveVelocity_0.Execute := FALSE;
        AxisStep := STATE_READY;
    END_IF
    (* check if error occured *)
    IF (MC_MoveVelocity_0.Error = TRUE) THEN
        BasicControl.Status.ErrorID := MC_MoveVelocity_0.ErrorID;
        MC_MoveVelocity_0.Execute := FALSE;
        AxisStep := STATE_ERROR;
    END_IF
```

*Código 7.Programa Automation Studio. Estado de inicio de movimiento del servomotor 1.*

Cuando se desee parar el servomotor en movimiento, el usuario apretaría en el botón de OFF y la variable *AxisStep* pasaría al estado de HALT, como hemos visto en el código anterior. En este estado, el autómata parará el motor y, una vez cese completamente el movimiento, se volverá al estado de READY, preparado para realizar la siguiente orden.

```
    (**********************HALT_MOVEMENT**************************)
STATE_HALT:   (* STATE: Halt movement *) // Para el motor y vuelve a STATE_READY para realizar otra acción
    MC_Halt_0.Deceleration := BasicControl.Parameter.Deceleration;
    MC_Halt_0.Execute := TRUE;
    IF (MC_Halt_0.Done = TRUE) THEN
        MC_Halt_0.Execute := FALSE;
        AxisStep := STATE_READY; //Una vez el motor está parado, vuelve al estado de READY
    END_IF
    (* check if error occured *)
    IF (MC_Halt_0.Error = TRUE) THEN
        BasicControl.Status.ErrorID := MC_Halt_0.ErrorID;
        MC_Halt_0.Execute := FALSE;
        AxisStep := STATE_ERROR;
    END_IF
```

*Código 8. Programa Automation Studio. Estado de HALT del servomotor 1.*

Por último, cabe destacar el estado de error que se ha nombrado numerosas veces anteriormente. Este reconocerá de qué tipo de error se trata y mandará un mensaje al usuario. Una vez informado del error, el cliente tendrá que apretar el botón de ERROR, el cual activa la variable *BasicControl.Command.ErrorAcknowledge* para avisar al autómata que se conoce el error. Así, una vez pulsado y si no sigue existiendo ese error, se volvería al estado de espera inicial.

```
    (******************** FB-ERROR OCCURED ************************)
STATE_ERROR:  (* STATE: Error *)
    (* check if FB indicates an axis error -> AxisStep=100*)
    IF (MC_ReadAxisError_0.AxisErrorCount<>0) THEN
        AxisStep := STATE_ERROR_AXIS;
    ELSE
        IF (BasicControl.Command.ErrorAcknowledge = TRUE) THEN
            BasicControl.Command.ErrorAcknowledge := FALSE;
            BasicControl.Status.ErrorID := 0;
            (* reset axis if it is in axis state ErrorStop *)
            IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid = TRUE)) THEN
                AxisStep := STATE_ERROR_RESET;
            ELSE
                AxisStep := STATE_WAIT;
            END_IF
        END_IF
    END_IF
```

*Código 9. Programa Automation Studio. Estado de error del servomotor 1.*

La tercera y última fase de este módulo se trata de la llamada a las funciones dependiendo del valor del dato que nos llega por parte del usuario. Por un lado, cuando le llega al autómata un dato de velocidad, la variable *DatoCambiado_Vel* estará a 0, por lo que será capaz de entrar en la condición y modificar el parámetro de velocidad por el nuevo valor que se ha enviado. Una vez cambiada la velocidad, se iguala *DatoCambiado_Vel* a TRUE para informar que el dato ha sido modificado y poder recibir nuevos valores de velocidad.

```
IF NOT DatoCambiado_Vel THEN

    BasicControl.Parameter.Velocity := NuevaVelocidad; //le damos el nuevo valor de velocidad
    BasicControl.Command.MoveVelocity := TRUE;

    DatoCambiado_Vel := TRUE; //avisamos que el dato de velocidad ha sido cambiado

END_IF
```

*Código 10. Programa Automation Studio. Modificación del parámetro de la velocidad por el dato enviado por el usuario.*

Por otro lado, cuando le llega una acción al autómata, existe un condicional CASE que en función del valor de la variable *AccionRecibida,* ejecutará un código u otro. A este solo entrará si *AccionTerminada* es falsa, es decir, si la acción que se acaba de enviar todavía no ha sido ejecutada. De esta manera, a partir del dato de la acción recibida, se activará el comando correspondiente. En cualquier caso, al acabar de ejecutarse la acción, la variable *AccionTerminada* se igualará a 1, para avisar de que se ha terminado la acción actual y poder empezar a realizar la siguiente.

```
IF NOT AccionTerminada THEN //cuando la accion está en proceso

    CASE AccionRecibida OF //dependiendo del valor de la accion recibida realizará una orden u otra
        5:
            (************************** MC_POWER ****************************)

            IF MC_Power_0.Status THEN
                BasicControl.Command.Power := FALSE;
            ELSE
                BasicControl.Command.Power := TRUE;
            END_IF

            AccionTerminada := TRUE; //la acción ha acabado de ejecutarse

        7:
            (************************** MC_HOME ****************************)

            BasicControl.Command.Home := TRUE;
            AccionTerminada := TRUE;

        9:
            (************************** MC_RESET ****************************)

            BasicControl.Command.ErrorAcknowledge := TRUE;

            AccionTerminada := TRUE;

         11:
             (**************************MC_HALT*****************************)

             BasicControl.Command.Halt := TRUE;

             AccionTerminada := TRUE;

         12:
             (********************** MC_MOVEVELOCITY ************************)

             BasicControl.Command.MoveVelocity := TRUE;

             AccionTerminada := TRUE;

        ELSE
            // nothing
    END_CASE

END_IF
```

*Código 11. Programa Automation Studio. Ejecución de acciones del servomotor 1 en función del valor recibido.*

Por último, se almacenan los valores de *AxisStep* y *BasicControl.Command.Power* en variables globales a las cuales se les ha llamado *Paso* y *Power*, respectivamente. Estas serán de gran utilidad en el módulo general para encender o apagar las salidas en función del botón que el usuario apriete durante el proceso de puesta en marcha de los servomotores.

```
Paso := AxisStep;
Power := BasicControl.Command.Power;
```

*Código 12. Programa Automation Studio.*
*Almacenamiento de valores en variables globales.*

### *4.6. Motion 2*

A continuación se describirá el módulo *motion 2*, encargado del movimiento del segundo servomotor que funciona en modo control de par. Al igual que el módulo *motion 1*, la base de este código ha sido provista por el fabricante B&R añadiendo partes específicas para el correcto funcionamiento del proyecto. En general, el programa es muy parecido al descrito para el primer servomotor, salvo la adición del parámetro de par.

En primer lugar, cuenta con un programa de inicialización en el cual se definen los diferentes parámetros de movimiento y se inicializa, en este caso, el segundo motor a un estado de espera.

```
PROGRAM _INIT

// Axis reference
Axis2Obj := ADR(gAxis02);

AxisStep := STATE_WAIT; //Estado inicial de espera

BasicControl.Parameter.Velocity             := 1000;  //velocity for movement
BasicControl.Parameter.Acceleration         := 5000;  //acceleration for movement
BasicControl.Parameter.Deceleration         := 5000;  //deceleration for movement
BasicControl.Parameter.JogVelocity          := 400;   //velocity for jogging
END_PROGRAM
```

*Código 13. Programa Automation Studio. Inicialización del Motion 2.*

El programa cíclico funciona de la misma manera que el anterior módulo, ya que se pueden distinguir las mismas tres fases. La primera parte del programa trata la lectura del estado del motor. En este caso, el autómata además de medir la velocidad real en el momento, es capaz de leer el par actual del motor.

```
(********************* MC_READACTUALTORQUE **********************)
MC_ReadActualTorque_0.Enable := (NOT(MC_ReadActualTorque_0.Error));
MC_ReadActualTorque_0.Axis :=  Axis2Obj;
MC_ReadActualTorque_0();

IF(MC_ReadActualTorque_0.Valid = TRUE) THEN
        BasicControl.Status.ActTorque := MC_ReadActualTorque_0.Torque;
        ActualTorque := BasicControl.Status.ActTorque;
END_IF
```

*Código 14. Programa Automation Studio. Lectura del valor de par del servomotor 2.*

La segunda parte del programa se basa de nuevo en la definición de los distintos estados. El motor 2 seguirá el mismo proceso que el primer motor: iniciará en un estado de espera hasta que se ejecute la orden de Power. Sin embargo, tras pasar por el estado de búsqueda del punto de referencia o Homing se ejecutará la orden de iniciar la velocidad con ciertas modificaciones respecto al código anterior: para que el regulador de velocidad trabaje saturado ofreciendo a su salida un valor de par igual al límite establecido se solicita el movimiento con una velocidad nula y un valor especificado de límite de par. En cuanto el motor principal intente separar al conjunto de motores acoplados de su estado de velocidad cero con una velocidad de giro incluso aunque sea pequeña, el control de velocidad del motor secundario, que tiene una consigna de velocidad nula y un límite de par establecido por el usuario, establecerá a su salida una consigna de par igual a dicho límite, consigna que el control interno de par intentará mantener de forma indefinida.

```
(******************** START VELOCITY MOVEMENT **********************)
    STATE_MOVE_VELOCITY:  (* STATE: Start velocity movement *)
        MC_MoveVelocity_0.Velocity      := 0;
        MC_MoveVelocity_0.Acceleration  := BasicControl.Parameter.Acceleration;
        MC_MoveVelocity_0.Deceleration  := BasicControl.Parameter.Deceleration;
        MC_MoveVelocity_0.Direction     := BasicControl.Parameter.Direction;
        MC_MoveVelocity_0.Torque        := BasicControl.Parameter.Torque;
        MC_MoveVelocity_0.Execute := TRUE;
        (* check if commanded velocity is reached *)
        IF (BasicControl.Command.Halt) THEN
            BasicControl.Command.Halt := FALSE;
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_HALT;
        ELSIF (MC_MoveVelocity_0.InVelocity = TRUE) THEN
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_READY;
        END_IF
        (* check if error occured *)
        IF (MC_MoveVelocity_0.Error = TRUE) THEN
            BasicControl.Status.ErrorID := MC_MoveVelocity_0.ErrorID;
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF
```

*Código 15. Programa Automation Studio. Estado de inicio de movimiento del servomotor 2.*

Por último, se llega a la fase de llamada de las funciones en la que se añadirá parte de código propio de este proyecto. Primeramente se codificará la modificación del parámetro de par al cual se va a ejecutar el motor 2 por el nuevo valor de par que ha venido dado por el usuario, de la misma manera que se ha hecho en el módulo del primer motor con la velocidad.

```
IF NOT DatoCambiado_Par THEN

    BasicControl.Parameter.Torque := NuevoPar;
    BasicControl.Command.Torque := TRUE;
    DatoCambiado_Par := 1;

END_IF
```

*Código 16. Programa Automation Studio. Modificación del parámetro de par por el valor dado por el usuario.*

Además, se añadirá las diferentes órdenes que se pueden ejecutar para la puesta en marcha de este motor mediante una estructura CASE dependiente del valor de *AccionRecibida*, de la misma manera que en el *motion 1*, pero con los valores asociados a las órdenes del servomotor 2.

```
    CASE AccionRecibida OF
        6:
            IF MC_Power_0.Status THEN
                BasicControl.Command.Power := TRUE;
            ELSE
                BasicControl.Command.Power := FALSE;
            END_IF

            AccionTerminada := TRUE;

        8:
            BasicControl.Command.Home := TRUE;

            AccionTerminada := TRUE;

        10:
            BasicControl.Command.ErrorAcknowledge := TRUE;

            AccionTerminada := TRUE;

        14:

            BasicControl.Command.Halt := TRUE;

            AccionTerminada := TRUE;

        13:

            BasicControl.Command.MoveVelocity := TRUE;

            AccionTerminada := TRUE;


    END_CASE

END_IF
```

*Código 17. Programa Automation Studio. Ejecución de órdenes del servomotor 2 en función del valor recibido.*

### *4.7. Server*

Al igual que los dos programas anteriores, gran parte de este módulo ha sido provisto por parte de la empresa B&R y en este trabajo se ha adaptado para el envío y recepción de órdenes y estados de funcionamiento del banco de ensayos para una comunicación efectiva con el cliente TCP que se programa en el ESP32. En primer lugar, cuenta con un programa de inicialización donde se les da valores a los parámetros que se van a necesitar en el código. Entre ellos se encuentra el puerto por el cual se va a iniciar la comunicación entre el autómata y el ESP32, al igual que ciertos parámetros de tiempo que se usarán para la duración del envío y recepción de datos.

```
PROGRAM _INIT

    (* Server configuration *)
    gServer.ParaPort := 12010; (* TCP port on the server *)
    gServer.ParaSendTime := T#100ms;
    gServer.ParaReceiveTimeout := T#750ms;
```

*Código 18. Programa Automation Studio. Inicialización del módulo Server.*

Dentro del programa cíclico, el código se basa en un condicional CASE dependiente del valor de la variable *Server.sStep*, el cual va pasando por una serie de pasos para realizar la comunicación TCP. El funcionamiento es el típico de una máquina de estados: en cada estado se analiza el resultado de las acciones ejecutadas en dicho estado y, en función de ellos, se establece el siguiente estado de trabajo. En cada paso comprueba que la ejecución se ha realizado correctamente y, si no, envía un mensaje de error y cierra la comunicación.

Tras un primer paso previo de inicialización, la segunda fase se trata de la conexión al puerto por donde se va a mandar información, para abrir la comunicación. El valor del puerto ha sido dado en el proceso de iniciar las variables a través de la variable *gServer.ParaPort*, el cual, en este caso, es igual a 12010. Si se ejecuta correctamente, se pasará al paso siguiente.

```
CASE Server.sStep OF

    0:
        IF gServer.Enable THEN
            gServer.Status := 1; //Esperando la primera comunicación
            Server.sStep := 5;
        END_IF;

    5:  (* Open Ethernet Interface *) // Escucha en el puerto que hemos dicho que iba a mandar info
        Server.TcpOpen_0.enable := 1;
        Server.TcpOpen_0.pIfAddr := 0;  (* Listen on all TCP/IP Interfaces*)
        Server.TcpOpen_0.port := gServer.ParaPort;  (* Port to listen*)
        Server.TcpOpen_0.options := tcpOPT_REUSEADDR;
        (* Allows the linking of several instances of a TCP server with the same port *)
        Server.TcpOpen_0;  (* Call the Function*)

        IF Server.TcpOpen_0.status = 0 THEN  (* TcpOpen successfull*)
            Server.sStep := 6;
```

*Código 19. Programa Automation Studio. Paso de inicialización y paso de conexión al puerto de comunicación.*

Si existiera algún error durante esta acción, se pasaría al último paso que se encarga de cerrar la comunicación. Esta parte del código se repetirá en cada una de las partes de este programa, de manera que, si surge algún problema en cualquier punto del proceso de conexión, envío y recepción de datos con el cliente, se cerrará inmediatamente la comunicación.

```
ELSIF Server.TcpOpen_0.status = tcpERR_ALREADY_EXIST OR gServer.Enable = 0 THEN
    (* Log error *)
    FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
        brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));
    END_FOR;
    brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
    GetTimeStruct(enable := 1,pDTStructure := ADR(ServerLog[0].Date));
    ServerLog[0].FBK := 'TcpOpen_0';
    ServerLog[0].ErrorID := Server.TcpOpen_0.status;

    (* Close communication *)
    Server.sStep := 50;
```

*Código 20. Programa Automation Studio. Reconocimiento de errores y paso al cierre de la comunicación.*

A continuación, existe un paso de preparación para la conexión con el cliente, que simplemente comprueba que el cliente no esté ocupado. Si lo estuviera, se mantendría en este estado hasta que el usuario al que se le ha dicho que se conecte, estuviera disponible.

```
6:
    Server.linger_opt.lLinger := 0; (* linger Time = 0 *)
    Server.linger_opt.lOnOff := 1; (* linger Option ON *)

    Server.TcpIoctl_0.enable := 1;
    Server.TcpIoctl_0.ident := Server.TcpOpen_0.ident; (* Connection Ident from AsTCP.TCP_Open *)
    Server.TcpIoctl_0.ioctl := tcpSO_LINGER_SET; (* Set Linger Options *)
    Server.TcpIoctl_0.pData := ADR(Server.linger_opt);
    Server.TcpIoctl_0.datalen := SIZEOF(Server.linger_opt);
    Server.TcpIoctl_0;

    IF Server.TcpIoctl_0.status = 0 THEN  (* TcpIoctl successfull *)
        Server.sStep := 10;
    ELSIF Server.TcpIoctl_0.status = ERR_FUB_BUSY THEN  (* TcpIoctl not finished -> redo *)
        (* Busy *)
```

*Código 21. Programa Automation Studio. Paso de preparación para la conexión con cliente.*

Una vez el cliente está disponible, el autómata es capaz de conectarse a él tras ejecutar los dos pasos siguientes:

```
10: (* Wait for Client Connection *)
    Server.TcpServer_0.enable := 1;
    Server.TcpServer_0.ident := Server.TcpOpen_0.ident;  (* Connection Ident from AsTCP.TCP_Open *)
    Server.TcpServer_0.backlog := 1;  (* Number of clients waiting simultaneously for a connection*)
    Server.TcpServer_0.pIpAddr := ADR(Server.client_address);  (* Where to write the client IP-Address*)
    Server.TcpServer_0;  (* Call the Function*)

    IF Server.TcpServer_0.status = 0 THEN  (* Status = 0 if an client connects to server *)
        Server.sStep := 15;
```

*Código 22. Programa Automation Studio. Paso de espera de la conexión con el cliente.*

```
15:
    Server.TcpIoctl_0.enable := 1;
    Server.TcpIoctl_0.ident := Server.TcpServer_0.identclnt; (* Connection Ident from AsTCP.TCP_Server *)
    Server.TcpIoctl_0.ioctl := tcpSO_LINGER_SET;  (* Set Linger Options *)
    Server.TcpIoctl_0.pData := ADR(Server.linger_opt);
    Server.TcpIoctl_0.datalen := SIZEOF(Server.linger_opt);
    Server.TcpIoctl_0;

    IF Server.TcpIoctl_0.status = 0 THEN  (* TcpIoctl successfull *)
        // Aquí ya todo esta listo para el envío y la recepción de datos
        (* Start send timer *)
        TON_Send_Server.IN := 1;
        TON_Send_Server.PT := gServer.ParaSendTime;

        Server.sStep := 20;
    ELSIF Server.TcpIoctl_0.status = ERR_FUB_BUSY THEN  (* TcpIoctl not finished -> redo *)
        (* Busy *)
```

*Código 23. Programa Automation Studio. Paso de conexión con el cliente.*

Llegados a este punto, ya está todo listo para iniciarse el envío y recepción de datos con el cliente. En primer lugar, el paso de recepción de datos se basa en una función *Server.TcpRecv_0* que recibe los datos y los almacena en una variable *Server.data_buffer*. Así, en el momento en que el autómata recibe algún valor, se guarda y se marca que se ha recibido.

```
20: (* Wait for Data *)
    Server.TcpRecv_0.enable := 1;
    Server.TcpRecv_0.ident := Server.TcpServer_0.identclnt;  (* Client Ident from AsTCP.TCP_Server *)
    Server.TcpRecv_0.pData  := ADR(Server.data_buffer);  (* Where to store the incoming data *)
    Server.TcpRecv_0.datamax := SIZEOF(Server.data_buffer);  (* Length of data buffer *)
    Server.TcpRecv_0.flags := 0;
    Server.TcpRecv_0;  (* Call the Function*)

    TON_RecvTimeout_Server.IN := 1;
    TON_RecvTimeout_Server.PT := gServer.ParaReceiveTimeout;
```

*Código 24. Programa Automation Studio. Recepción de datos enviados por el ESP32.*

Si existe algún error o la conexión se interrumpe durante este proceso, se pasará al paso correspondiente que ejecuta el código que toca cuando la conexión falla.

```
        (* Connection lost or disabled *)
        Server.sStep := 40;
    ELSIF Server.TcpRecv_0.status = 0 THEN  (* Data received *)
        gServer.Status := 0; //COMMUNICATION OK
        (* Reset timeout timer, since data was received *)
```

*Código 25. Programa Automation Studio. Confirmación de que la comunicación ha sido realizada con éxito y los datos recibidos correctamente. Y si no es así, paso al cierre de la conexión.*

En cambio, si se ha recibido los datos correctamente, se pasa a la parte del código ya añadido para este trabajo específicamente, la cual se basa en guardar los datos recibidos en una variable propia *ValorRecibido*.

```
ValorRecibido := UDINT_TO_UINT(Server.data_buffer[0]);
//el valor recibido se almacena en la variable ValorRecibido


IF ValorRecibido.15 THEN //distingue si es una orden o un dato nuevo de velocidad o par

    ValorRecibido.15:=FALSE;
    DatoRecibido := ValorRecibido;  //si es un nuevo dato, pasa el valor a la variable DatoRecibido
    DatoCambiado := 0;

ELSIF NOT ValorRecibido.15 AND AccionTerminada THEN
    //si manda una acción, esta solo podrá ejecutarse si la acción anterior ha terminado

    AccionRecibida := ValorRecibido; //pasa el valor recibido a otra variable que lo reconoce como acción
    AccionTerminada :=FALSE; //la acción que acaba de llegar ha iniciado

END_IF
```

*Código 26. Programa Automation Studio. Diferenciación del tipo de dato y almacenamiento en la variable correspondiente.*

Existen dos opciones respecto a este valor que envía el usuario: que se trate de un dato que representa una acción o un valor nuevo de velocidad o par. Para diferenciarlos, se ha utilizado el bit de mayor peso, de manera que, si éste es 1, se trate de un valor nuevo de velocidad o par, y si es 0, represente una acción a ejecutar por el autómata. Una vez distinguido el tipo de dato, este se guarda en la variable correspondiente, además de avisar al autómata de que ha llegado un nuevo dato o una nueva acción.

Si no se recibe ningún dato en un tiempo límite establecido, se entenderá que el cliente ha pasado a esperar respuesta por parte del autómata, por tanto, se pasaría al envío de datos.

```
    TON_RecvTimeout_Server.IN := 0;

    (* Time for new communication *)
    IF TON_Send_Server.Q THEN
        TON_Send_Server.IN := 0;
        (* Go to send step *) // Si no se ha recibido nada en un tiempo limite pasamos a leer.
        Server.sStep := 30;
    END_IF;
ELSIF Server.TcpRecv_0.status = tcpERR_NO_DATA THEN
    (* No data received - wait for data, timeout or new communication.
       The client will send data again, while the RecvTimeout timer is
       still counting. If this timeout is reached, an entry will be
       written in the communication logger. *)

    IF TON_Send_Server.Q THEN
        (* Reset communication timer *)
        TON_Send_Server.IN := 0;
        (* Go to send step *)
        Server.sStep := 30;
    END_IF;
```

*Código 27. Programa Automation Studio. Comprobación de que no se reciben datos en un tiempo límite y paso a enviar valores al cliente.*

A través de la variable *SentData*, en la cual introducimos los valores de velocidad que queremos mostrar al usuario, utilizaremos la función *Server.TcpSend_0* para enviarlos de vuelta al ESP32.

```
30: (* Send Data back to Client *)  // UaCounter será nuestra manera de introducir data
    Server.TcpSend_0.enable := 1;
    Server.TcpSend_0.ident := Server.TcpServer_0.identclnt;  (* Client Ident from AsTCP.TCP_Server *)

    SentData := VelocidadReal; //mandaremos la velocidad ya convertida

    Server.TcpSend_0.pData := ADR(SentData);  (* Which data to send *)
    Server.TcpSend_0.datalen := SIZEOF(SentData);  (* Lenght of data to send *)
    Server.TcpSend_0.flags := 0;
    Server.TcpSend_0;  (* Call the Function*)
```

*Código 28. Programa Automation Studio. Envío de datos por parte del autómata al cliente.*

Si existieran ciertos datos que no ha sido posible enviar en el propio paso anterior, el autómata saltaría a un paso extra que se encarga justo de eso, de enviar los valores que han quedado sin enviar.

```
        (* Calculate leftover data *)
        pLeftoverData := Server.TcpSend_0.pData + Server.TcpSend_0.sentlen;
        LeftoverDataSize := Server.TcpSend_0.datalen - Server.TcpSend_0.sentlen;

        (* Go to a new Send state, where leftover data will be sent *)
        Server.sStep := 35;

 35:
    (* Sent leftover data *)
    Server.TcpSend_0.enable := 1;
    Server.TcpSend_0.ident := Server.TcpServer_0.identclnt;  (* Client Ident from AsTCP.TCP_Server *)
    Server.TcpSend_0.pData := pLeftoverData;  (* Which data to send *)
    Server.TcpSend_0.datalen := LeftoverDataSize;  (* Lenght of data to send *)
    Server.TcpSend_0.flags := 0;
    Server.TcpSend_0;  (* Call the Function*)
```

*Código 29. Programa Automation Studio. Envío de datos que se han quedado sin enviar.*

Una vez se envían todos los valores que se quiere hacer llegar al microcontrolador, se llega al paso que cierra la conexión con el cliente a partir de la función *Server.TcpClose_0.* Una vez cerrada la conexión, tanto por parte del cliente como por parte del autómata, se volvería al paso inicial a la espera de una nueva comunicación.

```
40:
    Server.TcpClose_0.enable := 1;
    Server.TcpClose_0.ident := Server.TcpServer_0.identclnt;
    Server.TcpClose_0.how := 0;; // tcpSHUT_RD OR tcpSHUT_WR;
    Server.TcpClose_0;

    IF Server.TcpClose_0.status = 0 THEN
        Server.sStep := 50;
 50:
     Server.TcpClose_0.enable := 1;
     Server.TcpClose_0.ident := Server.TcpOpen_0.ident;
     Server.TcpClose_0.how := 0; //tcpSHUT_RD OR tcpSHUT_WR;
     Server.TcpClose_0;

     IF Server.TcpClose_0.status = 0 THEN
         Server.sStep := 0;
```

*Código 30. Programa Automation Studio. Cierre de la conexión entre cliente y autómata.*

Por último, se ejecutará un programa de salida, en el caso de que se apagara el programa sin haber terminado de cerrar las comunicaciones entre cliente y autómata.

```
PROGRAM _EXIT

    REPEAT
        Server.TcpClose_0.enable := 1;
        Server.TcpClose_0.ident := Server.TcpServer_0.identclnt;
        Server.TcpClose_0.how := 0; //tcpSHUT_RD OR tcpSHUT_WR;
        Server.TcpClose_0;

        UNTIL
        Server.TcpClose_0.status <> ERR_FUB_BUSY
    END_REPEAT;

    REPEAT
        Server.TcpClose_0.enable := 1;
        Server.TcpClose_0.ident := Server.TcpOpen_0.ident;
        Server.TcpClose_0.how := 0; //tcpSHUT_RD OR tcpSHUT_WR;
        Server.TcpClose_0;

        UNTIL
        Server.TcpClose_0.status <> ERR_FUB_BUSY
    END_REPEAT;

END_PROGRAM
```

*Código 31. Programa Automation Studio. Salida del módulo Server.*

### *4.8. Main Program*

Como ya se ha mencionado previamente, el programa general se encarga de enlazar los diferentes módulos para asegurar el correcto funcionamiento del proyecto. En primer lugar, en el programa *init* se inicializan las variables que actúan en el módulo principal.

```
PROGRAM _INIT
    // Inicialización de variables
    DatoRecibido := 0;
    DatoCambiado_Vel := 0;
    DatoCambiado_Par := 0;
    NuevoPar := 0;
    NuevaVelocidad := 0;
    Salida3:=FALSE;
    Salida4:=FALSE;
    Salida5:=FALSE;
    Salida6:=FALSE;
    Salida7:=FALSE;
    Salida8:=FALSE;
    AccionTerminada:=TRUE;
    Power := FALSE;
    Paso := 0;

END_PROGRAM
```

*Código 32. Programa Automation Studio. Inicialización del módulo general.*

Una vez dentro del programa cíclico, se actualizan los datos de velocidad y par de los dos servomotores. Como se ha explicado antes, el autómata lee los datos que le envía el ESP32 y los guarda en la variable *DatoRecibido,* además de que pone a 0 una variable llamada *DatoCambiado.* Así, si el dato todavía no ha sido actualizado, se reconocerá si se trata de una nueva velocidad o un par nuevo, y se guardará el valor en la variable correspondiente, *NuevoPar* y *NuevaVelocidad*. Una vez se ha reemplazado el valor se igualan a 0 las variables *DatoCambiado_Par* o *DatoCambiado_Vel*. Por otro lado, *DatoCambiado* será 1 hasta que se reciba un nuevo valor de parte del usuario

```
PROGRAM _CYCLIC


    //Actualización de datos de par y velocidad

    IF NOT DatoCambiado THEN //cuando llega un nuevo dato la variable DatoCambiado estará a 0

        DatoCambiado := 1;

        IF DatoRecibido.14 THEN //para distinguir si es dato de par o velocidad

            DatoRecibido.14:=FALSE; //nos quedamos con el propio valor
            NuevoPar := MAX_PAR*DatoRecibido/16384;
            DatoCambiado_Par := 0;

        ELSIF NOT DatoRecibido.14 THEN

            NuevaVelocidad := MAX_VEL/16384*DatoRecibido*100/6;
            DatoCambiado_Vel := 0;

        END_IF

    END_IF
```

*Código 33. Programa Automation Studio. Modificación de los valores de velocidad y par por los nuevos recibidos.*

Es necesario explicar la manera de distinguir si el dato recibido se trata de una velocidad o un par. Al igual que en el caso de la distinción entre dato o acción que se ha explicado en el apartado anterior, los datos que le llegan al autómata están formados por 16 bits. El bit de mayor importancia, es decir el número 15, ya ha sido utilizado para diferenciar los nuevos valores de par y velocidad de las acciones, por tanto, se utilizará el bit número 14 para saber si se trata de un dato de velocidad o de par. Si el bit catorceavo es 1, se trata de un nuevo valor de par y si, por el contrario, es 0 se tratará de un dato de velocidad. Una vez ya se sabe de qué tipo de dato se trata, se multiplica el propio valor por unas constantes que generan el valor digital que queremos, ajustando la escala entre el rango seleccionado para el envío y el rango correspondiente a la variable de que se trate, velocidad o par.
En el caso de la velocidad, el programa Automation Studio trabaja en revoluciones por segundo por 1000 como unidades, por lo que habrá que convertir el valor medido en revoluciones por minuto. Simplemente basta con multiplicarlo por 100/6.

La siguiente parte de la que se encarga el programa general es habilitar y deshabilitar los dos servomotores. Para ello, se ha creado dos variables (*Salida3* y *Salida4*) que cuando están encendidas habilitan la propia salida donde se encuentran los motores 1 y 2, respectivamente.

```
//  Habilitación de servomotores 1 y 2

IF AccionRecibida = 1 THEN
    //cuando la acción recibida sea un 1, se encenderá la salida 3 que corresponde al motor 1

    Salida3:=TRUE;
    AccionTerminada:=TRUE; //una vez encendemos la salida, la acción ha terminado

ELSIF AccionRecibida = 2 THEN
    //cuando la acción recibida sea un 2, se encenderá la salida 4 que corresponde al motor 2

    Salida4:=TRUE;
    AccionTerminada:=TRUE;

END_IF
```

*Código 34. Programa Automation Studio. Habilitación de los servomotores.*

En cambio, cuando están a 0, deshabilitan las salidas conectadas a los servomotores, impidiendo su funcionamiento.

```
// Deshabilitacion de servomotores 1 y 2

IF AccionRecibida = 3 THEN
    //cuando la acción recibida sea un 3, se apagará la variable salida 3 que corresponde al motor 1

    Salida3:=FALSE;
    AccionTerminada:=TRUE;

ELSIF AccionRecibida = 4 THEN
    //cuando la acción recibida sea un 4, se apagará la variable salid a4 que corresponde al motor 2

    Salida4:=FALSE;
    AccionTerminada:=TRUE;

END_IF
```

*Código 35. Programa Automation Studio. Deshabilitación de los servomotores.*

La variable *AccionRecibida* contiene el valor que ha sido recibido, el cual ya se conoce que se trata de una acción a realizar. Previamente se ha decidido qué número corresponde a cada acción, de manera que cuando se reciba un 1 o un 2 se encienda la salida de habilitación del motor correspondiente, y cuando se reciba un 3 o un 4 se apague. Una vez se ha finalizado la acción, se iguala a 1 la variable *AccionTerminada*. Esta se utiliza para que no existan dos órdenes de dos acciones distintas a la vez, ya que lo más seguro sería que una de las dos no se completara.

Por otro lado, como ya se ha explicado anteriormente, se va a habilitar otras salidas para mostrar información de manera visual al usuario. Así, dependiendo de la acción que éste desee que el autómata ejecute, se encenderá o apagará la salida correspondiente.

En primer lugar, la salida número cinco está asociada al botón de *Power*, de manera que, si el usuario lo aprieta y pasa al estado de *Power On*, la salida estará encendida. En cambio, cuando se quiera pasar al estado de *Power Off* se deberá volver a apretar el botón de *Power*, apagando la misma salida. Se utilizará la variable global *Power* que almacena el valor del parámetro del comando Power, es decir, será TRUE cuando se esté en el estado *Power On* y FALSE en el estado de *Power Off.*

```
// Habilitación y deshabilitación de las salidas en función del valor dado por el usuario
IF (Power = TRUE) THEN // Si el comando de POWER está encendido, se encenderá la salida 5
    Salida5:=TRUE;

ELSIF (Power = FALSE) THEN // Si el comando de POWER está apagado, se apagará la salida 5
    Salida5:=FALSE;
END_IF
```

*Código 36. Programa Automation Studio. Vinculación de la salida 5 y el botón de Power.*

De la misma manera se ha codificado el estado de *Homing*, pero con ayuda de la variable *Paso,* la cual contiene el valor que se corresponde al número de paso en el que el autómata se encuentra durante el proceso de la puesta en marcha del servomotor 1. En el momento en el que el PLC entre en el estado de Home la salida 6 se encenderá, y, una vez sale de este paso, se apagará.

```
IF (Paso = 2) THEN // Cuando se encuentre en el paso de HOME se encenderá la salida 6
    Salida6 := TRUE;
ELSIF NOT (Paso = 2) THEN // Una vez ejecutado el estado HOME se apagará la salida 6
    Salida6 := FALSE;
END_IF
```

*Código 37. Programa Automation Studio. Vinculación de la salida 6 al estado de Homing*

En cuento a la salida 7, estará relacionada con el propio movimiento del servomotor. Mientras el motor se encuentre girando, la salida estará encendida. En el momento que se apriete al botón que cesa el movimiento, la salida se apagará.

```
IF AccionRecibida = 12 THEN //Cuando la acción recibida sea un 12, se encenderá la variable salida5
    Salida7:=TRUE;
ELSIF AccionRecibida = 11 THEN //Cuando la acción recibida sea un 11, se apagará la variable salida5
    Salida7:=FALSE;
END_IF
```

*Código 38. Programa Automation Studio. Vinculación de la salida 7 a los estados de inicio y cese de movimiento.*

Para finalizar las vinculaciones a las salidas, el estado de error se representará mediante el encendido de la salida número 8. En cuanto el autómata abandone este estado la salida volverá a su estado inicial de apagado. Se ha necesitado de nuevo la variable *Paso* que, como se ha visto antes, será igual a 100 mientras se encuentre en el estado de error.

```
IF (Paso = 100) THEN // Si se encuentra en el estado de error, se encenderá la salida 8
    Salida8 := TRUE;
ELSIF NOT (Paso = 100) THEN // Si no existiera ningún error, la salida 8 estaría apagada
    Salida8 := FALSE;
END_IF
```

*Código 39. Programa Automation Studio. Vinculación de la salida 8 al estado de Error.*

Por último, se utiliza la variable *VelocidadReal* y para convertir la velocidad leída en tiempo real. Estos valores serán los que se envíen por TCP Server al ESP32 y se muestren en el servidor web en forma de gráfica de datos en tiempo real.

```
// Damos el valor convertido de la velocidad y par actuales para mandar al cliente
VelocidadReal := TRUNC(ActualVelocity*CONST_VEL);
```

*Código 40. Programa Automation Studio. Conversión y almacenamiento de la velocidad y el par actual.*

# CAPÍTULO 5: PROGRAMA ARDUINO IDE: DISEÑO DE LA INTERFAZ HTML

### *5.1. Microcontrolador ESP32*

El ESP32 pertenece a una familia de microcontroladores de la empresa Espressif Systems y se caracteriza por ofrecer además de conectividad Wifi, Bluetooth. De esta manera, el ESP32 ofrece una amplia cantidad de aplicaciones debido a que es capaz de conectarse tanto a Internet como directamente a cualquier dispositivo. (fácil, 2021)

El System On a Chip ESP32 ha superado a su antecesor, el ESP8266, mejorando su conectividad y procesamiento. En concreto, en este proyecto se ha trabajado con el módulo ESP32-WROOM-32, el cual cuenta con una arquitectura de 32 bits.

La plataforma ESP32 permite desarrollar sus aplicaciones mediante numerosas librerías, lenguajes de programación y recursos. En este trabajo es la encargada de diseñar la interfaz HTML que facilitará al usuario estudiar los servomotores. Se ha decidido realizar el programa con Arduino IDE que es uno de los entornos más comunes utilizados con el microcontrolador. (Mechatronics, 2018)

### *5.2. Introducción al programa Arduino IDE*

El entorno de desarrollo integrado o IDE de Arduino es una aplicación que utiliza un lenguaje de programación propio basado en el C++ y que permite cargar y escribir programas en placas Arduino y compatibles, como ESP32 o ESP8266. Es un programa altamente utilizado debido a su facilidad, rapidez y versatilidad, ya que pese a estar basado en el lenguaje C++, es capaz de soportar otros como C, Wiring o Processing. (Softzone, 2023) Además de ser distribuido con una licencia libre y de manera gratuita, tiene la ventaja de ser un software multiplataforma, de manera que es posible utilizarla en distintos sistemas operativos. También cabe destacar el reducido tiempo en el que es capaz de ejecutar el programa en la placa requerida para el proyecto y la cantidad de información y ayudas que se pueden encontrar en librerías y proyectos en Internet. (Peña, 2020)

Aunque surgió como superconjunto de C, el lenguaje de programación C++ es uno de los lenguajes predominantes en el desarrollo de software comercial en la actualidad, debido principalmente a sus amplias aplicaciones. Se caracteriza por su rapidez y dinamismo, así como su compatibilidad con numerosas bibliotecas. Además, la programación con el lenguaje C++ está orientada a objetos y a configuración de sus parámetros y/o propiedades. (OpenWebinars, 2019)

### *5.3. Preparación*

#### *5.3.1.Instalación del programa*

A la hora de comenzar a programar con Arduino, el primer paso claro se trata de descargarse la aplicación en el ordenador. Entrando en la página web oficial de Arduino en el apartado de Software, se encontrará la versión más actualizada del programa. Para este trabajo se ha utilizado la versión Arduino IDE 2.0.4. debido a ser la más nueva a la hora del inicio del proyecto. Además, es posible descargarse el programa para cualquier dispositivo mediante el cual se quiera realizar la programación.

Como se ha explicado a lo largo de este documento, el código programado en Arduino será compilado y ejecutado a partir de un microcontrolador ESP32. Para que esto funcione, habrá que instalar la librería que permite mostrar la tarjeta del ESP32 en Arduino. A continuación, se explicará el procedimiento para instalarla, que se ha seguido desde la página web RANDOM NERD TUTORIALS. (TUTORIALS, 2021)

En primer lugar, al abrir la aplicación, se irá a la página de Preferencias que se encuentra en la pestaña de Arduino IDE.

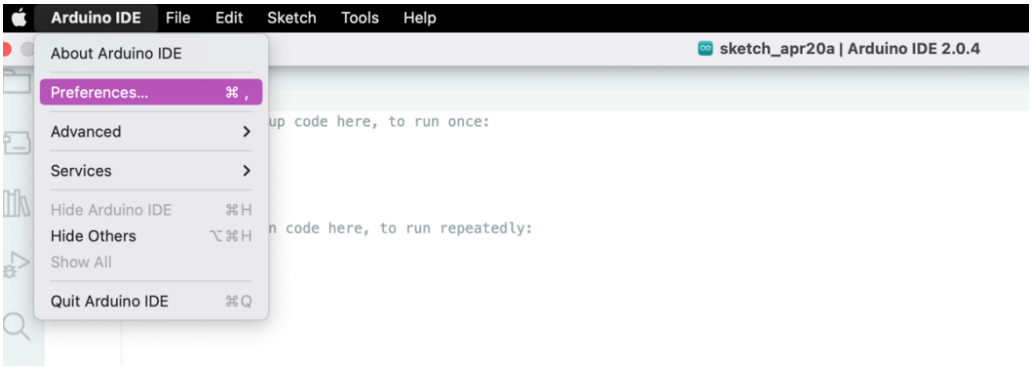



*Figura 17. Programa Arduino IDE. Apertura de la pestaña de preferencias.*

Una vez abierta la pestaña, deberemos añadir un gestor URL de tarjetas que contenga el ESP32 en la línea *Additional boards manager URLs*. Para ello, copiaremos el siguiente enlace: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

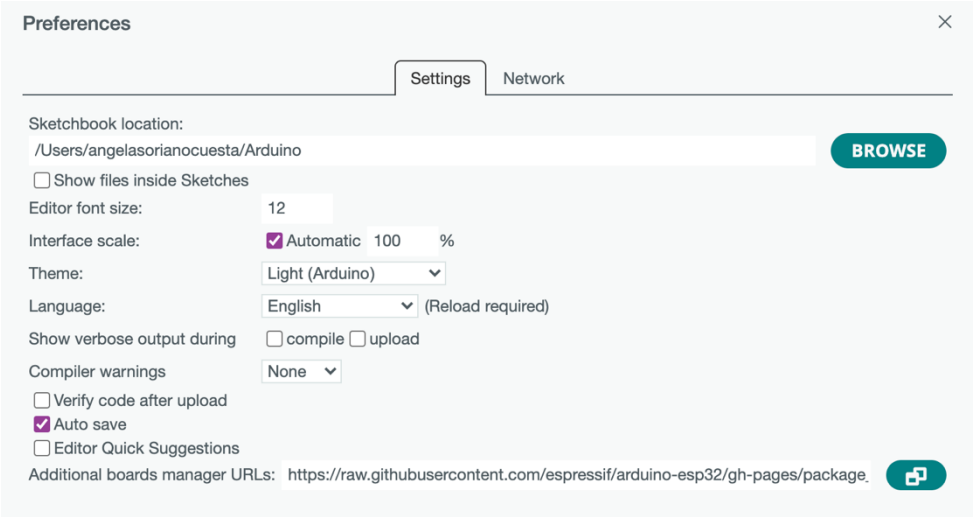



*Figura 18. Programa Arduino IDE. Pestaña de preferencias*

A continuación, se deberá abrir el gestor de tarjetas desde Herramientas, Tarjetas, Gestor de Tarjetas o directamente desde el icono en la parte izquierda de la aplicación. Habrá que buscar esp32 de Espressif Systems e instalarlo.

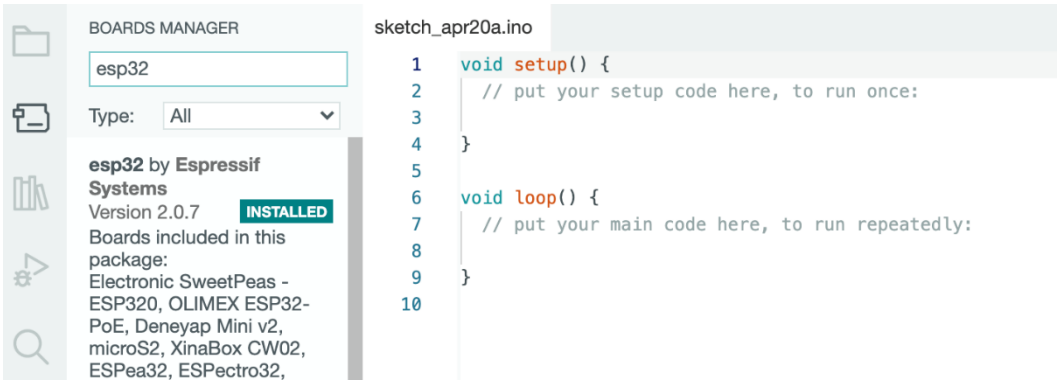
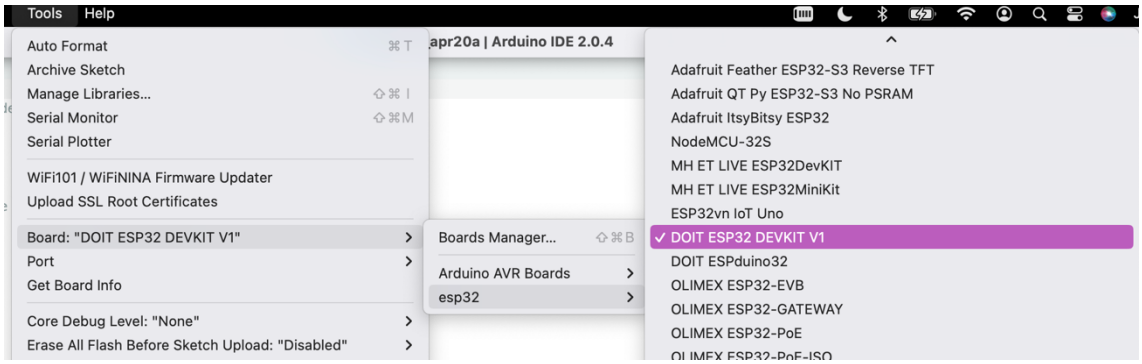



*Figura 19. Programa Arduino IDE. Instalación del gestor de tarjetas del ESP32.*

A la hora de subir un sketch a la placa del ESP32, primero habrá que seleccionar una tarjeta conocida. Entrando en Herramientas, Tarjeta, ahora aparece esp32 y una lista extensa de posibles tarjetas. En este caso, se ha escogido DOIT ESP32 DEVKIT V1 que corresponde al ESP32 utilizado en el proyecto.



Del mismo modo será necesario seleccionar el puerto mediante el cual se conecta el ESP32 al dispositivo que está siendo utilizado.

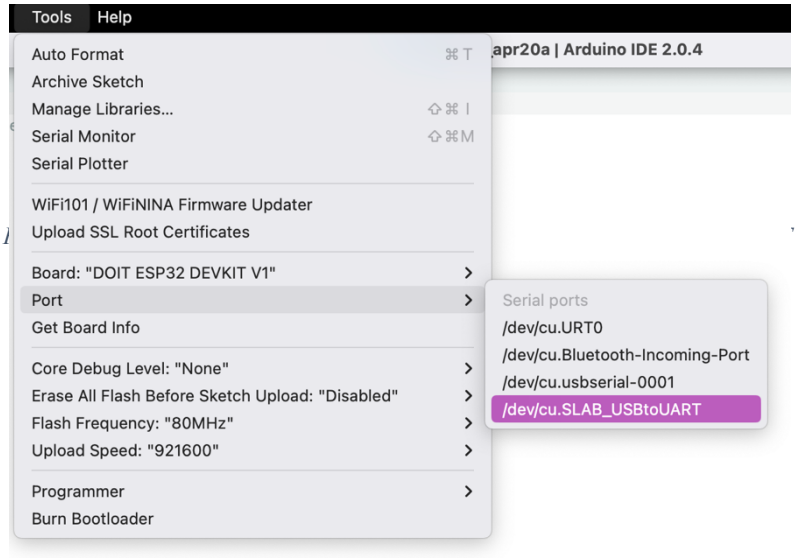



*Figura 21. Programa Arduino IDE. Selección del puerto.*

Finalmente, ya será posible subir el programa a la placa mediante el botón *Upload*:


*Figura 22. Programa Arduino IDE. Subida del sketch a la placa del ESP32.*

Si se ha realizado correctamente se mostrará por la salida el siguiente mensaje:

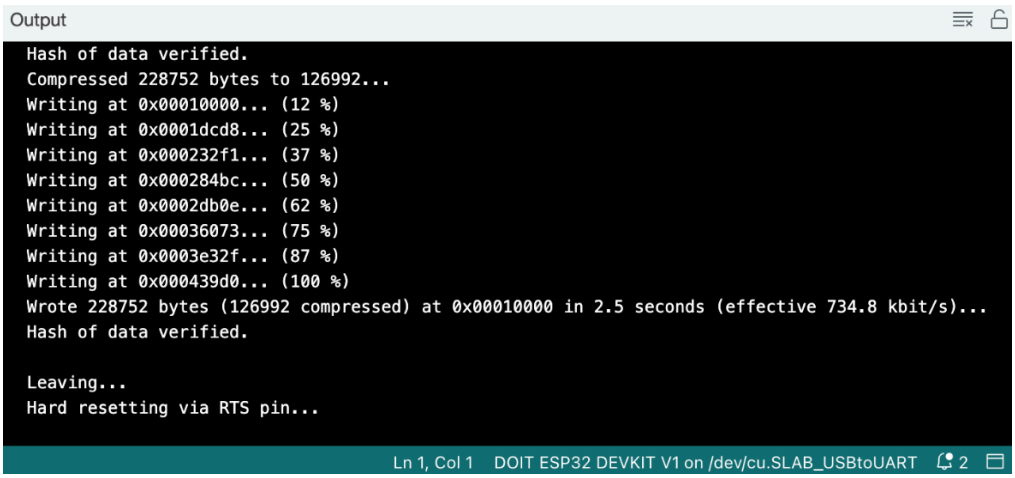



*Figura 23. Programa Arduino IDE. Pestaña de salida.*

### *5.3.2.Instalación de librerías*

Para este proyecto en concreto, se ha necesitado la librería *ESPAsyncWebServer* que es la encargada de generar el servidor web asíncrono. Como no es posible añadirla mediante el gestor de librerías de Arduino, es necesario descargársela y añadirla a la carpeta de librerías del programa. Su descarga está en disponible en el siguiente enlace: https://github.com/me-no-dev/ESPAsyncWebServer#the-async-web-server. Simplemente apretando en el botón "Code" permite descargarse el archivo tipo zip. Este archivo llevará el nombre de *ESPAsyncWebServer-master* y será necesario añadirlo en la carpeta de librerías de Arduino en el dispositivo utilizado durante el proyecto. Finalmente, se deberá renombrar la carpeta descargada a simplemente *ESPAsyncWebServer* para que funcione correctamente.

Para que esta librería funcione con el ESP32, se requiere descargar otra llamada *AsyncTCP,* disponible en: https://github.com/me-no-dev/AsyncTCP. El procedimiento es exactamente el mismo que con la librería anterior.

## *5.4.Funcionamiento básico del programa*

En cuanto a la programación, el funcionamiento básico del Arduino IDE se divide en dos funciones que ejecuta de manera obligatoria, el "*setup*" y el "*loop*". Dentro de la función *setup()* se inicializarán las variables y se ejecutarán los códigos que se desee realizar tan solo una vez. Mientras que en la función *loop()*, se ejecutarán las partes del código un número infinito de veces, hasta que se desconecte o se apague el microcontrolador.

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

*Código 41. Programa Arduino IDE. Base de cualquier programa con Arduino IDE.*

## *5.5.Introducción al código del proyecto*

El objetivo principal del código que se ejecutará en Arduino IDE y que se cargará en el ESP32 es la creación del servidor web que actuará de interfaz del usuario y el intercambio de información entre éste y el programa Automation Studio, encargado de controlar al autómata.

Para la codificación del programa se ha utilizado como referencia diversos programas base de la página web RANDOM NERD TUTORIALS, a partir de los cuales se ha añadido y modificado partes del código de manera que se acoplaran a este proyecto concreto. (TUTORIALS, 2021) (TUTORIALS, 2019)

En primer lugar, dentro de la parte de creación de la aplicación web, se puede distinguir diferentes pestañas que se mostrarán al usuario. La primera, se trata de una página que sirve como menú, en la que se encuentran los botones que conectan o desconectan al ESP32 con el autómata, además de los que llevan al resto de pestañas del proyecto. También se mostrará en la interfaz el estado de la conexión en cada instante.


*Figura 24. Interfaz de usuario. Pestaña del menú. Pantalla que se le muestra al cliente cuando este entra inicialmente al servidor web.*

Dos de las pestañas serán Motor 1 y Motor 2, en cada una de estas se podrá encontrar el estado del servomotor correspondiente. En el caso del primer motor, se encuentran una serie de botones que serán capaces de encender, poner en movimiento y parar el servomotor que se encarga de generar un par motor. Además, se mostrará por pantalla el estado de este motor en todo momento (ON/OFF) y el usuario será capaz de mandar un dato de velocidad en revoluciones por minuto, de manera que el servomotor se mueva a ese valor.


*Figura 25. Interfaz de usuario. Pestaña del estado del servomotor 1.*

En el caso del segundo motor sería igual, con la diferencia de poder mandar un valor de par en unidades del sistema internacional. Así, el servomotor estaría en movimiento cualquiera que sea la velocidad intentando establecer un par resistente igual al establecido por el operador.

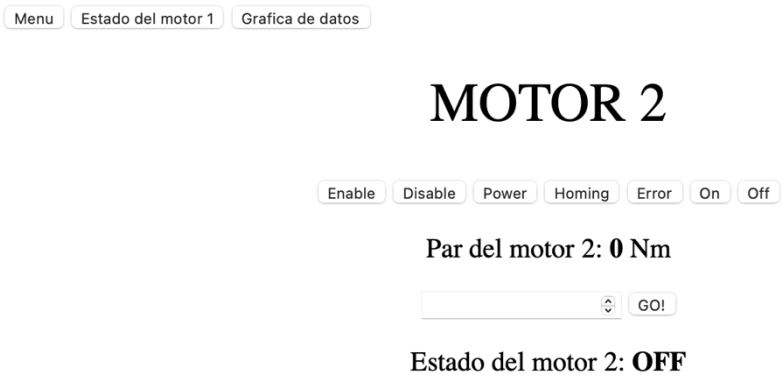



*Figura 26. Interfaz de usuario. Pestaña del estado del servomotor 2.*

Por último, la otra pestaña se encarga de mostrar por la interfaz de usuario una gráfica que representa los valores en tiempo real de la velocidad del motor. Cada cinco segundos se leerá el valor actual de los servomotores y se representará gráficamente.


*Figura 27. Interfaz de usuario. Pestaña que presenta la gráfica de velocidad.*

Como se puede observar, en todas las pestañas se encontrarán tres botones en la parte superior izquierda, que llevan a cualquiera de las pantallas distintas. De esta manera no hace falta volver al menú si se quiere cambiar de pestaña.

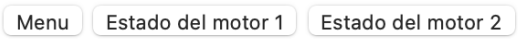



*Figura 28. Interfaz de usuario. Botones que permiten volver a cada una de las pestañas.*

### *5.6.Explicación del código*

#### *5.6.1.Conexión a wifi*

El primer paso que hay que seguir a la hora de realizar la programación del servidor web con el ESP32, es realizar la conexión a Wifi de la placa. Para ello, se necesita de una librería llamada *WiFi* que contiene las funciones necesarias para conectarse a la red correctamente. En este trabajo se ha empleado la conexión a Wifi del rúter que se encuentra en el laboratorio asociado al banco de ensayos.

```
#include <WiFi.h>

#define WIFI_SSID " " // nombre de la wifi
#define WIFI_PASSWORD " " // contraseña
```

*Código 42. Programa Arduino IDE. Definición de la librería y los parámetros de la red.*

Es importante, una vez se entra en el *setup*, inicializar la comunicación serial para que el programa sea capaz de mostrar por pantalla del ordenador lo que le mande el usuario. Esto no es necesario para la funcionalidad normal de la interfaz de usuario vía página web pero es un medio muy cómodo de depurar las respuestas a los diferentes eventos.

```
void setup(void)
{

  Serial.begin(115200);
```

*Código 43. Programa Arduino IDE. Inicialización de la comunicación serial.*

A continuación, mediante la función *WiFi.begin* se añade una nueva red cuyo nombre y contraseña han sido definidos como constantes previamente. Existen múltiples librerías con funciones que son capaces de realizar la conexión a la red Wifi. Por ejemplo, la librería *WifiMulti* es interesante debido a que, al contrario que otras, es capaz de conectarse a más de una red, de manera que si una falla, se conectará automáticamente a la otra. Sin embargo, para este proyecto no ha sido necesario la utilización de más de una red, por tanto, se ha escogido la librería más simple que cumple la función que buscamos.
Una vez llamada la función que inicia la conexión a Wifi, hasta que esta se realice correctamente, se irá escribiendo un mensaje cada segundo en el monitor serial que indica que la conexión está en proceso. Se conocerá que el microcontrolador se ha conectado a la red correctamente en el momento en que se pueda leer por pantalla “Wifi conectado” y la dirección IP que identifica al dispositivo de Wifi al cual se ha conectado.

```
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Conectando a WiFi..");
}
Serial.println("Wifi conectado");
Serial.println(WiFi.localIP());
```

*Código 44. Programa Arduino IDE. Conexión a Wifi.*

La dirección IP será la dirección web a la que el usuario se conectará y donde se mostrará la página web que habremos diseñado.

### *5.6.2.Diseño de la página web*

Como se ha comentado previamente, a la hora de diseñar la página web se pueden diferenciar una serie de pestañas que se presentan al iniciarse la interfaz de usuario. Cada una de estas se han diseñado como documentos HTML diferentes para estructurar de una manera más sencilla el trabajo. El proceso de generar este documento es sencillo: se definirá al principio del código como dato *char*, y todo lo que se quiera añadir a la página se escribirá entre una línea <!DOCTYPE html> y otra </html>, de la siguiente manera.

```
char paginaweb_menu[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
  //código
</html>
```

*Código 45. Programa Arduino IDE. Definición del documento HTML que contendrá la página web diseñada*

A partir de aquí, simplemente habrá que añadir los elementos que se desea mostrar al usuario. En primer lugar, se puede poner distintos títulos de tamaños diferenciados, además de cualquier tipo de texto:

```
<h1>ESP32 Servidor Web</h1>     <h2> MOTOR 1 </h2>
```

*Código 46. Programa Arduino IDE. Ejemplo del diseño de títulos en el documento HTML.*

Tan solo habrá que darles el estilo y tamaño que se quiera al inicio de la definición del documento HTML. En este trabajo se ha elegido la fuente "*Apolline*" y se le ha dado un tamaño al título principal de 50 píxeles, y al resto un poco más pequeños, de 40. En esta parte del código es donde se añadiría todo lo relacionado a la disposición de la página web, ya sea márgenes, orientación, ajuste y tamaño del texto, etc.

```
<style>
    h1 {
      font-family: apolline, serif;
      font-size: 50px;
      font-style: normal;
      font-weight:400;
      text-align: center;
    }
    h2 {
      font-family: apolline, serif;
      font-size: 40px;
      font-style: normal;
      font-weight:400;
      text-align: center;
      margin-top: 30px;
    }
</style>
```

*Código 47. Programa Arduino IDE. Ejemplo del diseño del estilo del servidor web.*

Otra parte importante de la parte del diseño del servidor web es la creación de los botones. A cada uno de estos se les asignará un nombre que permita al usuario reconocer la acción que genera. El código de todos los botones implementados en el proyecto se basa en lo siguiente:

```
<button onclick="window.location = 'http://'+location.hostname+'/connect'">Connect</button>
<button onclick="window.location = 'http://'+location.hostname+'/disconnect'">Disconnect</button>
```

*Código 48. Programa Arduino IDE. Diseño de botones del servidor web.*

### *5.6.3.Llamada al servidor web*

Una vez se tiene el diseño de la página web, hay que programar cómo funciona el servidor web cuando un cliente accede a éste. Para ello, se ha utilizado la librería *ESPAsyncWebServer,* la cual se encarga de mandar el documento HTML cuando el cliente entra en la dirección que se ha nombrado.

```
#include <ESPAsyncWebServer.h>
```

*Código 49. Programa Arduino IDE. Librería encargada de crear el servidor web.*

Es necesario, primero que nada, inicializar el servidor web en el puerto TCP 80.

```
AsyncWebServer server(80);
```

*Código 50. Programa Arduino IDE. Función que inicializa el servidor web.*

Por ejemplo, cuando el usuario entre en la dirección IP de la red Wifi utilizada en el proyecto, le llegará la solicitud al ESP32 y el siguiente código le permitirá mostrar la página web diseñada que funciona como menú. Esta se ha generado como documento HTML como se ha explicado antes.

```
server.on("/", [](AsyncWebServerRequest * request) {
request->send_P(200, "text/html", paginaweb_menu);
});
```

*Código 51. Programa Arduino IDE. Llamada al servidor web.*

Así, dentro de la función *setup*, habrá que relacionar cada pedido que sea capaz de hacer el cliente con el documento HTML que se corresponda.

### *5.6.4.Menú*

En la pestaña del menú, además de todo lo que se ha explicado como ejemplo en los apartados anteriores, también habría que diseñar el código que permite mostrar el estado de la conexión. En el documento HTML que se acaba de explicar tan solo habría que añadir una línea de texto tal que:

```
<p><b>%connection%</b></p>
```

*Código 52. Programa Arduino IDE. Línea de texto en el documento HTML que mostrará el estado de la conexión con el cliente en todo momento.*

También se necesitará definir una cadena de caracteres que hemos llamado *processor* la cual recibe una constante *String* y una variable *var*. Si *var* es igual a *connection*, la palabra que se ha decidido poner entre porcentajes en el documento HTML, dependiendo de cómo esté la conexión mostrará por pantalla una cosa u otra. Se crea una variable entera *conexión*, la cual tendrá un valor de cero si el microcontrolador está desconectado, uno si está en proceso de conectarse y dos si ya se ha conectado con éxito al autómata. Además, se define una cadena de caracteres *estadoConexion* que en un principio se inicializa igual a “DESCONECTADO”. Esta *String* cambiará de valor dependiendo del valor de *conexión* y así lo mostrará en la interfaz.

```
String processor(const String& var){

 if(var=="connection"){
   if(conexion==2){
     estadoConexion = "CONECTADO";
     printf("Connected\n");
     }
   if(conexion==1){
     estadoConexion = "CONECTANDO...";
     printf("Connecting...\n");
     }
   if(conexion==0){
     estadoConexion = "DESCONECTADO";
     printf("Disconnected\n");
     }
   return estadoConexion;
   }
```

*Código 53. Programa Arduino IDE. Cadena de caracteres que permite mostrar los diferentes estados de la conexión.*

Será necesario añadir la cadena *processor* una vez se envia a la interfaz el documento HTML de la página web de la siguiente manera:

```
server.on("/connect", HTTP_GET, [](AsyncWebServerRequest * request)
{
  conexion=1;
request->send_P(200, "text/html", paginaweb_menu, processor);
});

server.on("/disconnect", HTTP_GET, [](AsyncWebServerRequest * request)
{
  conexion=0;
request->send_P(200, "text/html", paginaweb_menu, processor);
});
```

*Código 54. Programa Arduino IDE. Llamada al servidor web que muestra los estados de conexión.*

Se puede observar en este código anterior también como se modifica el valor de la variable conexión. Se inicializa con un valor en cero, desconectado, y se pone a 1 cuando el usuario aprieta el botón para conectarse y manda la dirección web correspondiente. Una vez el cliente ha decidido conectarse, se utiliza la función *localClient.connect* con los parámetros de la dirección IP y el puerto del autómata para realizar la conexión correctamente. Solo en el momento que está finalmente conectado, se envía un mensaje por el monitor serial y se modifica el valor de *conexión* a 2 para informar también al usuario.

```
void loop(void)
{
  if(conexion==1|conexion==2){
    if(localClient.connect(ip, port)){
      Serial.println("localClient connected");
      conexion=2;
```

*Código 55. Programa Arduino IDE. Muestra de la conexión con el cliente por el monitor serial.*

### *5.6.5.Estado de los motores*

Al igual que se ha explicado para la pestaña del menú, los botones y estilos de texto se diseñarían de la misma manera. Sin embargo, los botones que modifican el estado de los motores tienen una función adicional a la de generar una dirección de internet nueva que lleve a la siguiente pestaña. Cada uno de los botones deben cambiar el valor a una variable a la que se ha llamado *sal*, este será el dato que se le enviará al autómata para que genere una acción de los servomotores. Por tanto, cuando el usuario entre a cada dirección asociada a los diferentes botones, el valor de esta variable debe cambiar al que se ha decidido que haga cada función. Así, cada botón está relacionado con el dato que se ha decidido en el programa del Automation Studio realice cada acción.

Por ejemplo, en el caso del primer servomotor, se ha explicado anteriormente que, si el valor que recibe es un 1, el autómata habilita la salida de este motor. De esta manera, cuando un usuario apriete el botón de *Enable* del motor 1, la variable *sal* será igual a 1 y más tarde será enviada al autómata por TCP.

```
server.on("/enable1", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=1;
request->send_P(200, "text/html", paginaweb_motor1);
});
```

*Código 56. Programa Arduino IDE. Llamada al servidor web que muestra el estado del motor 1.*

Se debe explicar también el procedimiento para poder mostrar en la interfaz de usuario en todo momento, el estado de los servomotores. Es lo suficiente parecido a la manera que se ha mostrado el estado de conexión en el menú, sin embargo, aquí tan solo se tienen dos estados diferenciados: encendido y apagado. Además, no es necesario crear una nueva variable entera, ya que, se sabe cuándo estará en movimiento o no a través de la variable *sal.* Primero se añade la siguiente línea al código que crea el documento HTML de la página web que muestra el estado del motor 1.

```
<h3> Estado del motor 1: <b>%state1%</b></h3>
```

*Código 57. Programa Arduino IDE. Línea de texto en el documento HTML que muestra el estado del motor 1.*

Después, no hace falta crear una nueva variable *String*, simplemente dentro de *processor* se añade una condición si *var* es *state1*. Si esto se cumple, se mostrará inicialmente que el motor está en OFF hasta que el valor de salida sea 12. Este se corresponde al valor que toma *sal* cuando el usuario manda la dirección correspondiente a iniciar el movimiento del servomotor 1. Lo mismo pasaría cuando mandamos un 11, que representa el cese del movimiento del motor. Sí que se ha tenido que añadir una variable estadoMotor1 que escribirá ON en el primer caso y OFF en el segundo.

```
String processor(const String& var){

  if(var=="state1"){
    if(sal==12){
      estadoMotor1 = "ON";
      printf("Motor 1 On\n");
      }
    if(sal==11){
      estadoMotor1 = "OFF";
      printf("Motor 1 Off\n");
      }
    return estadoMotor1;
    }
```

*Código 58. Programa Arduino IDE. Cadena de caracteres que permite mostrar los diferentes estados del motor 1.*

Otra parte importante de esta parte del proyecto es programar la manera en la que el usuario es capaz de mandarle un valor de velocidad o par al autómata. Dentro del documento HTML del diseño de la interfaz de cada motor, se añadirá el siguiente código:

```
<form action="/" method="POST">
    <h3 for="vel">Velocidad del motor 1: <b>%vel1%</b> rpm </h3>
    <input type="number" name="vel">
    <input type="submit" value="GO!">
</form>
```

*Código 59. Programa Arduino IDE. Línea del documento HTML que permite al usuario introducir un valor de velocidad y mostrarlo por pantalla.*

En primer lugar, habrá una línea de texto que escriba el propio valor de la velocidad, que se codifica de la misma manera que el estado de los motores y de la conexión. Dentro de la variable de cadena de caracteres que hemos llamado *processor*, se añadirán dos condiciones más: si *var* es igual a *vel1* y si es igual a *par2*. En ambos casos, a partir de las variables *String valorVel* y *valorPar*, se escribirá el valor que se ha pedido por parte del usuario.

```
if(var=="par2"){
  valorPar=par;
  return valorPar;
  }
if(var=="vel1"){
  valorVel=vel;
  return valorVel;
  }
```

*Código 60. Programa Arduino IDE. Cadena de caracteres que permite mostrar los diferentes valores de velocidad y par.*

La cuestión es cómo guardar el valor de velocidad y par, una vez lo pide el cliente. Se ha añadido en el código del documento HTML anterior que se escribirá un número que llamaremos "*vel*", en el caso de la velocidad del motor 1 y "*par*", en el caso del par del motor 2. Entonces, dentro del *setup*, se añadirá el siguiente código:

```
int params = request->params();
 for(int i=0;i<params;i++){
   AsyncWebParameter* p = request->getParam(i);
   if(p->isPost()){
     if (p->name() == PARAM_INPUT_1) {
       vel = p->value().c_str();
       Serial.print("Velocidad del motor 1: ");
       Serial.println(vel);
       vel_int = vel.toInt();
       DEBUG(vel_int);
       salida_vel=vel_int/(VEL_MAX/16383);
       salida_vel= 32768 | salida_vel;
     }

     if (p->name() == PARAM_INPUT_2) {
       par = p->value().c_str();
       Serial.print("Par del motor 2: ");
       Serial.println(par);
       par_int = par.toInt();
       DEBUG(par_int);
       salida_par=par_int/(PAR_MAX/16383);
       salida_par= 49152 | salida_par;
     }
   }
 }
```

*Código 61. Programa Arduino IDE. Almacenamiento y conversión del valor dado por el usuario para ser enviado al autómata.*

El funcionamiento de este se basa en esperar a que el usuario le solicite un parámetro, el cual distinguirá si se trata de una velocidad o un par, a través de las constantes PARAM_IMPUT_1 y PARAM_IMPUT_2, que se han definido inicialmente.

```
const char* PARAM_INPUT_1 = "vel";
const char* PARAM_INPUT_2 = "par";
```

*Código 62. Programa Arduino IDE. Definición de constantes de los parámetros de velocidad y par.*

Ese valor se guarda como cadena de caracteres y se muestra tanto por pantalla en la interfaz de usuario, como por el monitor serial. Además, se ha inicializado una variable entera *vel_int* la cual contendrá ese valor de velocidad entero. Como se ha explicado al inicio del documento, este es un valor analógico, el cual, si se quiere enviar al autómata, habrá que pasarlo a digital. Para ello, se ha seguido los pasos explicados, de manera que las variables *salida_vel* o *salida_par*, contienen los valores en digital de la velocidad o par dados, respectivamente. También cabe mencionar que se le suma a cada uno el dato que pone a 1 los bits correspondientes para distinguir que se trata de un dato de velocidad o de par.

Además, una vez se tiene el propio valor pasado a digital, cabe recordar que hay que convertir ese dato de manera que cuando le llegue al autómata, este pueda reconocer de qué tipo se trata. Por tanto, en el caso de la velocidad se le ha sumado el número que convierte el bit número 16 a 1, mientras que en el caso del par se le ha sumado el número que convierte los dos bits de mayor importancia a 1. De esta manera, cuando el autómata recibe los valores reconoce su procedencia y pasando los bits correspondientes a cero, obtiene el propio valor que el usuario ha solicitado. Serán los valores *salida_vel* y *salida_par* los que serán enviados por comunicación TCP, más tarde se explicará la manera detalladamente.

### *5.6.6.Gráfica de datos*

Para generar la representación gráfica de los datos de velocidad de los servomotores, dentro del documento HTML además del título que se le quiere dar, se diseñará una división en la que se generará la gráfica.

```
<h2>Grafica de datos</h2>
<div id="chart-velocidad" class="container"> </div>
```

*Código 63. Programa Arduino IDE. Documento HTML de la ventana que muestra la gráfica.*

A continuación, se utilizará una función que es capaz de generar los títulos, los ejes y todos los estilos de la propia representación.

```
<script>
var chartT = new Highcharts.Chart({
  chart:{ renderTo : 'chart-velocidad' },
  title: { text: 'Velocidad del motor 1' },
  series: [{
    showInLegend: false,
    data: []
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#059e8a' }
  },
  xAxis: { type: 'datetime',
    dateTimeLabelFormats: { second: '%H:%M:%S' }
  },
  yAxis: {
    title: { text: 'Velocidad (rpm)' }
  },
  credits: { enabled: false }
});
```

*Código 64. Programa Arduino IDE. Función que diseña la gráfica que representa los valores de velocidad.*

También es necesaria una segunda función que sea la encargada de generar cada cierto tiempo un pedido a la dirección que ha sido generada para la velocidad. En este caso se ha decidido que cada 5 segundos genere una solicitud de los datos.

```
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var x = (new Date()).getTime(),
          y = parseFloat(this.responseText);
      //console.log(this.responseText);
      if(chartT.series[0].data.length > 40) {
        chartT.series[0].addPoint([x, y], true, true, true);
      } else {
        chartT.series[0].addPoint([x, y], true, false, true);
      }
    }
  };
  xhttp.open("GET", "/velocidad", true);
  xhttp.send();
}, 5000 ) ;
</script>
```

*Código 65. Programa Arduino IDE. Función que solicita un dato de velocidad al autómata cada cinco segundos y lo representa en la gráfica*

Es importante, dentro del documento HTML, escribir ambas funciones dentro de un script como se muestra al principio y al final de los dos códigos anteriores, ya que de esta manera permitirá que las dos funciones actúen como deben.

Lo único que cambia a la hora de programar el funcionamiento del código cuando el usuario entra en la dirección de la gráfica, es que se añadirán la dirección de la velocidad, pero lo que se mandará será, en vez de un documento HTML, una *String* que contenga el valor leído del autómata. Esta cadena de caracteres estará guardada en una variable *entrada_velocidad,* que será explicada con más detenimiento en el apartado siguiente.

```
server.on("/grafica", [](AsyncWebServerRequest * request)
{
entrada=1;
request->send_P(200, "text/html", paginaweb_grafica);
});
server.on("/velocidad", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", entrada_velocidad.c_str());
});
```

*Código 66. Programa Arduino IDE. Llamada al servidor web que muestra las gráficas de velocidad y par.*

### *5.6.7.Conexión TCP*

A continuación, se explicará el funcionamiento del código encargado de enviar y recibir datos con el autómata, pudiendo dividirlo en esas dos acciones diferenciadas. Esta parte del código está basada en un ejemplo de la página web Stack Overflow, con ciertas modificaciones aplicadas al trabajo concreto. (Overflow, 2022)

*5.6.7.1.Envío de datos*

Se definirán en un primer lugar las dos constantes que hacen referencia al puerto y a la dirección IP del autómata, parámetros necesarios para el inicio de la comunicación entre el PLC y el ESP32.

```
WiFiClient localClient;

const uint port = 12010;
const char* ip = "192.168.137.50";
```

*Código 67. Programa Arduino IDE. Definición de parámetros para la conexión con el autómata.*

Se creará entonces una función que se conecta al cliente mediante *localClient.connect* y, una vez conectado, envía el dato de salida que este quiere mandarle al autómata. Además, en ese momento se mostrará por el monitor serial que el mensaje ya ha sido enviado con éxito.

```
void sendRequest(int sal) {

  if (localClient.connect(ip, port)) {
    localClient.write(sal);
    localClient.write(sal>>8);
    Serial.println("msg sent");
    }

}
```

*Código 68. Programa Arduino IDE. Función que se encarga de leer los datos que recibe por parte del autómata*

Existe una dificultad en este punto, ya que ese valor de salida puede tratarse de tres tipos de datos diferentes. Puede ser un dato de acción o un valor de velocidad o de par. Además, para que el autómata sea capaz de leer este dato, no se podría mandar más de uno diferente a la vez. El procedimiento dentro de la función *loop* se basa en que, si el usuario entra a una dirección que modifica el valor de algún dato de salida, este se manda a través de la función *sendRequest*. Una vez enviado al autómata, la variable que almacena el valor se igualará a cero. De esta manera, el ESP32 estará preparado para enviar el siguiente valor de salida, sea de acción, velocidad o par.

```
if(sal!=0){
  sendRequest(sal);
  Serial.println(sal);
  sal=0;
}
```

*Código 69. Programa Arduino IDE. Código que permite enviar un dato de acción al autómata.*

El código se repetiría para el caso de salida de dato de velocidad y de par.

*5.6.7.2.Recepción de datos*

Como se ha explicado anteriormente, se ha creado una función que será la encargadas de leer los datos de velocidad cuando el usuario entre en la interfaz que muestra la gráfica. De esta manera, si el ESP32 está conectado al autómata, mostrará por pantalla que ha iniciado el proceso de lectura y, mientras que esté disponible, leerá los valores que le envía. Cada dato se escribirá en el monitor serial y, además, será devuelto como cadena de caracteres para representarse en la gráfica. Este proceso se repetirá cada 5 segundos mientras que el usuario se encuentre en la pestaña de representación gráfica.

```
String readVelocity(){
  while((localClient.connected()) && (entrada!=0)) {
    Serial.println("Start reading");
    while ((localClient.available() > 0) && (entrada!=0)){
      Serial.println("Client available\n");
      entrada_vel = localClient.read();
      Serial.println(entrada_vel);
      Serial.println("\n");
      return String(entrada_vel);
    }
    delay(1000);
  }
return String(entrada_vel);
}
```

*Código 70. Programa Arduino IDE. Función que se encarga de leer los datos recibidos del autómata.*

Existe otra complicación, que es que el autómata no es capaz de leer y escribir a la vez, por lo que no se podrá enviarle datos mientras que éste está mandando los valores de velocidad y par. Se ha decidido crear una variable *entrada* que será igual a cero en los momentos que el usuario le esté mandando datos y acciones al autómata. Se igualará a uno cuando el cliente entre en la página de la gráfica para avisar al código que dejé de mandar los valores de salida. Así, el autómata deja de recibir datos y pasa a mandarlos para ser representados en la interfaz del usuario.
Finalmente, se añadirá en el *loop* la llamada a la función *readVelocity()*, y el valor que retorna se almacenará en la variable *entrada_velocidad,* que es el que se mostrará en la gráfica de datos.

```
if(entrada!=0){
entrada_velocidad=readVelocity();
}
```

*Código 71. Programa Arduino IDE. Llamada a la función encargada de leer el dato de velocidad.*

# CAPÍTULO 6: CONCLUSIÓN

El objetivo principal de este trabajo consistía en diseñar y controlar el banco de ensayos formado por dos servomotores mediante una interfaz HTML basada en el ESP32, lo cual ha sido realizado con éxito.

En primer lugar, se ha diseñado la interfaz de usuario de manera que el cliente sea capaz de ver el estado de los motores, modificar sus valores de velocidad y par y representar gráficamente la velocidad real. Como se ha explicado, el usuario entra en el servidor web y se encuentra con una pestaña que muestra el estado de la conexión con el autómata. A partir de esta puede entrar en las diferentes ventanas que muestran los estados de los motores y representan sus parámetros. De esta manera, se ha conseguido esa interfaz de usuario más sencilla y entendible que permite realizar ensayos de una manera más clara que la implementada en un principio.

Además, el ESP32 ha sido capaz de enviar esas acciones generadas por el usuario al autómata y, este las ha ejecutado correctamente. Es decir, la comunicación por TCP/IP se ha realizado correctamente de manera que el microcontrolador ha relacionado las órdenes del usuario con su dato binario asociado y se lo ha enviado al PLC. Éste a su vez, lo recibe y lo ejecuta tras diferenciar el tipo de dato que se trata y convertirlo en el valor real que el usuario desea.

Por último, el programa del Automation Studio se ha programado debidamente para cumplir sus distintos objetivos: recibir los datos y órdenes del usuario a través del ESP32 y ejecutar las distintas operaciones de los servomotores. Así, una vez recibe una orden del microcontrolador, el PLC es capaz de ejecutar la operación del motor asociada. A su vez, lee el valor actual de la velocidad de los servomotores y se lo envía a través de la comunicación TCP/IP al ESP32, el cual los muestra en la gráfica en la interfaz HTML.

# CAPÍTULO 7: BIBLIOGRAFÍA


ABB. (2020). *About ABB*. Recuperado el 20 de Abril de 2023, de https://global.abb/group/en/about

B&R. (2017). *Sobre nosotros*. Recuperado el 20 de Abril de 2023, de https://www.br-automation.com/es-es/sobre-nosotros/

B&R. (2020). *Servomotores*. Recuperado el 20 de Abril de 2023, de https://www.br-automation.com/es-es/productos/control-de-movimiento/servomotores/

Edición B&R (2017). *Texto Estructurado.*

Edición B&R (2018). *Trabajar con Automation Studio.*

fácil, P. (2021). *ESP32 Wifi y Bluetooth en un solo chip*. Recuperado el 21 de Abril de 2023, de https://programarfacil.com/esp8266/esp32/#Caracteristicas_y_especificaciones_del_ESP32

GENIA. (Octubre de 2006). *Autómatas Programables: Introducción al Estándar IEC-61131*. Recuperado el 19 de Mayo, de http://isa.uniovi.es/docencia/IngdeAutom/transparencias/Pres%20IEC%2061131.pdf

ISO. (Mayo de 2003). *ISO/IEC 15445:2000 Information technology — Document description and processing languages — HyperText Markup Language (HTML)*. Recuperado el 19 de Mayo de 2023, de https://www.iso.org/standard/27688.html

ISO. (2020). *ISO/IEC 14882:2020 Programming languages — C++*. Recuperado el 19 de Mayo, de https://www.iso.org/standard/79358.html

Mechatronics, N. (2018). *Módulo ESP-WROOM-32 ESP32 WIFI*. Recuperado el 21 de Abril de 2023, de https://naylampmechatronics.com/espressif-esp/382-modulo-esp-wroom-32-esp32-wifi.html

OpenWebinars. (2019). *Qué es C++: Características y aplicaciones*. Recuperado el 21 de Abril de 2023, de https://openwebinars.net/blog/que-es-cpp/

Overflow, S. (2022). *ESP32: Send a simple TCP Message and receive the response*. Recuperado el 21 de Marzo de 2023, de https://stackoverflow.com/questions/74551259/esp32-send-a-simple-tcp-message-and-receive-the-response

Peña, C. (2020). *Arduino IDE: Domina la programación y controla la placa.* RedUsers.

Aula 21. (s.f.) *Qué es un servomotor y para qué sirve*. Recuperado el 20 de Abril de 2023, de https://www.cursosaula21.com/que-es-un-servomotor/

Quimel, S.L. (2011). *Servo Brushless y sus componentes básicos*. Recuperado el 20 de Abril de 2023, de http://automatizacion-industrial.es/descargas/SERVO%20BRUSHLESS%20Y%20SUS%20COMPONENTES%20BASICOS.pdf

Softzone. (2023). *¿Vas a programar con Arduino? Todo lo que debes saber*. Recuperado el 21 de Abril de 2023, de https://www.softzone.es/programas/lenguajes/programar-arduino/

TUTORIALS, R. N. (2021). *Installing ESP32 Board in Arduino IDE 2.0 (Windows, Mac OS X, Linux)*. Recuperado el 8 de Marzo de 2023, de https://randomnerdtutorials.com/installing-esp32-arduino-ide-2-0/

TUTORIALS, R. N. (2019). *ESP32/ESP8266 Plot Sensor Reading in Real Time Charts - Web Server*. Recuperado el 5 de Abril de 2023, de https://randomnerdtutorials.com/esp32-esp8266-plot-chart-web-server/

TUTORIALS, R. N. (2021). *ESP32 WebSocket Server: Control Stepper Motor (WebSocket)*. Recuperado el 3 de Abril de 2023, de https://randomnerdtutorials.com/stepper-motor-esp32-websocket/

Unidas, N. (s.f.). *Objetivos de desarrollo sostenible*. Recuperado el 15 de Mayo de 2023, de https://www.un.org/sustainabledevelopment/es/education/

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

# PRESUPUESTO

*Diseño de un banco de ensayos de máquinas de imanes permanentes de técnica senoidal (servoaccionamiento PMSM Brushless AC) de 2,2kW operados mediante autómata programable y con interfaz HTML basada en ESP32*

Ángela Soriano Cuesta

Curso 2022/23

# ÍNDICE DEL PRESUPUESTO

# 1. INTRODUCCIÓN

A continuación se va a realizar una estimación del presupuesto del diseño de un banco de ensayos de máquinas de imanes permanentes con interfaz HTML basada en el microcontrolador ESP32. Se puede dividir el proyecto en tres unidades de obra diferenciadas cuyos elementos se estudiarán detalladamente. Estas son la programación del autómata, la programación del microcontrolador ESP32 y la redacción de la memoria.

Este documento contiene cuatro cuadros de precios: precios de la mano de obra, precios de los materiales, precios unitarios y precios descompuestos. Además se muestra el resultado del presupuesto de ejecución por contrata y base de licitación.

# 2. PRECIOS DE LA MANO DE OBRA

En la siguiente tabla se representan las diferentes personas involucradas en la realización del proyecto, así como el precio por hora que ha supuesto su trabajo.

| CONCEPTO | UNIDAD | DESCRIPCIÓN | PRECIO |
|---|---|---|---|
| INGENIERO JUNIOR | h | Persona que se ha dedicado a realizar el proyecto | 27,5 € |
| SUPERVISOR DEL PROYECTO | h | Persona encargada de supervisar el trabajo | 66 € |

*Tabla 1. Cuadro de precios 1: Precios de la mano de obra.*

Los precios han sido obtenidos aumentando un 10% los precios por hora de hace un par de años de un ingeniero junior y un ingeniero senior, respectivamente.

# 3. PRECIOS DE LOS MATERIALES

En cuanto a los costes de los materiales, en este trabajo los únicos materiales que intervienen son los distintos programas que se han utilizado, los dispositivos en los cuales se ha realizado el proyecto y el microcontrolador ESP32 que se ha obtenido específicamente para éste. Los elementos que forman el banco de ensayo no se cuentan parte de este proyecto, ya que se ha realizado para un banco ya existente.

| CONCEPTO | UNIDAD | DESCRIPCIÓN | AMORTIZACIÓN | PRECIO |
|---|---|---|---|---|
| PROGRAMA AUTOMATION STUDIO | ud | Herramienta de programación de la empresa B&R | - | 0 € |
| PROGRAMA ARDUINO IDE | ud | Plataforma de programación | - | 0 € |
| PROGRAMA MICROSOFT WORD | ud | Programa en el que se ha realizado la redacción de los documentos | - | 0 € |
| PC | meses | Ordenador de mesa | 5 años | 10 € |
| ORDENADOR PROPIO | meses | Ordenador portátil | 5 años | 16,67 € |
| MICROCONTROLADOR ESP32 | ud | Microcontrolador de módulo ESP32-WROOM-32 | - | 6,99 € |
| CABLE USB | ud | Conexión USB entre el ESP32 y el ordenador | - | 3,01 € |

*Tabla 2. Cuadro de precios 2: Precios de los materiales.*

Los programas que se han utilizado para la programación del PLC y del ESP32 y para la redacción de la memoria han sido gratuitos, pero es importante añadirlos como materiales que han intervenido en el proyecto.

En cuanto al PC, se le ha dado un coste de 600 € y una vida útil de 5 años de manera que el precio al mes es de 10 €. De la misma manera, al ordenador propio se le ha dado también 5 años de vida útil a un precio de unos 1000 €, obteniendo un precio al mes de 16,67 €.

# 4. PRECIOS UNITARIOS

A continuación se muestra el cuadro de precios unitarios correspondientes para cada una de las manos de obra.

| CONCEPTO | UNIDAD | DESCRIPCIÓN | PRECIO |
|---|---|---|---|
| UDO1 | ud | Programación del PLC mediante la aplicación Automation Studio de la empresa B&R | 5.091,84 € |
| UDO2 | ud | Programación del microcontrolador ESP32 mediante el entorno Arduino | 3.410,51 € |
| UDO3 | ud | Redacción de la memoria del proyecto | 4.250,14 € |

*Tabla 3. Cuadro de precios unitarios.*

En el siguiente apartado se descompondrán cada una de estas con las cantidades de materiales y mano de obra que intervienen.

# 5. PRECIOS DESCOMPUESTOS

### 5.1. Unidad de obra 1: Programación del PLC

El autómata programable es el elemento del proyecto que se encarga de controlar los servomotores del banco de ensayo, por lo que su programación es de suma importancia en este trabajo. Para ello se ha hecho uso de la herramienta de programación de la empresa B&R llamada Automation Studio y se ha utilizado el PC asociado al banco de ensayos del laboratorio.

| CONCEPTO | UNIDAD | RENDIMIENTO | PRECIO | IMPORTE |
|---|---|---|---|---|
| PROGRAMA AUTOMATION STUDIO | ud | 1 | 0 € | 0 € |
| PC | meses | 1,2 | 10 € | 12 € |
| INGENIERO JUNIOR | h | 120 | 27,5 € | 3.300 € |
| SUPERVISOR DEL PROYECTO | h | 24 | 66 € | 1.584 € |
| | | | Costes directos | 4.896 € |
| | | 4% | Costes indirectos | 195,84 € |
| | | | Coste total: | 5.091,84 € |

*Tabla 4. Cuadro de precios descompuestos. Unidad de obra 1: Programación del PLC.*

### 5.2. Unidad de obra 2: Programación del ESP32

El microcontrolador ESP32 es el encargado de diseñar la interfaz de usuario e intercambiar información con el autómata. Su programación se ha realizado mediante la plataforma Arduino IDE en el ordenador propio.

| CONCEPTO | UNIDAD | RENDIMIENTO | PRECIO | IMPORTE |
|---|---|---|---|---|
| PROGRAMA ARDUINO IDE | ud | 1 | 0 € | 0 € |
| ORDENADOR PROPIO | meses | 0,8 | 16,67 € | 13,34 € |
| MICROCONTROLADOR ESP32 | ud | 1 | 6,99 € | 6,99 € |
| CABLE USB | ud | 1 | 3,01 € | 3,01 € |
| INGENIERO JUNIOR | h | 80 | 27,5 € | 2.200 € |
| SUPERVISOR DEL PROYECTO | h | 16 | 66 € | 1.056 € |
| | | | Costes directos | 3.279,34 € |
| | | 4% | Costes indirectos | 131,17 € |
| | | | Coste total: | 3.410,51 € |

*Tabla 5. Cuadro de precios descompuestos. Unidad de obra 2: Programación del ESP32.*

### 5.3. Unidad de obra 3: Redacción de la memoria

Por último, debe realizarse una unidad de obra que contenga el coste de la redacción de los documentos técnicos en los que se expone el proyecto. Para ello se ha hecho uso del programa Microsoft Word descargado en el ordenador propio.

| CONCEPTO | UNIDAD | RENDIMIENTO | PRECIO | IMPORTE |
|---|---|---|---|---|
| PROGRAMA MICROSOFT WORD | ud | 1 | 0 € | 0 € |
| ORDENADOR PROPIO | meses | 1 | 16,67 € | 16,67 € |
| INGENIERO JUNIOR | h | 100 | 27,5 € | 2.750 € |
| SUPERVISOR DEL PROYECTO | h | 20 | 66 € | 1.320 € |
| | | | Costes directos | 4.086,67 € |
| | | 4% | Costes indirectos | 163,47 € |
| | | | Coste total: | 4.250,14 € |

*Tabla 6. Cuadro de precios descompuestos. Unidad de obra 3: Redacción de la memoria.*

# 6. PRESUPUESTO DE BASE DE LICITACIÓN

Finalmente, obtenemos el presupuesto de ejecución material sumando los precios unitarios de todas las unidades de obra. De esta manera:

| | |
|---|---|
| **Presupuesto Ejecución Material (PEM)** | **12.752,49 €** |
| Gastos Generales (12%) | 1.530,30 € |
| Beneficio Industrial (6%) | 765,15 € |
| **Presupuesto de Ejecución por Contrata (PEC)** | **15.047,94 €** |
| IVA (21%) | 3.160,07 € |
| **Presupuesto base de licitación** | **18.208,01 €** |

*Tabla 7. Cuadro de presupuestos.*

El presupuesto de base de licitación de este proyecto asciende a la cantidad de DIECIOCHO MIL DOSCIENTOS OCHO EUROS CON UN CÉNTIMO.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

# ANEXO 1

*Manual de usuario de la interfaz HTML*

Ángela Soriano Cuesta

Curso 2022/23

En este documento se explicará a fondo el funcionamiento de la interfaz HTML desde el punto de vista del usuario.

Primeramente se debe conocer la dirección IP del rúter de la wifi, ya que será la dirección del servidor web, y el dispositivo con el que se quiere trabajar deberá estar conectado a esta.


*Figura 1. Interfaz de usuario. Dirección IP.*

Una vez se entra en la dirección, se mostrará la pantalla que funciona como menú principal. En ésta se puede observar el estado de conexión entre el PLC y el ESP32, que cuando se accede a la página web inicialmente estará en DESCONECTADO.


*Figura 2. Interfaz de usuario. Estado de desconexión entre PLC y ESP32.*

A partir de los botones de abajo el usuario será capaz de entrar al resto de ventanas que ejecutan las órdenes del proyecto, pero ninguna se enviará hasta que no se apriete el botón de *Connect,* que se encarga de iniciar la comunicación entre autómata y microcontrolador. Se mostrará entonces el mensaje CONECTANDO... que informa al usuario que la conexión se está realizando como en la siguiente figura:


*Figura 3. Interfaz de usuario. Inicio del proceso de conexión entre PLC y ESP32 mediante el botón Connect.*

Una vez se inicie la conexión entre los dos elementos, se mostrará en la pantalla principal el mensaje CONECTADO. A partir de este momento se puede iniciar la puesta en marcha de los motores.


*Figura 4. Interfaz de usuario. Estado de conexión entre PLC y ESP32.*

Se estudiará en primer lugar las operaciones del primer servomotor, el que se encuentra en modo de control de velocidad. Para ello se elegirá el botón de Motor 1 desde el menú, abriendo una ventana que muestra el estado de éste y sus diferentes operaciones.


*Figura 5. Interfaz de usuario. Entrada a la ventana de estado del servomotor 1 mediante el botón Motor 1.*


*Figura 6. Interfaz de usuario. Ventana de estado del servomotor 1.*

Inicialmente se puede observar que el estado del motor está en OFF, ya que no está en movimiento, y la velocidad del motor es de 0 revoluciones por minuto debido a que todavía no se ha añadido un valor de velocidad.

El usuario deberá seguir un orden de acciones para conseguir la puesta en marcha del motor. El primer paso una vez conectada la alimentación de potencia del banco de ensayos, se trata de la habilitación del funcionamiento del servomotor mediante el botón *Enable*. Se podrá comprobar que el PLC ha recibido y ejecutado esta acción mediante el encendido de la tercera salida del autómata.


*Figura 7. Interfaz de usuario. Ventana de estado del servomotor 1. Habilitación de la salida mediante el botón Enable.*

La siguiente orden se trata del encendido del servoamplificador, que se ejecuta al apretar el botón *Power*. En el momento en el que el autómata la reciba y la ejecute, la salida cinco se encenderá, confirmando al usuario su correcta ejecución.


*Figura 8. Interfaz de usuario. Ventana de estado del servomotor 1. Encendido del servomotor mediante el botón Power.*

A continuación, deberá seleccionar el botón *Homing* que realiza la búsqueda del punto de referencia, esta orden se realiza de forma rápida de manera que la salida número seis asociada a esta orden simplemente parpadeará. Una vez, ha parpadeado y, por tanto, ejecutado la acción, se puede iniciar el movimiento del motor.


*Figura 9. Interfaz de usuario. Ventana de estado del servomotor 1. Búsqueda del punto de referencia mediante el botón Homing.*

Para poner en marcha el motor habría que enviarle primero una velocidad a través de la interfaz. Para ello se tiene que escribir el valor en revoluciones por minuto que se desea y apretar el botón *Go*.


*Figura 10. Interfaz de usuario. Ventana de estado del servomotor 1. Envío de un nuevo valor de velocidad.*

En pantalla se mostrará la velocidad que se acaba de enviar al motor, de manera que solo faltaría apretar el botón *On*. En este momento se iniciaría el movimiento del primer motor y se encendería la salida siete, avisando al usuario de que se ha ejecutado. Por pantalla se le mostraría también que el estado del motor está en ON.


*Figura 11. Interfaz de usuario. Ventana de estado del servomotor 1. Puesta en marcha del motor mediante el botón On.*

A la hora de parar el movimiento bastaría con apretar el botón *Off*, apagando a su vez la salida asociada. Además, el estado del motor pasaría a OFF, ya que no está girando en ese momento, como se puede observar en la siguiente figura.


*Figura 12. Interfaz de usuario. Ventana de estado del servomotor 1. Apagado del movimiento del motor mediante el botón Off.*

Si se quisiera volver a iniciar el movimiento mediante la selección del botón *On*, éste funcionaría al último valor de velocidad que se ha pedido. Si se quisiera cambiar la velocidad no sería necesario parar el motor, simplemente se escribiría un nuevo valor y se apretaría de nuevo a *Go*. El motor empezaría a girar a esa nueva velocidad de manera inmediata.

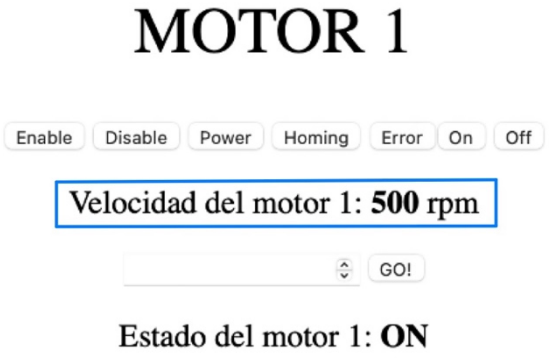



*Figura 13. Interfaz de usuario. Ventana de estado del servomotor 1. Modificación del parámetro de velocidad.*

Una vez se quiera apagar el banco, se deberá parar el movimiento del motor mediante el botón *Off*, como se ha explicado previamente. A continuación, se deberá volver a apretar el botón de *Power*, apagando el funcionamiento del motor y la salida cinco asociada a esta orden.


*Figura 14. Interfaz de usuario. Ventana de estado del servomotor 1. Apagado del servomotor mediante el botón Power.*

Finalmente el usuario deshabilitará el servomotor seleccionando *Disable* en la interfaz, apagando a su vez la tercera salida del autómata.


*Figura 15. Interfaz de usuario. Ventana de estado del servomotor 1. Deshabilitación de la salida mediante el botón Disable.*

Cabe destacar que para acceder a cualquiera del resto de pantallas bastará con apretar el botón correspondiente situado a la esquina superior izquierda de la página.


*Figura 16. Interfaz de usuario. Paso a la ventana del servomotor 2 mediante el botón Estado del motor 2.*

Por otro lado, si se quisiera ensayar el segundo servomotor que trabaja en modo control de par se deberá entrar en la pestaña Motor 2, como se ha mostrado en la figura anterior. El proceso será idéntico al del primer motor.

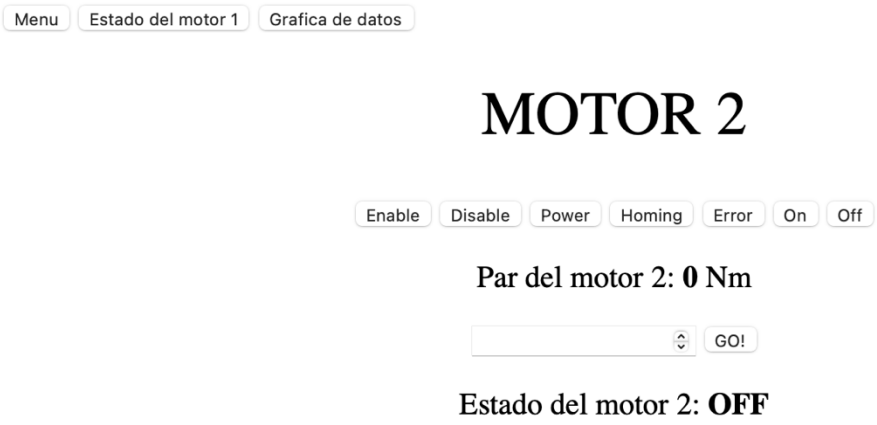



*Figura 17. Interfaz de usuario. Ventana de estado del servomotor 2.*

Finalmente, se accederá a la última pestaña cuando se quiera observar la representación del parámetro de velocidad.

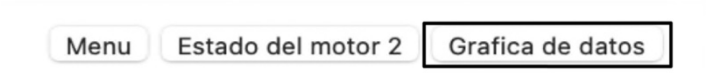



*Figura 18. Interfaz de usuario. Paso a la ventana de la representación de valores mediante el botón Grafica de datos.*

El funcionamiento es sencillo, cada cinco segundos se mostrará el valor actual de velocidad y el usuario será capaz de verlo representado en la gráfica.


*Figura 19. Interfaz de usuario. Ventana de la gráfica de datos.*

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

# ANEXO 2

*Código del programa Arduino IDE*

Ángela Soriano Cuesta

Curso 2022/23

A continuación se muestra el código completo del programa Arduino IDE que se ha utilizado para diseñar la interfaz de usuario y para intercambiar información con el autómata programable. Se puede distinguir la inicialización de las variables y del documento HTML, las funciones de lectura y escritura de los datos intercambiados con el PLC y las funciones de iniciación *setup* y de bucle *loop*.

```
#include <ESPAsyncWebServer.h>
#include <WiFi.h>

#define WIFI_SSID "BandRTestBench" // nombre de la wifi
#define WIFI_PASSWORD "diedosii_servid" // contraseña

#define DEBUG(a)

WiFiClient localClient;

const uint port = 12010;
const char* ip = "192.168.137.50";

const char* PARAM_INPUT_1 = "vel";
const char* PARAM_INPUT_2 = "par";

int sal=0;
int conexion=0;
int vel_int=0;
int salida_vel=0;
int par_int=0;
int salida_par=0;
int entrada=0;
int entrada_vel=0;

const uint VEL_MAX = 30000;
const uint PAR_MAX = 30000;

String estadoMotor1="OFF";
String estadoMotor2="OFF";
String estadoConexion="Desconectado";
String vel="0";
String par="0";
String valorPar="0";
String valorVel="0";
String entrada_velocidad="0";

bool newRequest = false;

char paginaweb_menu[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<head>
```

```
<style>
    body {
      min-width: 310px;
      max-width: 800px;
      height: 400px;
      margin: 0 auto;
    }
    h1 {
      font-family: apolline, serif;
      font-size: 50px;
      font-style: normal;
      font-weight:400;
      text-align: center;
    }
</style>
</head>
<body>
<center>
<h1>ESP32 Servidor Web</h1>
<center>
<button onclick="window.location =
'http://'+location.hostname+'/connect'">Connect</button>
<button onclick="window.location =
'http://'+location.hostname+'/disconnect'">Disconnect</button>
<p><b>%connection%</b></p>
<button onclick="window.location =
'http://'+location.hostname+'/motor1'">Motor 1</button>
<button onclick="window.location =
'http://'+location.hostname+'/motor2'">Motor 2</button>
<button onclick="window.location =
'http://'+location.hostname+'/grafica'">Grafica de datos</button>
</body>
</html>
)=====";

char paginaweb_motor1[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<head>
<style>
    body {
      min-width: 310px;
      max-width: 800px;
      height: 400px;
      margin: 0 auto;
      margin-top: 25px;
    }
    h2 {
      font-family: apolline, serif;
      font-size: 40px;
      font-style: normal;
```

```
      font-weight:400;
      text-align: center;
      margin-top: 30px;
    }
    h3 {
      font-family: apolline, serif;
      font-size: 20px;
      font-style: normal;
      font-weight:400;
      text-align: center;
    }
  </style>
</head>
<body>
<button onclick="window.location =
'http://'+location.hostname+'/'">Menu</button>
<button onclick="window.location =
'http://'+location.hostname+'/motor2'">Estado del motor 2</button>
<button onclick="window.location =
'http://'+location.hostname+'/grafica'">Grafica de datos</button>
<center>
  <h2> MOTOR 1 </h2>
  <button onclick="window.location =
'http://'+location.hostname+'/enable1'">Enable</button>
  <button onclick="window.location =
'http://'+location.hostname+'/disable1'">Disable</button>
  <button onclick="window.location =
'http://'+location.hostname+'/power1'">Power</button>
  <button onclick="window.location =
'http://'+location.hostname+'/homing1'">Homing</button>
  <button onclick="window.location =
'http://'+location.hostname+'/error1'">Error</button\n>
  <button onclick="window.location =
'http://'+location.hostname+'/on1'">On</button>
  <button onclick="window.location =
'http://'+location.hostname+'/off1'">Off</button>
  <form action="/" method="POST">
      <h3 for="vel">Velocidad del motor 1: <b>%vel1%</b> rpm </h3>
      <input type="number" name="vel">
      <input type="submit" value="GO!">
  </form>
<h3> Estado del motor 1: <b>%state1%</b></h3>
</center>
</body>
</html>
)=====";

char paginaweb_motor2[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<head>
```

```
<style>
    body {
      min-width: 310px;
      max-width: 800px;
      height: 400px;
      margin: 0 auto;
      margin-top: 25px;
    }
    h2 {
      font-family: apolline, serif;
      font-size: 40px;
      font-style: normal;
      font-weight:400;
      text-align: center;
      margin-top: 30px;
    }
    h3 {
      font-family: apolline, serif;
      font-size: 20px;
      font-style: normal;
      font-weight:400;
      text-align: center;
    }
  </style>
</head>
<body>
<button onclick="window.location =
'http://'+location.hostname+'/'">Menu</button>
<button onclick="window.location =
'http://'+location.hostname+'/motor1'">Estado del motor 1</button>
<button onclick="window.location =
'http://'+location.hostname+'/grafica'">Grafica de datos</button>
<center>
<h2> MOTOR 2 </h2>
<button onclick="window.location =
'http://'+location.hostname+'/enable2'">Enable</button>
<button onclick="window.location =
'http://'+location.hostname+'/disable2'">Disable</button>
<button onclick="window.location =
'http://'+location.hostname+'/power2'">Power</button>
<button onclick="window.location =
'http://'+location.hostname+'/homing2'">Homing</button>
<button onclick="window.location =
'http://'+location.hostname+'/error2'">Error</button>
<button onclick="window.location =
'http://'+location.hostname+'/on2'">On</button>
<button onclick="window.location =
'http://'+location.hostname+'/off2'">Off</button>

<form action="/" method="POST">
<h3 for="par">Par del motor 2: <b>%par2%</b> Nm</h3>
```

```
      <input type="number" name="par">
      <input type="submit" value="GO!">
</form>

  <h3> Estado del motor 2: <b>%state2%</b></h3>
</center>
</body>
</html>
)=====";

char paginaweb_grafica[] PROGMEM = R"=====(
<!DOCTYPE HTML><html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <style>
    body {
      min-width: 310px;
      max-width: 800px;
      height: 400px;
      margin: 0 auto;
      margin-top: 25px;
    }
    h2 {
      font-family: apolline, serif;
      font-size: 40px;
      font-style: normal;
      font-weight:400;
      text-align: center;
      margin-top: 30px;
    }
  </style>
</head>
<body>
<button onclick="window.location =
'http://'+location.hostname+'/'">Menu</button>
<button onclick="window.location =
'http://'+location.hostname+'/motor1'">Estado del motor 1</button>
<button onclick="window.location =
'http://'+location.hostname+'/motor2'">Estado del motor 2</button>
 <h2>Grafica de datos</h2>
 <div id="chart-velocidad" class="container"> </div>
</body>
<script>
var chartT = new Highcharts.Chart({
  chart:{ renderTo : 'chart-velocidad' },
  title: { text: 'Velocidad del motor' },
  series: [{
    showInLegend: false,
    data: []
  }],
```

```
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#059e8a' }
  },
  xAxis: { type: 'datetime',
    dateTimeLabelFormats: { second: '%H:%M:%S' }
  },
  yAxis: {
    title: { text: 'Velocidad (rpm)' }
  },
  credits: { enabled: false }
});
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var x = (new Date()).getTime(),
          y = parseFloat(this.responseText);
      //console.log(this.responseText);
      if(chartT.series[0].data.length > 40) {
        chartT.series[0].addPoint([x, y], true, true, true);
      } else {
        chartT.series[0].addPoint([x, y], true, false, true);
      }
    }
  };
  xhttp.open("GET", "/velocidad", true);
  xhttp.send();
}, 5000 );
</script>
</html>
)=====";

AsyncWebServer server(80);

String processor(const String& var){

  if(var=="state1"){
    if(sal==12){
      estadoMotor1 = "ON";
      printf("Motor 1 On\n");
      }
    if(sal==11){
      estadoMotor1 = "OFF";
      printf("Motor 1 Off\n");
      }
    return estadoMotor1;
    }
  if(var=="state2"){
```

```
    if(sal==13){
      estadoMotor2 = "ON";
      printf("Motor 2 On\n");
      }
    if(sal==14){
      estadoMotor2 = "OFF";
      printf("Motor 2 Off\n");
      }
    return estadoMotor2;
    }
  if(var=="connection"){
    if(conexion==2){
      estadoConexion = "CONECTADO";
      printf("Connected\n");
      }
    if(conexion==1){
      estadoConexion = "CONECTANDO...";
      printf("Connecting...\n");
      }
    if(conexion==0){
      estadoConexion = "DESCONECTADO";
      printf("Disconnected\n");
      }
    return estadoConexion;
    }
  if(var=="par2"){
    valorPar=par;
    return valorPar;
    }
  if(var=="vel1"){
    valorVel=vel;
    return valorVel;
    }
  return String();
  }


void notFound(AsyncWebServerRequest *request)
{
  request->send(404, "text/plain", "Página no encontrada");
}

void sendRequest(int sal) {

    localClient.write(sal);
    localClient.write(sal>>8);
    Serial.println("Mensaje enviado");
}

String readVelocity(){
  while((localClient.connected()) && (entrada!=0)) {
```

```
    Serial.println("Start reading");
    while ((localClient.available() > 0) && (entrada!=0)){
      Serial.println("Client available\n");
      entrada_vel = localClient.read();
      Serial.println(entrada_vel);
      Serial.println("\n");
      return String(entrada_vel);
    }
    delay(1000);
  }
return String(entrada_vel);
}

void setup(void)
{

  Serial.begin(115200);

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a WiFi..");
  }
  Serial.println("Wifi conectado");
  Serial.println(WiFi.localIP());

  server.on("/", [](AsyncWebServerRequest * request)
  {
   int params = request->params();
    for(int i=0;i<params;i++){
      AsyncWebParameter* p = request->getParam(i);
      if(p->isPost()){
        if (p->name() == PARAM_INPUT_1) {
          vel = p->value().c_str();
          Serial.print("Velocidad del motor 1: ");
          Serial.println(vel);
          vel_int = vel.toInt();
          DEBUG(vel_int);
          salida_vel=vel_int/(VEL_MAX/16383);
          salida_vel= 32768 | salida_vel;
        }
        if (p->name() == PARAM_INPUT_2) {
          par = p->value().c_str();
          Serial.print("Par del motor 2: ");
          Serial.println(par);
          par_int = par.toInt();
          DEBUG(par_int);
          salida_par=par_int/(PAR_MAX/16383);
          salida_par= 49152 | salida_par;
        }
      }
```

```
  }
request->send_P(200, "text/html", paginaweb_menu, processor);
newRequest = true;
});
server.on("/connect", HTTP_GET, [](AsyncWebServerRequest * request)
{
  conexion=1;
request->send_P(200, "text/html", paginaweb_menu, processor);
});
server.on("/disconnect", HTTP_GET, [](AsyncWebServerRequest * request)
{
  conexion=0;
request->send_P(200, "text/html", paginaweb_menu, processor);
});
server.on("/motor1", HTTP_GET, [](AsyncWebServerRequest * request)
{
request->send_P(200, "text/html", paginaweb_motor1, processor);
});
server.on("/motor2", HTTP_GET, [](AsyncWebServerRequest * request)
{
request->send_P(200, "text/html", paginaweb_motor2, processor);
});
server.on("/enable1", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=1;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor1, processor);
});
server.on("/enable2", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=2;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor2, processor);
});
server.on("/disable1", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=3;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor1, processor);
});
server.on("/disable2", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=4;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor2, processor);
});
server.on("/power1", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=5;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor1, processor);
```

```
});
server.on("/power2", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=6;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor2, processor);
});
server.on("/homing1", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=7;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor1, processor);
});
server.on("/homing2", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=8;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor2, processor);
});
server.on("/error1", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=9;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor1, processor);
});
server.on("/error2", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=10;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor2, processor);
});
server.on("/on1", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=12;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor1, processor);
});
server.on("/off1", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=11;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor1, processor);
});
server.on("/on2", HTTP_GET, [](AsyncWebServerRequest * request)
{
  sal=13;
  entrada=0;
request->send_P(200, "text/html", paginaweb_motor1, processor);
});
server.on("/off2", HTTP_GET, [](AsyncWebServerRequest * request)
{
```

```
    sal=14;
    entrada=0;
  request->send_P(200, "text/html", paginaweb_motor1, processor);
  });
  server.on("/grafica", [](AsyncWebServerRequest * request)
  {
  entrada=1;
  request->send_P(200, "text/html", paginaweb_grafica);
  });
  server.on("/velocidad", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", entrada_velocidad.c_str());
  });

  server.onNotFound(notFound);

  server.begin();  // Finalmente inciamos el servidor
}

void loop(void)
{
  if(conexion==1|conexion==2){
    if(localClient.connect(ip, port)){
      Serial.println("localClient connected");
      conexion=2;
      if(sal!=0){
        sendRequest(sal);
        Serial.println(sal);
        sal=0;
      }
      if(salida_par!=0){
        sendRequest(salida_par);
        Serial.println(salida_par);
        salida_par=0;

      }
      if(salida_vel!=0){
        sendRequest(salida_vel);
        Serial.println(salida_vel);
        salida_vel=0;
      }
      if(entrada!=0){
      entrada_velocidad=readVelocity();
      }
    }
  }

  delay(1000);


}
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

# ANEXO 3

*Código del programa Automation Studio*

Ángela Soriano Cuesta

Curso 2022/23

En el siguiente documento se muestra el código completo del programa Automation Studio, en el cual se pueden distinguir cuatro módulos de control: *Motion 1*, *Motion 2*, *Server* y *Main Program.* La mayor parte de estos ha sido provista por el fabricante, la empresa B&R, modificando y añadiendo las partes específicas relacionadas con el proyecto.

### *1.1. Motion 1*

```
PROGRAM _INIT

// Axis reference:
Axis1Obj := ADR(gAxis01);

AxisStep := STATE_WAIT; (* start step *) //estado inicial de espera

//Inicialización de parámetros que más tarde pueden ser modificados
BasicControl.Parameter.Velocity             := 1000;  //velocity
for movement
BasicControl.Parameter.Acceleration         := 5000;
//acceleration for movement
BasicControl.Parameter.Deceleration         := 5000;
//deceleration for movement
BasicControl.Parameter.JogVelocity          := 400;   //velocity
for jogging
END_PROGRAM

PROGRAM _CYCLIC

(*************************************************************
    Control Sequence

*************************************************************)
(* status information is read before the step sequencer to attain a
shorter reaction time *)
(*Almacena en su propia estructura de variables "Basicontrol" los
distintos estados leídos de la situación del eje*)
(*Además va dando valores a parámetros de bloques de funciones que
luego influirán en el uso*)

//INICIO DE FASE 1: LECTURA DE ESTADO DEL MOTOR

(************************ MC_READSTATUS ************************)
MC_ReadStatus_0.Enable := NOT(MC_ReadStatus_0.Error);
MC_ReadStatus_0.Axis := Axis1Obj;
MC_ReadStatus_0();
BasicControl.AxisState.Disabled := MC_ReadStatus_0.Disabled;
BasicControl.AxisState.StandStill := MC_ReadStatus_0.StandStill;
BasicControl.AxisState.Stopping := MC_ReadStatus_0.Stopping;
BasicControl.AxisState.Homing := MC_ReadStatus_0.Homing;
BasicControl.AxisState.DiscreteMotion :=
MC_ReadStatus_0.DiscreteMotion;
BasicControl.AxisState.ContinuousMotion :=
MC_ReadStatus_0.ContinuousMotion;
BasicControl.AxisState.SynchronizedMotion :=
MC_ReadStatus_0.SynchronizedMotion;
BasicControl.AxisState.ErrorStop := MC_ReadStatus_0.Errorstop;
```

```
(********************MC_BR_READDRIVESTATUS**********************)
MC_BR_ReadDriveStatus_0.Enable :=
NOT(MC_BR_ReadDriveStatus_0.Error);
MC_BR_ReadDriveStatus_0.Axis := Axis1Obj;
MC_BR_ReadDriveStatus_0.AdrDriveStatus :=
ADR(BasicControl.Status.DriveStatus);
MC_BR_ReadDriveStatus_0();

(******************** MC_READACTUALPOSITION ********************)
MC_ReadActualPosition_0.Enable :=
(NOT(MC_ReadActualPosition_0.Error));
MC_ReadActualPosition_0.Axis := Axis1Obj;
MC_ReadActualPosition_0();
IF(MC_ReadActualPosition_0.Valid = TRUE)THEN
      BasicControl.Status.ActPosition :=
      MC_ReadActualPosition_0.Position;
END_IF

(******************** MC_READACTUALVELOCITY ********************)
MC_ReadActualVelocity_0.Enable:=
(NOT(MC_ReadActualVelocity_0.Error));
MC_ReadActualVelocity_0.Axis := Axis1Obj;
MC_ReadActualVelocity_0();
IF(MC_ReadActualVelocity_0.Valid = TRUE)THEN
      BasicControl.Status.ActVelocity :=
      MC_ReadActualVelocity_0.Velocity;
      ActualVelocity := BasicControl.Status.ActVelocity;
      //le mandamos el valor actual de la velocidad a la variable
      ActualVelocity
END_IF

(************************ MC_READAXISERROR *********************)
MC_ReadAxisError_0.Enable := NOT(MC_ReadAxisError_0.Error);
MC_ReadAxisError_0.Axis := Axis1Obj;
MC_ReadAxisError_0.DataAddres:= ADR(BasicControl.Status.ErrorText);
MC_ReadAxisError_0.DataLength:=
SIZEOF(BasicControl.Status.ErrorText);
MC_ReadAxisError_0.DataObjectName := 'acp10etxen';
MC_ReadAxisError_0();

(**************** CHECK FOR GENERAL AXIS ERROR ******************)
IF ((MC_ReadAxisError_0.AxisErrorID <> 0) AND
(MC_ReadAxisError_0.Valid = TRUE)) THEN
      AxisStep := STATE_ERROR_AXIS;
(***************** CHECK IF POWER SHOULD BE OFF*******************)
ELSIF ((BasicControl.Command.Power = FALSE) AND
(MC_ReadAxisError_0.Valid = TRUE)) THEN
      IF ((MC_ReadStatus_0.Errorstop = TRUE) AND
      (MC_ReadStatus_0.Valid = TRUE)) THEN
                  AxisStep := STATE_ERROR_RESET;
      ELSE
                  AxisStep := STATE_WAIT;
      END_IF
END_IF

(* central monitoring OF stop command attains a shorter reaction
TIME in CASE OF emergency stop *)
```

```
      //Controla el comando stop en caso de que sea usado
(*******************CHECK FOR STOP COMMAND************************)
      IF (BasicControl.Command.Stop = TRUE) THEN
            IF ((AxisStep >= STATE_HOME) AND (AxisStep <=
            STATE_ERROR)) THEN
                  (* reset all FB execute inputs we use *)
                  MC_Home_0.Execute := 0;
                  MC_Stop_0.Execute := 0;
                  MC_MoveAbsolute_0.Execute := 0;
                  MC_MoveAdditive_0.Execute := 0;
                  MC_MoveVelocity_0.Execute := 0;
                  MC_ReadAxisError_0.Acknowledge := 0;
                  MC_Reset_0.Execute := 0;
                  MC_Halt_0.Execute := 0;

                  (* reset user commands *)
                  BasicControl.Command.Halt := 0;
                  BasicControl.Command.Home := 0;
                  BasicControl.Command.MoveJogPos := 0;
                  BasicControl.Command.MoveJogNeg := 0;
                  BasicControl.Command.MoveVelocity := 0;
                  BasicControl.Command.MoveAbsolute := 0;
                  BasicControl.Command.MoveAdditive := 0;
                  AxisStep := STATE_STOP;
            END_IF
      END_IF
      //FIN DE LA FASE 1 ERRORES ESTADO DE FB



//INICIO DE FASE 2
//Fase de definición de parámetros que participan en la ejecución
del movimiento del motor
      CASE AxisStep OF

            (******************** WAIT ************************)
            STATE_WAIT:  (* STATE: Wait -> AxisStep=0; *) //
                  IF (BasicControl.Command.Power = TRUE) THEN
                        AxisStep := STATE_POWER_ON;   // Si le
            mandamos la orden POWER, se pasará al estado de POWER ON
                  ELSE
                        MC_Power_0.Enable := FALSE;
                  END_IF

                  (* reset all FB execute inputs we use *)
                  MC_Home_0.Execute := FALSE;
                  MC_Stop_0.Execute := FALSE;
                  MC_MoveAbsolute_0.Execute := FALSE;
                  MC_MoveAdditive_0.Execute := FALSE;
                  MC_MoveVelocity_0.Execute := FALSE;
                  MC_Halt_0.Execute := FALSE;
                  MC_ReadAxisError_0.Acknowledge := FALSE;
                  MC_Reset_0.Execute := FALSE;

                  (* reset user commands *)
                  BasicControl.Command.Stop := FALSE;
                  BasicControl.Command.Halt := FALSE;
                  BasicControl.Command.Home := FALSE;
```

```
      BasicControl.Command.MoveJogPos := FALSE;
      BasicControl.Command.MoveJogNeg := FALSE;
      BasicControl.Command.MoveVelocity := FALSE;
      BasicControl.Command.MoveAbsolute := FALSE;
      BasicControl.Command.MoveAdditive := FALSE;

      BasicControl.Status.ErrorID := 0;

(******************** POWER ON **********************)
STATE_POWER_ON:  (* STATE: Power on -> AxisStep=1 *)
      MC_Power_0.Enable := TRUE;
      IF (MC_Power_0.Status = TRUE) THEN
            AxisStep := STATE_READY;//Verifica que el
            motor esté encendido y nos manda al estado de
            READY
      END_IF
      (* if a power error occured go to error state *)
      IF (MC_Power_0.Error = TRUE) THEN
            BasicControl.Status.ErrorID :=
            MC_Power_0.ErrorID;
            AxisStep := STATE_ERROR;
      END_IF

(******************** READY **********************)
STATE_READY:(*STATE: Waiting for commands->AxisStep=10*)

IF (BasicControl.Command.Home = TRUE)THEN
      BasicControl.Command.Home := FALSE;
      AxisStep := STATE_HOME;

ELSIF (BasicControl.Command.Stop = TRUE) THEN
      AxisStep := STATE_STOP;

ELSIF (BasicControl.Command.MoveJogPos = TRUE) THEN
      AxisStep := STATE_JOG_POSITIVE;

ELSIF (BasicControl.Command.MoveJogNeg = TRUE) THEN
      AxisStep := STATE_JOG_NEGATIVE;

ELSIF (BasicControl.Command.MoveAbsolute = TRUE) THEN
      BasicControl.Command.MoveAbsolute := FALSE;
      AxisStep := STATE_MOVE_ABSOLUTE;

ELSIF (BasicControl.Command.MoveAdditive = TRUE) THEN
      BasicControl.Command.MoveAdditive := FALSE;
      AxisStep := STATE_MOVE_ADDITIVE;

ELSIF (BasicControl.Command.MoveVelocity = TRUE) THEN
      BasicControl.Command.MoveVelocity := FALSE;
      AxisStep := STATE_MOVE_VELOCITY;

ELSIF (BasicControl.Command.Halt = TRUE) THEN
      BasicControl.Command.Halt := FALSE;
      AxisStep := STATE_HALT;

ELSIF (BasicControl.Command.PowerOff) THEN
            BasicControl.Command.PowerOff := FALSE;
            AxisStep := STATE_POWER_OFF;
```

```
END_IF


(******************** POWER OFF **********************)
STATE_POWER_OFF:
      MC_Power_0.Enable := FALSE;
      IF (MC_Power_0.Status = FALSE) THEN
            AxisStep := STATE_WAIT;//Verifica que el
            motor se apague y vuelve a la fase wait
      END_IF
      (* gestiona el error *)
      IF (MC_Power_0.Error = TRUE) THEN
            BasicControl.Status.ErrorID :=
            MC_Power_0.ErrorID;
            AxisStep := STATE_ERROR;
      END_IF

(******************** HOME **********************)
STATE_HOME:(* STATE: start homing process ->AxisStep=2*)
      MC_Home_0.Position :=
      BasicControl.Parameter.HomePosition;
      MC_Home_0.HomingMode :=
      BasicControl.Parameter.HomeMode;
      MC_Home_0.Execute := TRUE;
      IF (MC_Home_0.Done = TRUE) THEN
            MC_Home_0.Execute := FALSE;
            AxisStep := STATE_READY;//Volverá
            automaticamente al estado de READY una vez
            ejecutado
      END_IF
      (* if a homing error occured go to error state *)
      IF (MC_Home_0.Error = TRUE) THEN
            MC_Home_0.Execute := FALSE;
            BasicControl.Status.ErrorID :=
            MC_Home_0.ErrorID;
            AxisStep := STATE_ERROR;
      END_IF


(**********************HALT_MOVEMENT**********************)
      STATE_HALT:   (* STATE: Halt movement *) // Para el
motor y vuelve a STATE_READY para realizar otra acción
      MC_Halt_0.Deceleration :=
      BasicControl.Parameter.Deceleration;
      MC_Halt_0.Execute := TRUE;
      IF (MC_Halt_0.Done = TRUE) THEN
            MC_Halt_0.Execute := FALSE;
            AxisStep := STATE_READY; //Una vez el motor
            está parado, vuelve al estado de READY
      END_IF
      (* check if error occured *)
      IF (MC_Halt_0.Error = TRUE) THEN
            BasicControl.Status.ErrorID :=
            MC_Halt_0.ErrorID;
            MC_Halt_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
      END_IF
```

```
(********************** STOP MOVEMENT**********************)
     STATE_STOP:  (* STATE: Stop movement *) // Para el motor
     hasta que se retire la acciÛn TRUE
          MC_Stop_0.Deceleration :=
          BasicControl.Parameter.Deceleration;
          MC_Stop_0.Execute := TRUE;
          (* if axis is stopped go to ready state *)
          IF ((MC_Stop_0.Done = TRUE) AND
          (BasicControl.Command.Stop = FALSE)) THEN
               MC_Stop_0.Execute := FALSE;
               AxisStep := STATE_READY;
          END_IF
          (* check if error occured *)
          IF (MC_Stop_0.Error = TRUE) THEN
               BasicControl.Status.ErrorID :=
               MC_Stop_0.ErrorID;
               MC_Stop_0.Execute := FALSE;
               AxisStep := STATE_ERROR;
          END_IF

(*************** START JOG MOVEMENT POSITVE***************)
     STATE_JOG_POSITIVE:  (* STATE: Start jog movement in
     positive direction *) // Su movimiento para unicamente
     con STOP
     // Haciendo BC.Parameter.JV= FALSE de nuevo que hace el
     equivalente a un Halt. Halt no actua
          MC_MoveVelocity_0.Velocity      :=
          BasicControl.Parameter.JogVelocity;
          MC_MoveVelocity_0.Acceleration  :=
          BasicControl.Parameter.Acceleration;
          MC_MoveVelocity_0.Deceleration  :=
          BasicControl.Parameter.Deceleration;
          MC_MoveVelocity_0.Direction     := mcPOSITIVE_DIR;
          MC_MoveVelocity_0.Execute := TRUE;
          IF (BasicControl.Command.MoveJogPos = FALSE) THEN
               MC_MoveVelocity_0.Execute := FALSE;
               AxisStep := STATE_HALT;
          END_IF
          (* check if error occured *)
          IF (MC_MoveVelocity_0.Error = TRUE) THEN
               BasicControl.Status.ErrorID :=
               MC_MoveVelocity_0.ErrorID;
               MC_MoveVelocity_0.Execute := FALSE;
               AxisStep := STATE_ERROR;
          END_IF

(************** START JOG MOVEMENT NEGATIVE ***************)
     STATE_JOG_NEGATIVE:  (* STATE: Start jog movement in
     negative direction *)
          MC_MoveVelocity_0.Velocity      :=
          BasicControl.Parameter.JogVelocity;
          MC_MoveVelocity_0.Acceleration  :=
          BasicControl.Parameter.Acceleration;
          MC_MoveVelocity_0.Deceleration  :=
          BasicControl.Parameter.Deceleration;
          MC_MoveVelocity_0.Direction     := mcNEGATIVE_DIR;
          MC_MoveVelocity_0.Execute := TRUE;
          IF (BasicControl.Command.MoveJogNeg = FALSE) THEN
```

```
                MC_MoveVelocity_0.Execute := FALSE;
                AxisStep := STATE_HALT;
            END_IF
            (* check if error occured *)
            IF (MC_MoveVelocity_0.Error = TRUE) THEN
                BasicControl.Status.ErrorID :=
                MC_MoveVelocity_0.ErrorID;
                MC_MoveVelocity_0.Execute := FALSE;
                AxisStep := STATE_ERROR;
            END_IF

(***************** START ABSOLUTE MOVEMENT *****************)
      STATE_MOVE_ABSOLUTE:(* STATE: Start absolute movement *)
            MC_MoveAbsolute_0.Position      :=
            BasicControl.Parameter.Position;
            MC_MoveAbsolute_0.Velocity      :=
            BasicControl.Parameter.Velocity;
            MC_MoveAbsolute_0.Acceleration  :=
            BasicControl.Parameter.Acceleration;
            MC_MoveAbsolute_0.Deceleration  :=
            BasicControl.Parameter.Deceleration;
            MC_MoveAbsolute_0.Direction     :=
            BasicControl.Parameter.Direction;
            MC_MoveAbsolute_0.Execute := TRUE;
            (* check if commanded position is reached *)
            IF (BasicControl.Command.Halt) THEN
                BasicControl.Command.Halt := FALSE;
                MC_MoveAbsolute_0.Execute := FALSE;
                AxisStep := STATE_HALT;
            ELSIF (MC_MoveAbsolute_0.Done = TRUE) THEN
                MC_MoveAbsolute_0.Execute := FALSE;
                AxisStep := STATE_READY;
            END_IF
            (* check if error occured *)
            IF (MC_MoveAbsolute_0.Error = TRUE) THEN
                BasicControl.Status.ErrorID :=
                MC_MoveAbsolute_0.ErrorID;
                MC_MoveAbsolute_0.Execute := FALSE;
                AxisStep := STATE_ERROR;
            END_IF

(*************** START ADDITIVE MOVEMENT ******************)
      STATE_MOVE_ADDITIVE:(* STATE: Start additive movement *)
            MC_MoveAdditive_0.Distance      :=
            BasicControl.Parameter.Distance;
            MC_MoveAdditive_0.Velocity      :=
            BasicControl.Parameter.Velocity;
            MC_MoveAdditive_0.Acceleration  :=
            BasicControl.Parameter.Acceleration;
            MC_MoveAdditive_0.Deceleration  :=
            BasicControl.Parameter.Deceleration;
            MC_MoveAdditive_0.Execute := TRUE;
            (* check if commanded distance is reached *)
            IF (BasicControl.Command.Halt) THEN
                BasicControl.Command.Halt := FALSE;
                MC_MoveAdditive_0.Execute := FALSE;
                AxisStep := STATE_HALT;
            ELSIF (MC_MoveAdditive_0.Done = TRUE) THEN
```

```
                    MC_MoveAdditive_0.Execute := FALSE;
                    AxisStep := STATE_READY;
               END_IF
               (* check if error occured *)
               IF (MC_MoveAdditive_0.Error = TRUE) THEN
                    BasicControl.Status.ErrorID :=
                    MC_MoveAdditive_0.ErrorID;
                    MC_MoveAdditive_0.Execute := FALSE;
                    AxisStep := STATE_ERROR;
               END_IF

(**************** START VELOCITY MOVEMENT ******************)
     STATE_MOVE_VELOCITY:(* STATE: Start velocity movement ->
     AxisStep=16 *)
               MC_MoveVelocity_0.Velocity      :=
               BasicControl.Parameter.Velocity;
               MC_MoveVelocity_0.Acceleration  :=
               BasicControl.Parameter.Acceleration;
               MC_MoveVelocity_0.Deceleration  :=
               BasicControl.Parameter.Deceleration;
               MC_MoveVelocity_0.Direction     :=
               BasicControl.Parameter.Direction;
               MC_MoveVelocity_0.Execute := TRUE;
               (* check if commanded velocity is reached *)
               IF (BasicControl.Command.Halt) THEN //Si le llega
               la orden de parar, dejar· de moverse el motor y
               pasará al estado de HALT
                    BasicControl.Command.Halt := FALSE;
                    MC_MoveVelocity_0.Execute := FALSE;
                    AxisStep := STATE_HALT;
               ELSIF (MC_MoveVelocity_0.InVelocity = TRUE) THEN
                    MC_MoveVelocity_0.Execute := FALSE;
                    AxisStep := STATE_READY;
               END_IF
               (* check if error occured *)
               IF (MC_MoveVelocity_0.Error = TRUE) THEN
                    BasicControl.Status.ErrorID :=
                    MC_MoveVelocity_0.ErrorID;
                    MC_MoveVelocity_0.Execute := FALSE;
                    AxisStep := STATE_ERROR;
               END_IF

(****************** FB-ERROR OCCURED ***********************)
     STATE_ERROR:  (* STATE: Error *)
     (* check if FB indicates an axis error -> AxisStep=100*)
     IF (MC_ReadAxisError_0.AxisErrorCount<>0) THEN
               AxisStep := STATE_ERROR_AXIS;
     ELSE
               IF (BasicControl.Command.ErrorAcknowledge = TRUE)
               THEN
               BasicControl.Command.ErrorAcknowledge := FALSE;
               BasicControl.Status.ErrorID := 0;
               (* reset axis if it is in axis state ErrorStop *)
                    IF ((MC_ReadStatus_0.Errorstop = TRUE) AND
                    (MC_ReadStatus_0.Valid = TRUE)) THEN
                         AxisStep := STATE_ERROR_RESET;
                    ELSE
                         AxisStep := STATE_WAIT;
```

```
                END_IF
            END_IF
      END_IF

(****************** AXIS-ERROR OCCURED*********************)
STATE_ERROR_AXIS:  (* STATE: Axis Error *)
IF (MC_ReadAxisError_0.Valid = TRUE) THEN
      IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN
            BasicControl.Status.ErrorID :=
            MC_ReadAxisError_0.AxisErrorID;
      END_IF
      MC_ReadAxisError_0.Acknowledge := FALSE;
      IF (BasicControl.Command.ErrorAcknowledge = TRUE) THEN
            BasicControl.Command.ErrorAcknowledge := FALSE;
            (* acknowledge axis error *)
            IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN
                  MC_ReadAxisError_0.Acknowledge := TRUE;
            END_IF
      END_IF
      IF (MC_ReadAxisError_0.AxisErrorCount = 0) THEN
      (* reset axis if it is in axis state ErrorStop *)
            BasicControl.Status.ErrorID := 0;
            IF ((MC_ReadStatus_0.Errorstop = TRUE) AND
            (MC_ReadStatus_0.Valid = TRUE)) THEN
                  AxisStep := STATE_ERROR_RESET;
            ELSE
                  AxisStep := STATE_WAIT;
            END_IF
      END_IF
END_IF

(******************** RESET DONE ************************)
      STATE_ERROR_RESET:  (* STATE: Wait for reset done *)
            MC_Reset_0.Execute := TRUE;
            (* reset MC_Power.Enable if this FB is in Error*)
            IF (MC_Power_0.Error = TRUE) THEN
                  MC_Power_0.Enable := FALSE;
            END_IF
            IF(MC_Reset_0.Done = TRUE)THEN
                  MC_Reset_0.Execute := FALSE;
                  AxisStep := STATE_WAIT;
            ELSIF(MC_Reset_0.Error = TRUE) THEN
                  MC_Reset_0.Execute := FALSE;
                  AxisStep := STATE_ERROR;
            END_IF
(******************** SEQUENCE END ************************)
END_CASE

(*FINAL DE FASE 2/3 CASE*)

(*************************************************************
***
Function Block Calls
*************************************************************
**)
(*INICIO DE FASE 3/3 Llamando a funciones*)
(*Aquí es donde vamos a hacer uso de las funciones. Es la
parte que modificaremos y la que nos va a interesar*)
```

```
(************************ MC_POWER ************************)
      MC_Power_0.Axis := Axis1Obj;  (* pointer to axis *)
      MC_Power_0();

      (************************** MC_HOME
*****************************)
      MC_Home_0.Axis := Axis1Obj;
      MC_Home_0();

      (********************** MC_MOVEABSOLUTE
************************)
      MC_MoveAbsolute_0.Axis := Axis1Obj;
      MC_MoveAbsolute_0();

      (********************** MC_MOVEADDITIVE
************************)
      MC_MoveAdditive_0.Axis := Axis1Obj;
      MC_MoveAdditive_0();

      (********************** MC_MOVEVELOCITY
************************)
      MC_MoveVelocity_0.Axis := Axis1Obj;
      MC_MoveVelocity_0();

      (************************** MC_STOP
*****************************)
      MC_Stop_0.Axis := Axis1Obj;
      MC_Stop_0();

      (**************************MC_HALT**************************
****)
      MC_Halt_0.Axis := Axis1Obj;
      MC_Halt_0();

      (************************** MC_RESET
*****************************)
      MC_Reset_0.Axis := Axis1Obj;
      MC_Reset_0();



      IF NOT DatoCambiado_Vel THEN

            BasicControl.Parameter.Velocity := NuevaVelocidad; //le
            damos el nuevo valor de velocidad
            BasicControl.Command.MoveVelocity := TRUE;

            DatoCambiado_Vel := TRUE; //avisamos que el dato de
            velocidad ha sido cambiado

      END_IF



      IF NOT AccionTerminada THEN //cuando la acción está en
      proceso
```

```
      CASE AccionRecibida OF //dependiendo del valor de la accion
      recibida realizar· una orden u otra
                  5:
      (************************ MC_POWER ***********************)

                        IF MC_Power_0.Status THEN
                              BasicControl.Command.Power := FALSE;
                        ELSE
                              BasicControl.Command.Power := TRUE;
                        END_IF

                        AccionTerminada := TRUE; //la acción ha
                        acabado de ejecutarse

                  7:
      (************************ MC_HOME ************************)

                  BasicControl.Command.Home := TRUE;
                  AccionTerminada := TRUE;

                  9:
      (************************ MC_RESET ***********************)

                  BasicControl.Command.ErrorAcknowledge := TRUE;

                  AccionTerminada := TRUE;

                  11:

      (************************MC_HALT*************************)

                        BasicControl.Command.Halt := TRUE;

                        AccionTerminada := TRUE;

                  12:
      (******************* MC_MOVEVELOCITY ***********************)

                        BasicControl.Command.MoveVelocity := TRUE;

                        AccionTerminada := TRUE;

                  ELSE
                        // nothing
            END_CASE

      END_IF

      Paso := AxisStep;
      Power := BasicControl.Command.Power;



END_PROGRAM
```

### *1.2. Motion 2*

```
PROGRAM _INIT

(* get axis object *)
Axis2Obj := ADR(gAxis02);

AxisStep := STATE_WAIT; (* start step *)

BasicControl.Parameter.Velocity := 1000; (*velocity for movement*)
BasicControl.Parameter.Acceleration := 5000; (*acceleration for
movement*)
BasicControl.Parameter.Deceleration := 5000; (*deceleration for
movement*)
BasicControl.Parameter.JogVelocity  := 400; (*velocity for jogging
*)
END_PROGRAM

PROGRAM _CYCLIC

(*************************************************************
        Control Sequence
*************************************************************)
(* status information is read before the step sequencer to attain a
shorter reaction time *)
(************************ MC_READSTATUS ************************)
MC_ReadStatus_0.Enable := NOT(MC_ReadStatus_0.Error);
MC_ReadStatus_0.Axis := Axis2Obj;
MC_ReadStatus_0();
BasicControl.AxisState.Disabled  := MC_ReadStatus_0.Disabled;
BasicControl.AxisState.StandStill   := MC_ReadStatus_0.StandStill;
BasicControl.AxisState.Stopping  := MC_ReadStatus_0.Stopping;
BasicControl.AxisState.Homing   := MC_ReadStatus_0.Homing;
BasicControl.AxisState.DiscreteMotion  :=
MC_ReadStatus_0.DiscreteMotion;
BasicControl.AxisState.ContinuousMotion   :=
MC_ReadStatus_0.ContinuousMotion;
BasicControl.AxisState.SynchronizedMotion  :=
MC_ReadStatus_0.SynchronizedMotion;
BasicControl.AxisState.ErrorStop  := MC_ReadStatus_0.Errorstop;

(********************MC_BR_READDRIVESTATUS**********************)
MC_BR_ReadDriveStatus_0.Enable :=
NOT(MC_BR_ReadDriveStatus_0.Error);
MC_BR_ReadDriveStatus_0.Axis := Axis2Obj;
MC_BR_ReadDriveStatus_0.AdrDriveStatus :=
ADR(BasicControl.Status.DriveStatus);
MC_BR_ReadDriveStatus_0();

(******************** MC_READACTUALPOSITION ********************)
MC_ReadActualPosition_0.Enable :=
(NOT(MC_ReadActualPosition_0.Error));
MC_ReadActualPosition_0.Axis := Axis2Obj;
MC_ReadActualPosition_0();
IF(MC_ReadActualPosition_0.Valid = TRUE)THEN
    BasicControl.Status.ActPosition :=
MC_ReadActualPosition_0.Position;
END_IF
```

```
(******************** MC_READACTUALVELOCITY ********************)
MC_ReadActualVelocity_0.Enable :=
(NOT(MC_ReadActualVelocity_0.Error));
MC_ReadActualVelocity_0.Axis := Axis2Obj;
MC_ReadActualVelocity_0();
IF(MC_ReadActualVelocity_0.Valid = TRUE)THEN
    BasicControl.Status.ActVelocity :=
MC_ReadActualVelocity_0.Velocity;
END_IF

(******************** MC_READACTUALTORQUE ********************)
MC_ReadActualTorque_0.Enable := (NOT(MC_ReadActualTorque_0.Error));
MC_ReadActualTorque_0.Axis :=  Axis2Obj;
MC_ReadActualTorque_0();

IF(MC_ReadActualTorque_0.Valid = TRUE) THEN
          BasicControl.Status.ActTorque :=
MC_ReadActualTorque_0.Torque;
          ActualTorque := BasicControl.Status.ActTorque;
END_IF

(************************ MC_READAXISERROR ********************)
MC_ReadAxisError_0.Enable := NOT(MC_ReadAxisError_0.Error);
MC_ReadAxisError_0.Axis := Axis2Obj;
MC_ReadAxisError_0.DataAddress :=
ADR(BasicControl.Status.ErrorText);
MC_ReadAxisError_0.DataLength :=
SIZEOF(BasicControl.Status.ErrorText);
MC_ReadAxisError_0.DataObjectName := 'acp10etxen';
MC_ReadAxisError_0();

(***************** CHECK FOR GENERAL AXIS ERROR ******************)
    IF ((MC_ReadAxisError_0.AxisErrorID <> 0) AND
    (MC_ReadAxisError_0.Valid = TRUE)) THEN
        AxisStep := STATE_ERROR_AXIS;
(****************** CHECK IF POWER SHOULD BE OFF ******************)
    ELSIF ((BasicControl.Command.Power = FALSE) AND
    (MC_ReadAxisError_0.Valid = TRUE)) THEN
        IF ((MC_ReadStatus_0.Errorstop = TRUE) AND
        MC_ReadStatus_0.Valid = TRUE)) THEN
            AxisStep := STATE_ERROR_RESET;
        ELSE
            AxisStep := STATE_WAIT;
        END_IF
    END_IF

(* central monitoring OF stop command attains a shorter reaction
TIME in CASE OF emergency stop *)
(******************CHECK FOR STOP COMMAND***********************)
 IF (BasicControl.Command.Stop = TRUE) THEN
    IF ((AxisStep >= STATE_HOME) AND (AxisStep <= STATE_ERROR))THEN
            (* reset all FB execute inputs we use *)
            MC_Home_0.Execute := 0;
            MC_Stop_0.Execute := 0;
            MC_MoveAbsolute_0.Execute := 0;
            MC_MoveAdditive_0.Execute := 0;
            MC_MoveVelocity_0.Execute := 0;
```

```
            MC_ReadAxisError_0.Acknowledge := 0;
            MC_Reset_0.Execute := 0;
            MC_Halt_0.Execute := 0;

            (* reset user commands *)
            BasicControl.Command.Halt := 0;
            BasicControl.Command.Home := 0;
            BasicControl.Command.MoveJogPos := 0;
            BasicControl.Command.MoveJogNeg := 0;
            BasicControl.Command.MoveVelocity := 0;
            BasicControl.Command.MoveAbsolute := 0;
            BasicControl.Command.MoveAdditive := 0;
            AxisStep := STATE_STOP;
        END_IF
    END_IF

CASE AxisStep OF

(******************** WAIT ************************)
    STATE_WAIT:  (* STATE: Wait *)
        IF (BasicControl.Command.Power = TRUE) THEN
            AxisStep := STATE_POWER_ON;
        ELSE
            MC_Power_0.Enable := FALSE;
        END_IF

        (* reset all FB execute inputs we use *)
        MC_Home_0.Execute := FALSE;
        MC_Stop_0.Execute := FALSE;
        MC_MoveAbsolute_0.Execute := FALSE;
        MC_MoveAdditive_0.Execute := FALSE;
        MC_MoveVelocity_0.Execute := FALSE;
        MC_Halt_0.Execute := FALSE;
        MC_ReadAxisError_0.Acknowledge := FALSE;
        MC_Reset_0.Execute := FALSE;

        (* reset user commands *)
        BasicControl.Command.Stop := FALSE;
        BasicControl.Command.Halt := FALSE;
        BasicControl.Command.Home := FALSE;
        BasicControl.Command.MoveJogPos := FALSE;
        BasicControl.Command.MoveJogNeg := FALSE;
        BasicControl.Command.MoveVelocity := FALSE;
        BasicControl.Command.MoveAbsolute := FALSE;
        BasicControl.Command.MoveAdditive := FALSE;

        BasicControl.Status.ErrorID := 0;

(******************** POWER ON ********************)
    STATE_POWER_ON:  (* STATE: Power on *)
        MC_Power_0.Enable := TRUE;
        IF (MC_Power_0.Status = TRUE) THEN
            AxisStep := STATE_READY;
        END_IF
        (* if a power error occured go to error state *)
        IF (MC_Power_0.Error = TRUE) THEN
            BasicControl.Status.ErrorID := MC_Power_0.ErrorID;
            AxisStep := STATE_ERROR;
```

```
        END_IF

(******************** READY **********************)
    STATE_READY:  (* STATE: Waiting for commands *)
        IF (BasicControl.Command.Home = TRUE)THEN
            BasicControl.Command.Home := FALSE;
            AxisStep := STATE_HOME;

        ELSIF (BasicControl.Command.Stop = TRUE) THEN
            AxisStep := STATE_STOP;

        ELSIF (BasicControl.Command.MoveJogPos = TRUE) THEN
            AxisStep := STATE_JOG_POSITIVE;

        ELSIF (BasicControl.Command.MoveJogNeg = TRUE) THEN
            AxisStep := STATE_JOG_NEGATIVE;

        ELSIF (BasicControl.Command.MoveAbsolute = TRUE) THEN
            BasicControl.Command.MoveAbsolute := FALSE;
            AxisStep := STATE_MOVE_ABSOLUTE;

        ELSIF (BasicControl.Command.MoveAdditive = TRUE) THEN
            BasicControl.Command.MoveAdditive := FALSE;
            AxisStep := STATE_MOVE_ADDITIVE;

        ELSIF (BasicControl.Command.MoveVelocity = TRUE) THEN
            BasicControl.Command.MoveVelocity := FALSE;
            AxisStep := STATE_MOVE_VELOCITY;

        ELSIF (BasicControl.Command.Halt = TRUE) THEN
            BasicControl.Command.Halt := FALSE;
                 AxisStep := STATE_HALT;


(******************** HOME **********************)
    STATE_HOME:  (* STATE: start homing process *)
        MC_Home_0.Position := BasicControl.Parameter.HomePosition;
        MC_Home_0.HomingMode := BasicControl.Parameter.HomeMode;
        MC_Home_0.Execute := TRUE;
        IF (MC_Home_0.Done = TRUE) THEN
            MC_Home_0.Execute := FALSE;
            AxisStep := STATE_READY;
        END_IF
        (* if a homing error occured go to error state *)
        IF (MC_Home_0.Error = TRUE) THEN
            MC_Home_0.Execute := FALSE;
            BasicControl.Status.ErrorID := MC_Home_0.ErrorID;
            AxisStep := STATE_ERROR;
        END_IF

(**********************HALT_MOVEMENT**************************)
    STATE_HALT:   (* STATE: Halt movement *)
        MC_Halt_0.Deceleration :=
BasicControl.Parameter.Deceleration;
        MC_Halt_0.Execute := TRUE;
        IF (MC_Halt_0.Done = TRUE) THEN
            MC_Halt_0.Execute := FALSE;
            AxisStep := STATE_READY;
```

```
        END_IF
        (* check if error occured *)
        IF (MC_Halt_0.Error = TRUE) THEN
            BasicControl.Status.ErrorID := MC_Halt_0.ErrorID;
            MC_Halt_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF

(********************** STOP MOVEMENT ************************)
    STATE_STOP:  (* STATE: Stop movement *)
        MC_Stop_0.Deceleration :=
BasicControl.Parameter.Deceleration;
        MC_Stop_0.Execute := TRUE;
        (* if axis is stopped go to ready state *)
        IF ((MC_Stop_0.Done = TRUE) AND (BasicControl.Command.Stop
= FALSE)) THEN
            MC_Stop_0.Execute := FALSE;
            AxisStep := STATE_READY;
        END_IF
        (* check if error occured *)
        IF (MC_Stop_0.Error = TRUE) THEN
            BasicControl.Status.ErrorID := MC_Stop_0.ErrorID;
            MC_Stop_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF

(****************** START JOG MOVEMENT POSITVE ******************)
     STATE_JOG_POSITIVE:  (* STATE: Start jog movement in positive
     direction *)
        MC_MoveVelocity_0.Velocity      :=
        BasicControl.Parameter.JogVelocity;
        MC_MoveVelocity_0.Acceleration  :=
        BasicControl.Parameter.Acceleration;
        MC_MoveVelocity_0.Deceleration  :=
        BasicControl.Parameter.Deceleration;
        MC_MoveVelocity_0.Direction     := mcPOSITIVE_DIR;
        MC_MoveVelocity_0.Execute := TRUE;
        IF (BasicControl.Command.MoveJogPos = FALSE) THEN
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_HALT;
        END_IF
        (* check if error occured *)
        IF (MC_MoveVelocity_0.Error = TRUE) THEN
           BasicControl.Status.ErrorID :=
           MC_MoveVelocity_0.ErrorID;
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF

(****************** START JOG MOVEMENT NEGATIVE ******************)
    STATE_JOG_NEGATIVE: (* STATE: Start jog movement in negative
    direction *)
        MC_MoveVelocity_0.Velocity      :=
        BasicControl.Parameter.JogVelocity;
        MC_MoveVelocity_0.Acceleration  :=
        BasicControl.Parameter.Acceleration;
        MC_MoveVelocity_0.Deceleration  :=
        BasicControl.Parameter.Deceleration;
```

```
        MC_MoveVelocity_0.Direction        := mcNEGATIVE_DIR;
        MC_MoveVelocity_0.Execute := TRUE;
        IF (BasicControl.Command.MoveJogNeg = FALSE) THEN
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_HALT;
        END_IF
        (* check if error occured *)
        IF (MC_MoveVelocity_0.Error = TRUE) THEN
            BasicControl.Status.ErrorID :=
           MC_MoveVelocity_0.ErrorID;
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF

(****************** START ABSOLUTE MOVEMENT *********************)
STATE_MOVE_ABSOLUTE:  (* STATE: Start absolute movement *)
MC_MoveAbsolute_0.Position := BasicControl.Parameter.Position;
MC_MoveAbsolute_0.Velocity := BasicControl.Parameter.Velocity;
MC_MoveAbsolute_0.Acceleration :=
BasicControl.Parameter.Acceleration;
MC_MoveAbsolute_0.Deceleration  :=
BasicControl.Parameter.Deceleration;
MC_MoveAbsolute_0.Direction := BasicControl.Parameter.Direction;
        MC_MoveAbsolute_0.Execute := TRUE;
        (* check if commanded position is reached *)
        IF (BasicControl.Command.Halt) THEN
            BasicControl.Command.Halt := FALSE;
            MC_MoveAbsolute_0.Execute := FALSE;
            AxisStep := STATE_HALT;
        ELSIF (MC_MoveAbsolute_0.Done = TRUE) THEN
            MC_MoveAbsolute_0.Execute := FALSE;
            AxisStep := STATE_READY;
        END_IF
        (* check if error occured *)
        IF (MC_MoveAbsolute_0.Error = TRUE) THEN
            BasicControl.Status.ErrorID :=
           MC_MoveAbsolute_0.ErrorID;
            MC_MoveAbsolute_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF

(******************** START ADDITIVE MOVEMENT ********************)
 STATE_MOVE_ADDITIVE:  (* STATE: Start additive movement *)
 MC_MoveAdditive_0.Distance := BasicControl.Parameter.Distance;
 MC_MoveAdditive_0.Velocity := BasicControl.Parameter.Velocity;
MC_MoveAdditive_0.Acceleration  :=
BasicControl.Parameter.Acceleration;
MC_MoveAdditive_0.Deceleration :=
BasicControl.Parameter.Deceleration;
MC_MoveAdditive_0.Execute := TRUE;
        (* check if commanded distance is reached *)
        IF (BasicControl.Command.Halt) THEN
            BasicControl.Command.Halt := FALSE;
            MC_MoveAdditive_0.Execute := FALSE;
            AxisStep := STATE_HALT;
        ELSIF (MC_MoveAdditive_0.Done = TRUE) THEN
            MC_MoveAdditive_0.Execute := FALSE;
            AxisStep := STATE_READY;
```

```
        END_IF
        (* check if error occured *)
        IF (MC_MoveAdditive_0.Error = TRUE) THEN
           BasicControl.Status.ErrorID :=
           MC_MoveAdditive_0.ErrorID;
            MC_MoveAdditive_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF

(***************** START VELOCITY MOVEMENT ********************)
    STATE_MOVE_VELOCITY:  (* STATE: Start velocity movement *)
MC_MoveVelocity_0.Velocity := 0;
MC_MoveVelocity_0.Acceleration :=
BasicControl.Parameter.Acceleration;
MC_MoveVelocity_0.Deceleration  :=
BasicControl.Parameter.Deceleration;
MC_MoveVelocity_0.Direction := BasicControl.Parameter.Direction;
MC_MoveVelocity_0.Torque := BasicControl.Parameter.Torque;

        MC_MoveVelocity_0.Execute := TRUE;
        (* check if commanded velocity is reached *)
        IF (BasicControl.Command.Halt) THEN
            BasicControl.Command.Halt := FALSE;
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_HALT;
        ELSIF (MC_MoveVelocity_0.InVelocity = TRUE) THEN
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_READY;
        END_IF
        (* check if error occured *)
        IF (MC_MoveVelocity_0.Error = TRUE) THEN
           BasicControl.Status.ErrorID :=
           MC_MoveVelocity_0.ErrorID;
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF


(******************** FB-ERROR OCCURED ************************)
    STATE_ERROR:  (* STATE: Error *)
        (* check if FB indicates an axis error *)
        IF (MC_ReadAxisError_0.AxisErrorCount<>0) THEN
            AxisStep := STATE_ERROR_AXIS;
        ELSE
            IF (BasicControl.Command.ErrorAcknowledge = TRUE) THEN
                BasicControl.Command.ErrorAcknowledge := FALSE;
                BasicControl.Status.ErrorID := 0;
                (* reset axis if it is in axis state ErrorStop *)
                 IF ((MC_ReadStatus_0.Errorstop = TRUE) AND
                 (MC_ReadStatus_0.Valid = TRUE)) THEN
                    AxisStep := STATE_ERROR_RESET;
                ELSE
                    AxisStep := STATE_WAIT;
                END_IF
            END_IF
        END_IF

(******************** AXIS-ERROR OCCURED ************************)
```

```
    STATE_ERROR_AXIS:  (* STATE: Axis Error *)
        IF (MC_ReadAxisError_0.Valid = TRUE) THEN
            IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN
                BasicControl.Status.ErrorID :=
                MC_ReadAxisError_0.AxisErrorID;
            END_IF
            MC_ReadAxisError_0.Acknowledge := FALSE;
            IF (BasicControl.Command.ErrorAcknowledge = TRUE) THEN
                BasicControl.Command.ErrorAcknowledge := FALSE;
                (* acknowledge axis error *)
                IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN
                    MC_ReadAxisError_0.Acknowledge := TRUE;
                END_IF
            END_IF
            IF (MC_ReadAxisError_0.AxisErrorCount = 0) THEN
                (* reset axis if it is in axis state ErrorStop *)
                BasicControl.Status.ErrorID := 0;
                IF ((MC_ReadStatus_0.Errorstop = TRUE) AND
                (MC_ReadStatus_0.Valid = TRUE)) THEN
                    AxisStep := STATE_ERROR_RESET;
                ELSE
                    AxisStep := STATE_WAIT;
                END_IF
            END_IF
        END_IF

(******************** RESET DONE ************************)
    STATE_ERROR_RESET:  (* STATE: Wait for reset done *)
        MC_Reset_0.Execute := TRUE;
        (* reset MC_Power.Enable if this FB is in Error*)
        IF (MC_Power_0.Error = TRUE) THEN
            MC_Power_0.Enable := FALSE;
        END_IF
        IF(MC_Reset_0.Done = TRUE)THEN
            MC_Reset_0.Execute := FALSE;
            AxisStep := STATE_WAIT;
        ELSIF(MC_Reset_0.Error = TRUE) THEN
            MC_Reset_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF
(******************** SEQUENCE END ************************)
END_CASE

(*************************************************************
        Function Block Calls
*************************************************************)

(************************** MC_POWER ****************************)
      MC_Power_0.Axis := Axis2Obj;  (* pointer to axis *)
      MC_Power_0();

      (************************** MC_HOME
*****************************)
      MC_Home_0.Axis := Axis2Obj;
      MC_Home_0();

      (********************** MC_MOVEABSOLUTE
************************)
```

```
      MC_MoveAbsolute_0.Axis := Axis2Obj;
      MC_MoveAbsolute_0();

      (********************** MC_MOVEADDITIVE
************************)
      MC_MoveAdditive_0.Axis := Axis2Obj;
      MC_MoveAdditive_0();

      (********************** MC_MOVEVELOCITY
************************)
      MC_MoveVelocity_0.Axis := Axis2Obj;
      MC_MoveVelocity_0();

      (************************** MC_STOP
*****************************)
      MC_Stop_0.Axis := Axis2Obj;
      MC_Stop_0();

      (**************************MC_HALT**************************
****)
      MC_Halt_0.Axis := Axis2Obj;
      MC_Halt_0();

      (************************** MC_RESET
*****************************)
      MC_Reset_0.Axis := Axis2Obj;
      MC_Reset_0();


      IF NOT DatoCambiado_Par THEN

            BasicControl.Parameter.Torque := NuevoPar;
            BasicControl.Command.Torque := TRUE;
            DatoCambiado_Par := 1;

      END_IF


      IF NOT AccionTerminada THEN

            CASE AccionRecibida OF
                  6:
                  IF MC_Power_0.Status THEN
                        BasicControl.Command.Power := TRUE;
                  ELSE
                        BasicControl.Command.Power := FALSE;
                  END_IF

                  AccionTerminada := TRUE;

                  8:
                  BasicControl.Command.Home := TRUE;

                  AccionTerminada := TRUE;

                  10:
                  BasicControl.Command.ErrorAcknowledge := TRUE;
```

```
                AccionTerminada := TRUE;

                13:
                BasicControl.Command.MoveVelocity := TRUE;

                AccionTerminada := TRUE;

                14:
                BasicControl.Command.Halt := TRUE;

                AccionTerminada := TRUE;



          END_CASE

     END_IF


END_PROGRAM
```

### *1.3. Server*

```
PROGRAM _INIT

     (* Server configuration *)
     gServer.ParaPort := 12010; (* TCP port on the server *)
     gServer.ParaSendTime := T#100ms;
     gServer.ParaReceiveTimeout := T#750ms;

     (* Client communication timing configuration *)
     (* Read cycle time *)
     GetRTInfo(enable := 1);
     RTResult.Status := GetRTInfo.status;
     RTResult.CycleTime := GetRTInfo.cycle_time;
     RTResult.TaskClass := GetRTInfo.task_class;
     RTResult.InitReason := GetRTInfo.init_reason;

     (* Start Logger *)
     (* Move all entries *)
     FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO

     brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[iLogEntry-
     1]),SIZEOF(ServerLog[iLogEntry]));

     END_FOR;
     (* Empty Log[0] *)
     brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
     (* Write on Log[0] *)
     GetTimeStruct(enable := 1,pDTStructure :=
     ADR(ServerLog[0].Date));
     ServerLog[0].FBK := 'NA';
     ServerLog[0].ErrorID := 0;
     gServer.Enable := TRUE; // Importante para que al entrar en
     el ciclo entre ya con el server abierto.

END_PROGRAM
```

```
PROGRAM _CYCLIC

      (* Send and timeout timer calculation *)
      IF (TIME_TO_UDINT(gServer.ParaSendTime) < (RTResult.CycleTime
      / 1000) * SEND_TIME_MULT) AND RTResult.CycleTime > 0 THEN
            gServer.ParaSendTime := (RTResult.CycleTime / 1000) *
            SEND_TIME_MULT;
      END_IF;

      IF (TIME_TO_UDINT(gServer.ParaReceiveTimeout) <
      (RTResult.CycleTime / 1000) * TIMEOUT_MULT) AND
      RTResult.CycleTime > 0 THEN
            gServer.ParaReceiveTimeout := (RTResult.CycleTime /
            1000) * TIMEOUT_MULT;
      END_IF;


CASE Server.sStep OF

      0:
            IF gServer.Enable THEN
                  gServer.Status := 1; // WAITING FOR FIRST
                  COMMUNICATION
                  Server.sStep := 5;
            END_IF;

      5:  (* Open Ethernet Interface *) // Escucha en el puerto que
      hemos dicho que iba a mandar info
            Server.TcpOpen_0.enable := 1;
            Server.TcpOpen_0.pIfAddr := 0;  (* Listen on all TCP/IP
            Interfaces*)
            Server.TcpOpen_0.port := gServer.ParaPort;  (* Port to
            listen*)
            Server.TcpOpen_0.options := tcpOPT_REUSEADDR;
            (* Allows the linking of several instances of a TCP
            server with the same port *)
            Server.TcpOpen_0;  (* Call the Function*)

            IF Server.TcpOpen_0.status = 0 THEN  (* TcpOpen
            successfull*)
                  Server.sStep := 6;
            ELSIF Server.TcpOpen_0.status = tcpERR_ALREADY_EXIST OR
            gServer.Enable = 0 THEN
                  (* Log error *)
                  FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
                  brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
                  iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

                  END_FOR;

            brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
                  GetTimeStruct(enable := 1,pDTStructure :=
                  ADR(ServerLog[0].Date));
                  ServerLog[0].FBK := 'TcpOpen_0';
                  ServerLog[0].ErrorID := Server.TcpOpen_0.status;
```

```
            (* Close communication *)
            Server.sStep := 50;
      ELSIF Server.TcpOpen_0.status = ERR_FUB_BUSY THEN  (*
      TcpOpen not finished -> redo *)
                  (* Busy *)
            ELSE
                  (* Log error *)
                  FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
            brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
            iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

                  END_FOR;

      brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
            GetTimeStruct(enable := 1,pDTStructure :=
            ADR(ServerLog[0].Date));
            ServerLog[0].FBK := 'TcpOpen_0';
            ServerLog[0].ErrorID := Server.TcpOpen_0.status;

            END_IF

6:
      Server.linger_opt.lLinger := 0; (* linger Time = 0 *)
      Server.linger_opt.lOnOff := 1; (* linger Option ON *)

      Server.TcpIoctl_0.enable := 1;
      Server.TcpIoctl_0.ident := Server.TcpOpen_0.ident;
      (* Connection Ident from AsTCP.TCP_Open *)
      Server.TcpIoctl_0.ioctl := tcpSO_LINGER_SET;
      (* Set Linger Options *)
      Server.TcpIoctl_0.pData := ADR(Server.linger_opt);
      Server.TcpIoctl_0.datalen := SIZEOF(Server.linger_opt);
      Server.TcpIoctl_0;

      IF Server.TcpIoctl_0.status = 0 THEN  (* TcpIoctl
      successfull *)
            Server.sStep := 10;

      ELSIF Server.TcpIoctl_0.status = ERR_FUB_BUSY THEN  (*
      TcpIoctl not finished -> redo *)
                  (* Busy *)
      ELSE
            (* Log error *)
            FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
            brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
            iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

            END_FOR;

      brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
            GetTimeStruct(enable := 1,pDTStructure :=
            ADR(ServerLog[0].Date));
            ServerLog[0].FBK := 'TcpIoctl_0';
            ServerLog[0].ErrorID := Server.TcpIoctl_0.status;
      END_IF

10: (* Wait for Client Connection *)
      Server.TcpServer_0.enable := 1;
```

```
        Server.TcpServer_0.ident := Server.TcpOpen_0.ident;
        (* Connection Ident from AsTCP.TCP_Open *)
        Server.TcpServer_0.backlog := 1;  (* Number of clients
        waiting simultaneously for a connection*)
        Server.TcpServer_0.pIpAddr :=ADR(Server.client_address);
        (* Where to write the client IP-Address*)
        Server.TcpServer_0;  (* Call the Function*)

        IF Server.TcpServer_0.status = 0 THEN  (* Status = 0 if
        an client connects to server *)
              Server.sStep := 15;
        ELSIF Server.TcpServer_0.status = tcpERR_INVALID_IDENT
        OR gServer.Enable = 0 THEN
              (* Log error *)
              FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
              brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
              iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

              END_FOR;

        brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
              GetTimeStruct(enable := 1,pDTStructure :=
              ADR(ServerLog[0].Date));
              ServerLog[0].FBK := 'TcpServer_0';
              ServerLog[0].ErrorID := Server.TcpServer_0.status;

              (* Close communication *)
              Server.sStep := 50;
        ELSIF Server.TcpServer_0.status = ERR_FUB_BUSY THEN
        (* TcpServer not finished -> redo *)
                    (* Busy *)
        ELSE
              (* Log error *)
              FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
              brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
              iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

              END_FOR;

        brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
              GetTimeStruct(enable := 1,pDTStructure :=
              ADR(ServerLog[0].Date));
              ServerLog[0].FBK := 'TcpServer_0';
              ServerLog[0].ErrorID := Server.TcpServer_0.status;
        END_IF

15:
        Server.TcpIoctl_0.enable := 1;
        Server.TcpIoctl_0.ident := Server.TcpServer_0.identclnt;
        (* Connection Ident from AsTCP.TCP_Server *)
        Server.TcpIoctl_0.ioctl := tcpSO_LINGER_SET;  (* Set
        Linger Options *)
        Server.TcpIoctl_0.pData := ADR(Server.linger_opt);
        Server.TcpIoctl_0.datalen := SIZEOF(Server.linger_opt);
        Server.TcpIoctl_0;
```

```
      IF Server.TcpIoctl_0.status = 0 THEN  (* TcpIoctl
      successfull *) // Aquí ya todo esta listo para el envío
      y la recepción de datos
            (* Start send timer *)
            TON_Send_Server.IN := 1;
            TON_Send_Server.PT := gServer.ParaSendTime;

            Server.sStep := 20;

      ELSIF Server.TcpIoctl_0.status = ERR_FUB_BUSY THEN  (*
      TcpIoctl not finished -> redo *)
            (* Busy *)
      ELSE
            (* Log error *)
            FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
            brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
            iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

            END_FOR;

      brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
            GetTimeStruct(enable := 1,pDTStructure :=
            ADR(ServerLog[0].Date));
            ServerLog[0].FBK := 'TcpIoctl_0';
            ServerLog[0].ErrorID := Server.TcpIoctl_0.status;
      END_IF

20: (* Wait for Data *)
      Server.TcpRecv_0.enable := 1;
      Server.TcpRecv_0.ident := Server.TcpServer_0.identclnt;
      (* Client Ident from AsTCP.TCP_Server *)
      Server.TcpRecv_0.pData := ADR(Server.data_buffer);
      (* Where to store the incoming data *)
      Server.TcpRecv_0.datamax := SIZEOF(Server.data_buffer);
      (* Length of data buffer *)
      Server.TcpRecv_0.flags := 0;
      Server.TcpRecv_0;  (* Call the Function*)

      TON_RecvTimeout_Server.IN := 1;
      TON_RecvTimeout_Server.PT := gServer.ParaReceiveTimeout;

      IF gServer.Enable = 0 OR (Server.TcpRecv_0.status =
      tcpERR_NOT_CONNECTED) THEN
            (* Log error *)
            FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
            brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
            iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

            END_FOR;

      brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
            GetTimeStruct(enable := 1,pDTStructure :=
            ADR(ServerLog[0].Date));
            ServerLog[0].FBK := 'TcpRecv_0';
            ServerLog[0].ErrorID := Server.TcpRecv_0.status;

            (* Connection lost or disabled *)
            Server.sStep := 40;
```

```
            ELSIF Server.TcpRecv_0.status = 0 THEN(* Data received*)
                  gServer.Status := 0; //COMMUNICATION OK
                  (* Reset timeout timer, since data was received *)
//                ------------------------------------------ATENTION

            ValorRecibido := UDINT_TO_UINT(Server.data_buffer[0]);
            //el valor recibido lo almacenamos en la variable
            ValorRecibido


            IF ValorRecibido.15 THEN //distinguimos si es una orden
            o un dato nuevo de velocidad o par

                  ValorRecibido.15:=FALSE;
                  DatoRecibido := ValorRecibido;    //si es un nuevo
                  dato, pasamos el valor a la variable DatoRecibido
                  DatoCambiado := 0;

            ELSIF NOT ValorRecibido.15 AND AccionTerminada THEN //si
            nos mandan una acción, esta solo podrá ejecutarse si la
            acción anterior ha terminado

                  AccionRecibida := ValorRecibido; //pasamos el
                  valor recibido a otra variable que lo reconoce
                  como acción
                  AccionTerminada :=FALSE; //la acción que nos acaba
                  de llegar ha iniciado

            END_IF

                  TON_RecvTimeout_Server.IN := 0;

                  (* Time for new communication *)
                  IF TON_Send_Server.Q THEN
                        TON_Send_Server.IN := 0;
                        (* Go to send step *) // Si no se ha recibido
                        nada en un tiempo limite pasamos a enviar.
                        Server.sStep := 30;
                  END_IF;
            ELSIF Server.TcpRecv_0.status = tcpERR_NO_DATA THEN
                  (* No data received - wait for data, timeout or
                  new communication. The client will send data
                  again, while the RecvTimeout timer is still
                  counting. If this timeout is reached, an entry
                  will be written in the communication logger. *)

                  IF TON_Send_Server.Q THEN
                        (* Reset communication timer *)
                        TON_Send_Server.IN := 0;

                        (* Go to send step *)
                        Server.sStep := 30;
                  END_IF;

                  IF TON_RecvTimeout_Server.Q THEN
                        gServer.Status := 2; //COMMUNICATION ERROR:
                        TIMEOUT
                        (* Log error *)
```

```
                FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
                brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(Serve
                rLog[iLogEntry-
                1]),SIZEOF(ServerLog[iLogEntry]));

                END_FOR;

      brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
                GetTimeStruct(enable := 1,pDTStructure :=
                ADR(ServerLog[0].Date));
                ServerLog[0].FBK := 'TcpRecv_0';
                ServerLog[0].ErrorID :=
                Server.TcpRecv_0.status;

                (* Reset timeout counter and keep waiting
                for communication timer *)
                TON_RecvTimeout_Server.IN := 0;

                (* Stay in the TcpRecv step until a new
                communication is demanded *)
          END_IF;
          ELSIF Server.TcpRecv_0.status = ERR_FUB_BUSY THEN
          (* TcpRecv not finished -> redo *)

                (* Busy *)
          ELSE
                (* Log error and wait for new send command *)
                FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
                brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(Serve
                rLog[iLogEntry-
                1]),SIZEOF(ServerLog[iLogEntry]));

                END_FOR;

      brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
                GetTimeStruct(enable := 1,pDTStructure :=
                ADR(ServerLog[0].Date));
                ServerLog[0].FBK := 'TcpRecv_0';
                ServerLog[0].ErrorID :=
                Server.TcpRecv_0.status;

                (* Time for new communication *)
                IF TON_Send_Server.Q THEN
                      TON_Send_Server.IN := 0;

                      (* Go to send step *)
                      Server.sStep := 30;
                END_IF;
          END_IF


30:   (* Send Data back to Client *)    // UaCounter será
nuestra manera de introducir data
      Server.TcpSend_0.enable := 1;
      Server.TcpSend_0.ident := Server.TcpServer_0.identclnt;
      (* Client Ident from AsTCP.TCP_Server *)
```

```
SentData := VelocidadReal; //mandaremos la velocidad ya
convertida


Server.TcpSend_0.pData := ADR(SentData);  (* Which data
to send *)
Server.TcpSend_0.datalen := SIZEOF(SentData);  (* Lenght
of data to send *)
Server.TcpSend_0.flags := 0;
Server.TcpSend_0;  (* Call the Function*)

IF gServer.Enable = 0 OR (Server.TcpSend_0.status =
tcpERR_NOT_CONNECTED) THEN
      (* Log error *)
      FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
      brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
      iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

      END_FOR;

brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
      GetTimeStruct(enable := 1,pDTStructure :=
      ADR(ServerLog[0].Date));
      ServerLog[0].FBK := 'TcpSend_0';
      ServerLog[0].ErrorID := Server.TcpSend_0.status;

      (* Connection lost or disabled *)
      Server.sStep := 40;
ELSIF Server.TcpSend_0.status = 0 THEN  (* Data sent *)
      TON_Send_Server.IN := 1;
      TON_Send_Server.PT := gServer.ParaSendTime;
      Server.sStep := 20;
      IF TON_Send_Server.Q THEN
            TON_Send_Server.IN := 0;
            (* Go to send step *)
      Server.sStep := 30;
            END_IF;

ELSIF Server.TcpSend_0.status = ERR_FUB_BUSY OR
Server.TcpSend_0.status = tcpERR_WOULDBLOCK THEN
(* TcpSend not finished -> redo *)
      (* Busy *)
ELSIF Server.TcpSend_0.status = tcpERR_SENTLEN THEN
      (* Log error *)
      FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
      brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
      iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

      END_FOR;

brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
      GetTimeStruct(enable := 1,pDTStructure :=
      ADR(ServerLog[0].Date));
      ServerLog[0].FBK := 'TcpSend_0';
      ServerLog[0].ErrorID := Server.TcpSend_0.status;

      (* Calculate leftover data *)
```

```
            pLeftoverData := Server.TcpSend_0.pData +
            Server.TcpSend_0.sentlen;
            LeftoverDataSize := Server.TcpSend_0.datalen -
            Server.TcpSend_0.sentlen;

            (* Go to a new Send state, where leftover data
            will be sent *)
                  Server.sStep := 35;
      ELSE
            (* Log error *)
            FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
            brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
            iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

            END_FOR;

      brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
            GetTimeStruct(enable := 1,pDTStructure :=
            ADR(ServerLog[0].Date));
            ServerLog[0].FBK := 'TcpSend_0';
            ServerLog[0].ErrorID := Server.TcpSend_0.status;
      END_IF

35:
      (* Sent leftover data *)
      Server.TcpSend_0.enable := 1;
      Server.TcpSend_0.ident := Server.TcpServer_0.identclnt;
      (* Client Ident from AsTCP.TCP_Server *)
      Server.TcpSend_0.pData := pLeftoverData;  (* Which data
      to send *)
      Server.TcpSend_0.datalen := LeftoverDataSize;  (* Lenght
      of data to send *)
      Server.TcpSend_0.flags := 0;
      Server.TcpSend_0;  (* Call the Function*)

      IF gServer.Enable = 0 OR (Server.TcpSend_0.status =
      tcpERR_NOT_CONNECTED) THEN
            (* Log error *)
            FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
            brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
            iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

            END_FOR;

      brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
            GetTimeStruct(enable := 1,pDTStructure :=
            ADR(ServerLog[0].Date));
            ServerLog[0].FBK := 'TcpSend_0';
            ServerLog[0].ErrorID := Server.TcpSend_0.status;

            (* Connection lost or disabled *)
            Server.sStep := 40;
      ELSIF Server.TcpSend_0.status = 0 THEN  (* Data sent *)
            TON_Send_Server.IN := 1;
            TON_Send_Server.PT := gServer.ParaSendTime;
            Server.sStep := 20;
      ELSIF Server.TcpSend_0.status = ERR_FUB_BUSY OR
      Server.TcpSend_0.status = tcpERR_WOULDBLOCK THEN
```

```
(* TcpSend not finished -> redo *)

      (* Busy *)
ELSIF Server.TcpSend_0.status = tcpERR_SENTLEN THEN
      (* Log error *)
      FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
      brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
      iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

      END_FOR;

brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
      GetTimeStruct(enable := 1,pDTStructure :=
      ADR(ServerLog[0].Date));
      ServerLog[0].FBK := 'TcpSend_0';
      ServerLog[0].ErrorID := Server.TcpSend_0.status;

      (* Calculate leftover data *)
      pLeftoverData := Server.TcpSend_0.pData +
      Server.TcpSend_0.sentlen;
      LeftoverDataSize := Server.TcpSend_0.datalen -
      Server.TcpSend_0.sentlen;

      (* Go to a new Send state, where leftover data
      will be sent *)
      Server.sStep := 35;
ELSE
      (* Log error *)
      FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
      brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
      iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

      END_FOR;

brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
      GetTimeStruct(enable := 1,pDTStructure :=
      ADR(ServerLog[0].Date));
      ServerLog[0].FBK := 'TcpSend_0';
      ServerLog[0].ErrorID := Server.TcpSend_0.status;
END_IF

40:
      Server.TcpClose_0.enable := 1;
      Server.TcpClose_0.ident := Server.TcpServer_0.identclnt;
      Server.TcpClose_0.how := 0;; // tcpSHUT_RD OR
      tcpSHUT_WR;
      Server.TcpClose_0;

      IF Server.TcpClose_0.status = 0 THEN
            Server.sStep := 50;
      ELSIF Server.TcpClose_0.status = ERR_FUB_BUSY THEN
      (* TcpClose not finished -> redo *)
            (* Busy *)
      ELSIF Server.TcpClose_0.status = tcpERR_INVALID_IDENT
      THEN;
            (* Log error *)
            FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
```

```
                brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
                iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

                END_FOR;

          brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
                GetTimeStruct(enable := 1,pDTStructure :=
                ADR(ServerLog[0].Date));
                ServerLog[0].FBK := 'TcpClose_0';
                ServerLog[0].ErrorID := Server.TcpClose_0.status;

                (* Close server port *)
                Server.sStep := 50;
          END_IF
    50:
          Server.TcpClose_0.enable := 1;
          Server.TcpClose_0.ident := Server.TcpOpen_0.ident;
          Server.TcpClose_0.how := 0; //tcpSHUT_RD OR tcpSHUT_WR;
          Server.TcpClose_0;

          IF Server.TcpClose_0.status = 0 THEN
                Server.sStep := 0;
          ELSIF Server.TcpClose_0.status = ERR_FUB_BUSY THEN
          (* TcpClose not finished -> redo *)
                (* Busy *)
          ELSE
                (* Log error *)
                FOR iLogEntry := MAX_ENTRIES TO 1 BY -1 DO
                brsmemcpy(ADR(ServerLog[iLogEntry]),ADR(ServerLog[
                iLogEntry-1]),SIZEOF(ServerLog[iLogEntry]));

                END_FOR;

          brsmemset(ADR(ServerLog[0]),0,SIZEOF(ServerLog[0]));
                GetTimeStruct(enable := 1,pDTStructure :=
                ADR(ServerLog[0].Date));
                ServerLog[0].FBK := 'TcpClose_0';
                ServerLog[0].ErrorID := Server.TcpClose_0.status;
          END_IF

END_CASE

    TON_Send_Server;
    TON_RecvTimeout_Server;

END_PROGRAM

PROGRAM _EXIT

    REPEAT
          Server.TcpClose_0.enable := 1;
          Server.TcpClose_0.ident := Server.TcpServer_0.identclnt;
          Server.TcpClose_0.how := 0; //tcpSHUT_RD OR tcpSHUT_WR;
          Server.TcpClose_0;

          UNTIL
          Server.TcpClose_0.status <> ERR_FUB_BUSY
    END_REPEAT;
```

```
      REPEAT
            Server.TcpClose_0.enable := 1;
            Server.TcpClose_0.ident := Server.TcpOpen_0.ident;
            Server.TcpClose_0.how := 0; //tcpSHUT_RD OR tcpSHUT_WR;
            Server.TcpClose_0;

            UNTIL
            Server.TcpClose_0.status <> ERR_FUB_BUSY
      END_REPEAT;

END_PROGRAM
```

### 1.4. *Main Program*

```
PROGRAM _INIT
      // Inicialización de variables
      DatoRecibido := 0;
      DatoCambiado_Vel := 0;
      DatoCambiado_Par := 0;
      NuevoPar := 0;
      NuevaVelocidad := 0;
      Salida3:=FALSE;
      Salida4:=FALSE;
      Salida5:=FALSE;
      Salida6:=FALSE;
      Salida7:=FALSE;
      Salida8:=FALSE;
      AccionTerminada:=TRUE;
      Power := FALSE;
      Paso := 0;

END_PROGRAM

PROGRAM _CYCLIC


//Actualización de datos de par y velocidad

      IF NOT DatoCambiado THEN //cuando llega un nuevo dato la
      variable DatoCambiado estará a 0

            DatoCambiado := 1;

            IF DatoRecibido.14 THEN //para distinguir si es dato de
            par o velocidad

                  DatoRecibido.14:=FALSE; //nos quedamos con el
                  propio valor
                  NuevoPar := MAX_PAR*DatoRecibido/16384;
                  DatoCambiado_Par := 0;

            ELSIF NOT DatoRecibido.14 THEN

                  NuevaVelocidad:= MAX_VEL/16384*DatoRecibido*100/6;
                  DatoCambiado_Vel := 0;
```

```
            END_IF

      END_IF


//  Habilitación de servomotores 1 y 2

      IF AccionRecibida = 1 THEN //cuando la acción recibida sea un
      1, se encenderá la salida 3 que corresponde al motor 1

            Salida3:=TRUE;
            AccionTerminada:=TRUE; //una vez encendemos la salida,
            la acción ha terminado

      ELSIF AccionRecibida = 2 THEN //cuando la acción recibida sea
      un 2, se encenderá la salida 4 que corresponde al motor 2

            Salida4:=TRUE;
            AccionTerminada:=TRUE;

      END_IF


// Deshabilitación de servomotores 1 y 2

      IF AccionRecibida = 3 THEN //cuando la acción recibida sea un
      3, se apagará la variable salida 3 que corresponde al motor 1

            Salida3:=FALSE;
            AccionTerminada:=TRUE;

      ELSIF AccionRecibida = 4 THEN //cuando la acción recibida sea
      un 4, se apagará la variable salid a4 que corresponde al
      motor 2

            Salida4:=FALSE;
            AccionTerminada:=TRUE;

      END_IF

// Habilitación y deshabilitación de las salidas en función del
valor dado por el usuario

      IF (Power = TRUE) THEN // Si el comando de POWER está
      encendido, se encender· la salida 5

            Salida5:=TRUE;


      ELSIF (Power = FALSE) THEN // Si el comando de POWER está
      apagado, se apagará la salida 5

            Salida5:=FALSE;

      END_IF
```

```
      IF (Paso = 2) THEN // Cuando se encuentre en el paso de HOME
      se encenderá la salida 6

            Salida6 := TRUE;

      ELSIF NOT (Paso = 2) THEN // Una vez ejecutado el estado HOME
      se apagará la salida 6

            Salida6 := FALSE;

      END_IF

      IF AccionRecibida = 12 THEN //cuando la acción recibida sea
      un 12, se encenderá la variable salida5

            Salida7:=TRUE;

      ELSIF AccionRecibida = 11 THEN //cuando la acción recibida
      sea un 11, se apagará la variable salida5

            Salida7:=FALSE;

      END_IF

      IF (Paso = 100) THEN // Si se encuentra en el estado de
      error, se encenderá la salida 8

            Salida8 := TRUE;

      ELSIF NOT (Paso = 100) THEN

            Salida8 := FALSE;

      END_IF



// Damos el valor convertido de la velocidad y par actuales para
mandar al cliente

      VelocidadReal := TRUNC(ActualVelocity*CONST_VEL);



END_PROGRAM

PROGRAM _EXIT
      (* Insert code here *)

END_PROGRAM
```