Diseño, desarrollo e implementación de un sistema de control para el *servo driver* "R7D-BP04H" y servomotor "R88M-G10030H", mediante un circuito de acoplamiento y señal de modulación por ancho de pulso.

Julio Emanuel Lopez Ozaeta

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño, desarrollo e implementación de un sistema de control para el *servo driver* "R7D-BP04H" y servomotor "R88M-G10030H", mediante un circuito de acoplamiento y señal de modulación por ancho de pulso.

Trabajo de graduación presentado por Julio Emanuel Lopez Ozaeta
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2024

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

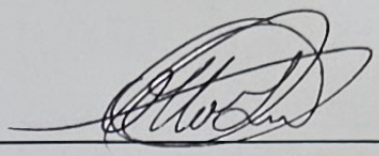
Diseño, desarrollo e implementación de un sistema de control para el *servo driver* "R7D-BP04H" y servomotor "R88M-G10030H", mediante un circuito de acoplamiento y señal de modulación por ancho de pulso.

Trabajo de graduación presentado por Julio Emanuel Lopez Ozaeta
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2024

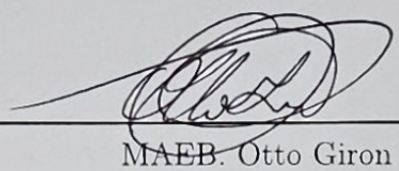
Vo.Bo.:



(f) _____

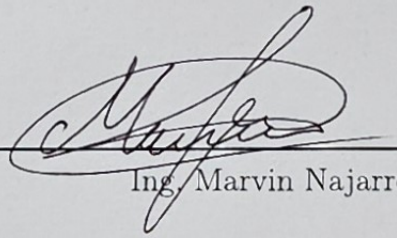
MAEB. Otto Giron

Tribunal Examinador:



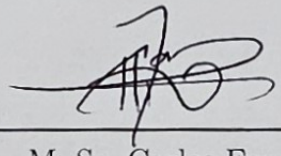
(f) _____

MAEB. Otto Giron



(f) _____

Ing. Marvin Najarro



(f) _____

M. Sc. Carlos Esquit

Fecha de aprobación: Guatemala, 19 de junio de 2024.

Prefacio

Quiero expresar mi más sincero agradecimiento a la Universidad del Valle por las diversas oportunidades de crecimiento personal y profesional que me ha proporcionado, así como a la amplia gama de profesores que me han guiado hasta este punto. Agradezco profundamente a mi familia, en especial a la señora Esperanza Díaz y a la Licenciada Marissa Ozaeta, así como a la familia De León Díaz y a la familia Mancio Ruballos por su constante apoyo y aliento. También deseo expresar mi gratitud al MSc. Carlos Esquit, al Ing. Otto Girón y al Ing. Marvin Nájaro por su valiosa orientación y asesoramiento a lo largo de este proyecto. Sus contribuciones han sido fundamentales para el éxito de esta investigación.

Índice

Lista de figuras

Lista de cuadros

Resumen

El Departamento de Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala cuenta con el servo controlador R7D-BP04H y los servomotores R88M-G10030H en su inventario. Estos dispositivos forman parte de la extensa gama de controladores de alta precisión y están destinados a procesos industriales de gran precisión.

El propósito primordial de este proyecto consistió en la adaptación de dicho equipamiento para su implementación en el Laboratorio de Automatización e Instrumentación Industrial, utilizando exclusivamente los recursos disponibles en la Universidad del Valle de Guatemala. El objetivo es proporcionar a los estudiantes una formación más completa mediante el uso de equipos que se encuentran comúnmente en el ámbito profesional.

Es relevante destacar que el servo driver cuenta con 8 puertos de accionamiento para su funcionamiento completo. Además, el servo controlador utiliza dos señales PWM, las cuales deben estar desfasadas en 90 grados para indicar la dirección de giro del motor. El servo controlador descompone esta señal PWM en dos componentes: la frecuencia, que determina la velocidad de giro, y la cantidad de pulsos, que indican la posición angular del motor.

Con estas consideraciones en mente, se optó por utilizar un microcontrolador para gestionar las entradas del servo controlador, generar las señales PWM y establecer una comunicación con el PLC mediante el protocolo de comunicación PROFIBUS DP.

Abstract

The Department of Electronics, Mechatronics, and Biomedical Engineering at the University of Valle de Guatemala possesses the R7D-BP04H servo controller and R88M-G10030H servomotors in its inventory. These devices are part of the extensive range of high-precision controllers and are intended for high-precision industrial processes.

The primary purpose of this project was to adapt this equipment for use in the Industrial Automation and Instrumentation Laboratory, utilizing exclusively the resources available at the University of Valle de Guatemala. The aim is to provide students with a more comprehensive education by using equipment commonly found in the professional field.

It is relevant to note that the servo driver has 8 drive ports for its full operation. Additionally, the servo controller uses two PWM signals, which must be phased at 90 degrees to indicate the motor's direction of rotation. The servo controller decomposes this PWM signal into two components: frequency, which determines the rotation speed, and pulse quantity, which indicates the motor's angular position.

With these considerations in mind, a microcontroller was chosen to manage the servo controller's inputs, generate PWM signals, and establish communication with the PLC via PROFIBUS DP.

CAPÍTULO 1

Introducción

Este trabajo se centra en la adaptación del servo controlador R7D-BP04H y el servomotor R88M-G10030H en el entorno del laboratorio de automatización e instrumentación industrial.

El servo controlador requiere 8 entradas de 24V y dos señales de PWM desfasadas en 90 grados para indicar la dirección de giro. Para controlar la velocidad, el servo controlador utiliza la frecuencia de la señal PWM, mientras que la cantidad de pulsos denota la posición angular del motor. Dado que los PLC del laboratorio no pueden generar estas señales PWM y la disponibilidad de puertos de entrada y salida es limitada, con tan solo 12 puertos, de los cuales 4 ya están ocupados por botones y actuadores del mismo panel, se decidió utilizar un microcontrolador Arduino UNO con relés de estado sólido para el accionamiento de las entradas del servo controlador, dado que estas operan a 24V. Asimismo, se empleó para la generación de las señales PWM y la comunicación PROFIBUS DP.

Durante este proceso, se enfrentó a ciertas limitaciones, entre las cuales se incluyó la necesidad de minimizar la invasión en el equipo existente para evitar daños. Esto implicó diseñar un equipo que fuera fácilmente montable y desmontable, requiriendo cuidadosa consideración y planificación en cada etapa del proceso.

El resultado de este trabajo se reflejó en la exitosa instalación de un circuito de acople poco invasivo, así como en su integración efectiva como parte integral de la planificación de las clases pertinentes para el uso de este equipo. Este logro representa un hito significativo en el avance de este proyecto.

CAPÍTULO 2

Antecedentes

Debido a la naturaleza de este trabajo de graduación, carecemos de investigaciones previas, resultando en una escasa documentación para la conducción de este trabajo. No obstante, gracias a una exhaustiva investigación respaldada por la revista científica Science Direct, específicamente el estudio titulado "Siemens S7-1200 PLC DC Motor Control Capabilities", hemos logrado adquirir una comprensión más profunda sobre los aspectos operativos de los motores en la industria. Este estudio también aborda los requisitos de seguridad tanto para los operarios como para el equipo, lo que nos ha permitido comprender mejor cómo llevar a cabo este proyecto de manera segura. [1]

Adicionalmente, este trabajo se adentra en las Aplicaciones Prácticas en el Control de Servomotores. La integración de sistemas embebidos, haciendo referencia a la amalgama de microcontroladores para la ejecución de funciones específicas, es de crucial importancia. En esta investigación, fusionamos sistemas industriales con sistemas de control para la gestión del servo controlador y el servomotor. Las considerables ventajas de estos sistemas, dirigidas al ámbito industrial, son evidentes en el artículo científico "Design and Research of Embedded PLC Development System", publicado en la IEEE por Dong Yulin[2]. Por otro lado, la investigación de Y.Kuang sobre la comunicación entre PLC y Arduino basada en protocolo Modbus es un elemento esencial para la materialización de este proyecto. [3].

Finalmente, el empleo de los manuales de instalación de Omron, junto con el respaldo adicional proveniente de foros especializados, facilitó una amplia comprensión en busca de una solución viable para este sistema de control. El recurso a estos documentos y foros desempeñó un papel trascendental en la interpretación de este trabajo.

CAPÍTULO 3

Justificación

La implementación de servomotores se erige como un elemento crucial en la industria moderna, desempeñando un papel fundamental al proporcionar un control preciso y eficiente de la posición, velocidad y torque en diversas aplicaciones. Estos dispositivos, esenciales en maquinaria industrial, robótica y sistemas de automatización, destacan por su capacidad para convertir señales de control en movimientos precisos. Su alta precisión y respuesta rápida no solo optimizan la eficiencia operativa y mejoran la calidad del producto, sino que también reducen el desperdicio en los procesos de fabricación. La versatilidad de los servo motores los convierte en herramientas ideales para aplicaciones que van desde la producción de bienes de consumo hasta la fabricación de equipos médicos y aeroespaciales. En resumen, estos dispositivos se erigen como catalizadores clave para la mejora de la productividad y la calidad en la industria, impulsando la innovación y la competitividad en un entorno empresarial cada vez más exigente.

Este proyecto de graduación encarna el lema de la Universidad del Valle, 'Excelencia que Trasciende', al centrarse en la implementación del servo controlador R7D-BP04H y el Servomotor R88M-G10030H. Estos recursos, ya disponibles en el departamento de Electrónica, Mecatrónica y Biomédica, se instalarán y utilizarán en el laboratorio de instrumentación y automatización industrial, proporcionando una experiencia educativa valiosa para los futuros estudiantes de este curso. Este trabajo no solo representa un logro individual, sino que también encapsula la amalgama de conocimientos y destrezas cultivadas a través de diversos cursos fundamentales del Departamento. Desde Circuitos Analógicos I y II, Circuitos Electrónicos I y II, Programación de Microcontroladores, Electrónica Digital I y II, Instalaciones Eléctricas, Máquinas Eléctricas, hasta Instrumentación y Automatización Industrial I y II, cada curso ha contribuido a la formación integral necesaria para llevar a cabo este trabajo de graduación.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Diseñar, desarrollar e implementar un sistema de control para el servomotor 'R88M-G10030H' mediante la creación de un circuito de acoplamiento entre un controlador, el servo controlador 'R7D-BP04H' y el PLC S7-300, con fines educativos para futuros estudiantes en la Universidad del Valle de Guatemala.

4.2. Objetivos específicos

- Instalar de manera segura, tanto para el usuario como para el equipo del laboratorio de automatización, del servo controlador R7D-BP04H y el servomotor R88M-G10030H.
- Establecer una comunicación entre el controlador y el PLC S7-300.
- Investigar y diseñar la señal de modulación por ancho de pulso que utiliza el servo controlador R7D-BP04H para el funcionamiento del servomotor R88M-G10030H.
- Investigar, diseñar y desarrollar un circuito de acople entre el servo controlador R7D-BP04H, el PLC S7-300 y un microcontrolador con el fin de orquestar la comunicación y accionamiento para un funcionamiento correcto.
- Implementar el uso de dos modos de movimiento del servo controlador R7D-BP04H cuyos son: *Internal velocity control* y *High Function Position control*.
- Desarrollar un programa para el controlador, que permita la interacción entre el servo controlador R7D-BP04H y el PLC S7-300.
- Desarrollar un manual de usuario para el funcionamiento de modos y uso de equipo.

CAPÍTULO 5

Alcance

El alcance de este proyecto se enfoca en la implementación del servo controlador (R7D-BP04H) y el servomotor (R88M-G10030H) en el laboratorio de automatización e instrumentación industrial. Se propone diseñar un circuito de acople que pueda integrarse con los tableros y programas existentes en dicho laboratorio. Con este objetivo, se ha seleccionado el protocolo de comunicación estándar conocido como PROFIBUS DP, aprovechando que todos los PLCs del laboratorio cuentan con un puerto compatible con este protocolo. Se busca una adaptación no invasiva a los módulos de entrada/salida (E/S) existentes, dada la limitación de puertos disponibles. Además, se persigue un funcionamiento flexible que permita una demostración educativa efectiva.

El circuito de acople debe encargarse de gestionar las entradas del servo controlador, la generación de las señales PWM y la comunicación mediante PROFIBUS DP. Para lograrlo, se ha optado por utilizar el microcontrolador Arduino Uno como parte del circuito de acople, debido a su capacidad para manejar estas tareas. Se ha incorporado relés de estado solido para adaptar los voltajes requeridos para accionar el servo controlador y aprovechar la potencia del Arduino. Además, se ha incluido el Módulo XW2B-34G5 para la separación de los 24 puertos de entrada y salida del servo controlador.

El programa desarrollado para el microcontrolador lleva a cabo la recepción, descomposición y, finalmente, la interpretación de las distintas tramas del protocolo PROFIBUS DP, así como la gestión de los puertos del módulo de relé de ocho canales y la generación de las señales PWM con dos frecuencias distintas.

CAPÍTULO 6

Marco teórico

Servo controlador R7D-BP04H

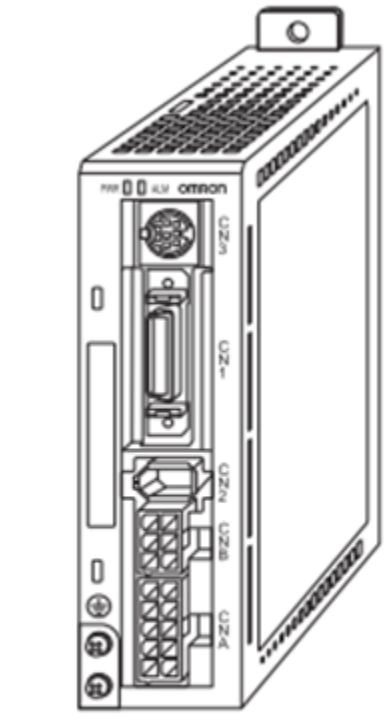


Figura 1: Flujo de diseño de un circuito integrado.

La serie Smart Step 2 (SS2) fabricado por la empresa Omron, es una familia de servos diseñada para el control de posición en aplicaciones de pulsos. El SS2 ofrece características como la función de Autotuning en tiempo real y el filtro adaptativo para calcular la inercia del sistema y ajustar las ganancias de manera automática. Además, cuenta con un filtro Notch para suprimir las vibraciones en la máquina, entre otras funciones. Las características clave del SS2 incluyen múltiples tipos de entradas de pulsos, configuración flexible, dos modos de control de posición, motores con "encoder" incremental de 2500 pulsos/rotación y control de velocidad con cuatro velocidades internas. También ofrece un posicionamiento de alta velocidad y la función Damping Control para la supresión de vibraciones mecánicas. En general la serie Smart Step 2 (SS2) es una gama de servos diseñada para el control de posición en aplicaciones de pulsos, con características como Autotuning en tiempo real, filtro adaptativo y funciones para suprimir vibraciones y mejorar el control de posición y velocidad en máquinas.[4]

6.1. Especificaciones

A continuación, se muestran las especificaciones generalas del servo controlador R7D-BP04H en los cuadros 1 y2.

Elementos	Especificación
Temperatura ambiente de funcionamiento	0 a 55°C
Humedad ambiente de funcionamiento	90 % máx. (sin condensación)
Temperatura ambiente de almacenamiento	–20 a 65°C
Humedad ambiente de almacenamiento	90 % máx. (sin condensación)
Atmósfera de almacenamiento/operación	Sin gases corrosivos.
Resistencia a vibraciones	10 a 60 Hz; aceleración: 5,9 m/s^2 (0,6 G) máx
Resistencia a golpes	Aceleración máx. 19,6 m/s^2, 3 veces en las direcciones X, Y y Z
Resistencia de aislamiento	Entre terminales de alimentación/línea de alimentación y marco de tierra: 0,5 M mín. (a 500 Vc.c.)
Rigidez dieléctrica	Entre fuente de alimentación/terminales de alimentación y marco de tierra: 1.500 Vc.a. a 50/60 Hz durante 1 minuto Entre cada señal de control y el marco de tierra: 500 Vc.a. durante 1 minuto
Grado de protección	Incorporado en el panel (IP10).
Normas internacionales	Aprobación CE: EMC EN55011 clase A grupo 1, EN 61000-6-2, baja tensión EN50178

Cuadro 1: Especificaciones generales

Elementos	Especificación (R7D-BP04H)
Corriente de salida permanente (eficaz)	2,5 A
Corriente de salida máxima instantánea (eficaz)	7,8 A
Fuente de alimentación del circuito principal	Monofásico de 200 a 240 Vc.a. (170 a 264 V), 50/60 Hz
Método de control	Servo totalmente digital
Realimentación	10.000 pulsos/revolución, 'encode' incremental
Método de variador	Método PWM basado en IGBT
Frecuencia de PWM	6 kHz
Peso	0,42 kg
Tensión del motor compatible	200
Respuesta de comando de pulsos	Line driver: 500 kpps
Capacidad del motor compatible	400 W
Servomotor aplicable (R88M-)	G40030H,GP40030H

Cuadro 2: Especificaciones de rendimiento

Diagrama de instalación eléctrica

A continuación en la Figura 2 podemos observar el diagrama de instalación eléctrica. Dicho diagrama utiliza voltajes 220Vac como fuente principal de alimentación del servo controlador. Adicionalmente podemos ver el uso de 24V DC para el puerto de entrada/salida CN1.

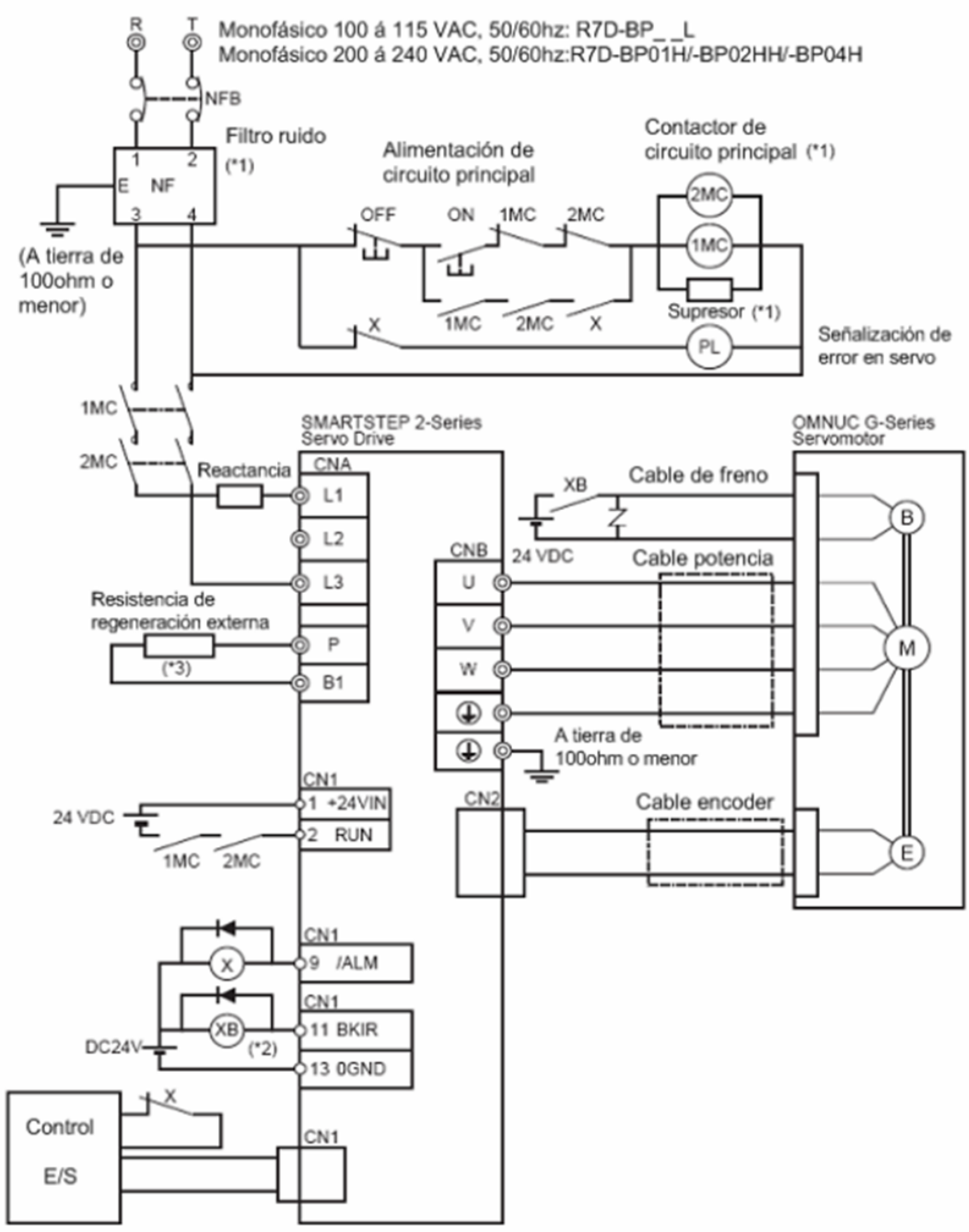


Figura 2: Diagrama de conexiones eléctricas.

6.2. Puertos

Designación de interfaces del controlador R7D-BP04H

La Figura 3 identifica los puertos que se encuentran en la parte frontal del servo controlador.

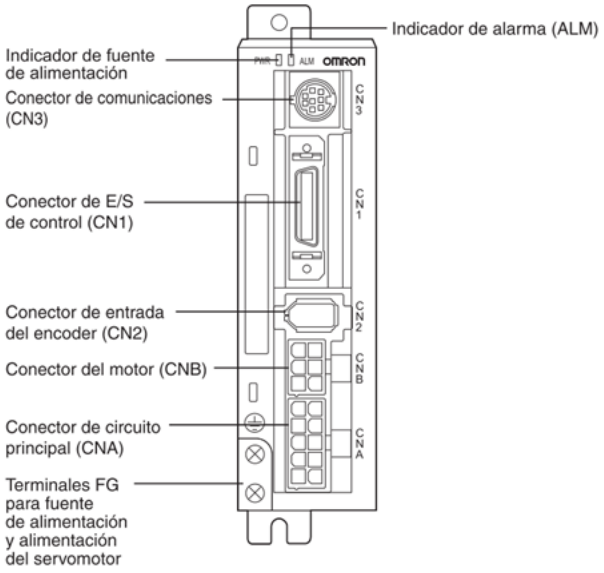


Figura 3: Nombre de puertos del servo controlador

Terminal CNA

El puerto CNA, representado en la Figura 4, es la fuente de alimentación principal del servo controlador. Utiliza un conector molex de 10 pines para su conexión. El Cuadro 3 detalla la configuración de conexión de cada pin para una instalación correcta. Debido al modelo que se posee se usan solo el pin 6 y el pin 10.

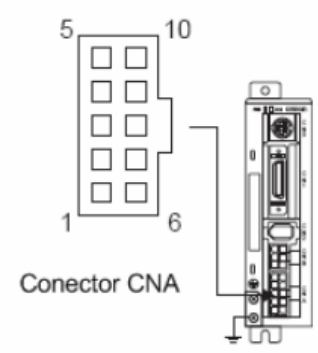


Figura 4: Terminales y conector de alimentación del driver (CNA)

Símbolo	Terminal	Descripción	Función
L1	10	Terminales circuitos de potencia	Monofásico 200-230VAC (170-160 VAC) 50/60 HZ
L2	8		
L3	6		
P	5	Terminales de conexión de resistencia de regeneración externa	Conectar una resistencia si la energía regenerada es excesiva.
B1	3		
FG	1	Conexión a tierra	Terminal de tierra

Cuadro 3: Terminales y conector de alimentación del driver (CNA)

Terminal CNB

El puerto CNB, representado en la Figura 5, es la fuente de alimentación del servomotor. Utiliza un conector molex de 6 pines para su conexión. El Cuadro 4 detalla la configuración de conexión de cada pin para una instalación correcta.

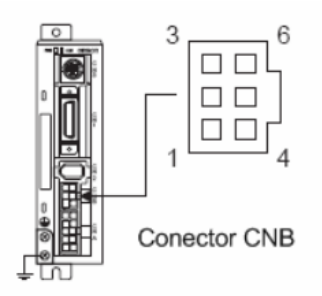


Figura 5: Terminales y conector de motor (CNB)

Símbolo	Terminal	Nombre
U	1(Rojo)	Terminales de motor
V	4(Blanco)	
W	6(Azul)	
GND	3(Verde/Amarillo)	Conexión a tierra

Cuadro 4: Terminales y conector de motor (CNB)

Terminal CN1

En la figura 6 se muestran los diversos circuitos ubicados detrás del puerto CN1, con el fin de proporcionar una comprensión más clara tanto de los voltajes como de los distintos dispositivos necesarios para el accionamiento de estos pines. Las descripciones detalladas de los diversos pines se encuentran en las figuras 5,6 7.

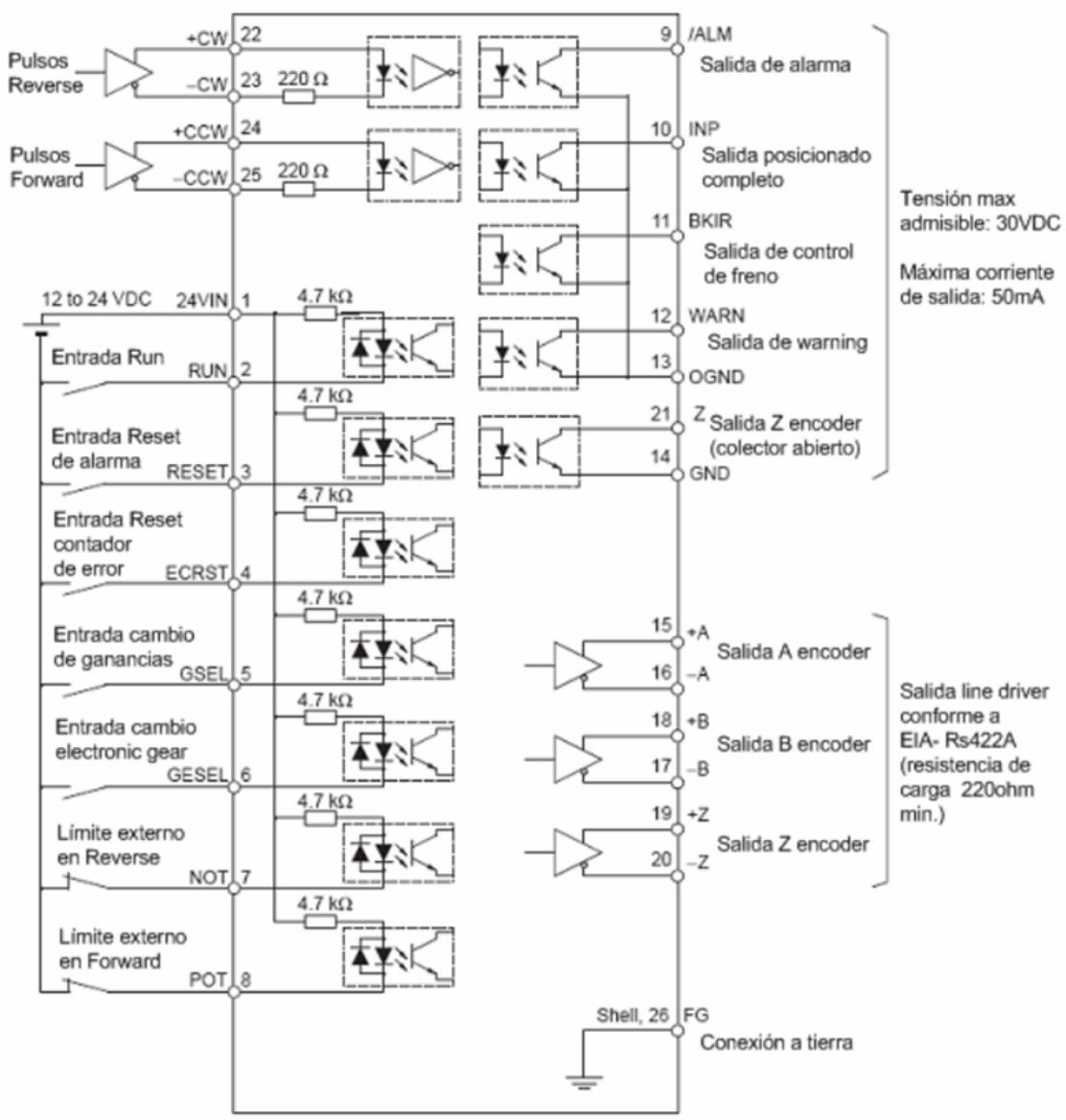


Figura 6: Conexionado de señales de control de E/S

No. Pin	Nombre de la señal	Descripción	Función/Interfaz
1	+24VIN	Entrada de fuente de alimentación CC para control	Terminal de entrada de alimentación (12 a 24 VCC) para entrada de secuencia (pin 1).
2	RUN	Entrada de Comando "RUN"	ENCENDIDO: Activa el servomotor (Suministra energía al servomotor).
3	RESET	Entrada de Reinicio de Alarma	EN: El estado de alarma del servomotor se restablece. Debe estar ENCENDIDO durante un mínimo de 120 ms
4	ECRST/ VSEL2	Entrada de reinicio del contador de desviación o entrada de selección de velocidad establecida internamente 2.	Entrada de Restablecimiento de Contador de Desviación en Modo de Control de Posición (cuando Pn02 está configurado en 0 o 2). ENCENDIDO: Se prohíben comandos de pulso y se borra el contador de desviación. Debe estar ENCENDIDO durante al menos 2 ms.
			Selección de velocidad interna establecida en 2 en el Modo de Control de Velocidad Interna (cuando Pn02 está configurado en 1). ENCENDIDO: Entrada de Selección de Velocidad Interna Establecida en 2.
5	GSEL/ VZERO/ TLSEL	Entrada de Interruptor de Ganancia / Entrada de Designación de Velocidad Cero / Entrada de Interruptor de Límite de Par.	Entrada del interruptor de ganancia en modo de control de posición (cuando Pn02 está configurado en 0 o 2) cuando el interruptor de designación de velocidad cero/limitación de par (Pn06) está configurado en 0 o 1.
			Entrada de designación de velocidad cero en el Modo de Control de Velocidad Interna (cuando Pn02 está configurado en 1). APAGADO: El comando de velocidad es cero. La entrada también puede ser desactivada mediante la configuración del Interruptor de Designación de Velocidad Cero/Límite de Par (Pn06): Habilitado: Pn06 = 1, Deshabilitado: Pn06 = 0.
			Selección del límite de par en el Modo de Control de Posición y en el Modo de Control de Velocidad Interna cuando el interruptor de Designación de Velocidad Cero/Límite de Par (Pn06) está configurado en 2. APAGADO: Límite de par 1 habilitado. (Pn70, 5E, 63) ENCENDIDO: Límite de par 2 habilitado. (Pn71, 72, 73)

Cuadro 5: Control de entradas del puerto CN1

No. Pin	Nombre de la señal	Descripción	Función/Interfaz
6	GESEL/ VSEL1	Entrada de Cambio de Velocidad Electrónico / Selección de Velocidad Establecida Internamente 1.	Entrada de Cambio de Relación Electrónica en Modo de Control de Posición (cuando Pn02 está configurado en 0 o 2). APAGADO: Numerador de Relación Electrónica 1 (Pn46) ENCENDIDO: Numerador de Relación Electrónica 2 (Pn47)
			Selección de velocidad interna configurada en 1 en el Modo de Control de Velocidad Interna (cuando Pn02 está configurado en 1). ENCENDIDO: Se ha introducido la selección de velocidad interna configurada en 1.
7	NOT	Prohibición de Retroceso de Entrada	Entrada de sobrecarrera de rotación inversa. APAGADO: Prohibido, ENCENDIDO: Permitido.
8	POT	Entrada de prohibición de avance	Entrada de sobrecarrera de rotación adelante. APAGADO: Prohibido, ENCENDIDO: Permitido.
22	+CW/PULS/FA	Entrada de Pulsos Inversos, Entrada de Pulsos de Avance o Pulsos con Diferencia de Fase de 90 grados (Fase A)	Terminales de entrada para pulsos de comando de posición. Entrada de línea: Frecuencia máxima de respuesta: 500 kpps Entrada de colector abierto: Frecuencia máxima de respuesta: 200 kpps Cualquiera de las siguientes opciones puede ser seleccionada utilizando la configuración Pn42: pulsos de avance y retroceso (CW/CCW); pulso de avance y señal de dirección (PULS/SIGN); señales con diferencia de fase de 90 grados (fase A/B) (FA/FB).
23	-CW/PULS/FA		
24	+CCW/SIGN/FB	Pulsos hacia adelante, señal de dirección o pulsos con una diferencia de fase de 90 grados (Fase B).	
25	-CCW/ SIGN/FB		

Cuadro 6: Control de entradas del puerto CN1

No. Pin	Nombre de la señal	Descripción	Función/Interfaz
9	/ALM	Salida de alarma	Cuando el Servo Drive genera una alarma, la salida se apaga.
10	INP/TGON	Salida de posicionamiento completado o salida de detección de velocidad de rotación del servomotor	Salida de posicionamiento completado en modo de control de posición (cuando Pn02 está configurado en 0 o 2). ENCENDIDO: Los pulsos residuales para el contador de desviación están dentro de la configuración del Rango de finalización de posicionamiento (Pn60).
			Salida de detección de rotación del motor en modo de control de velocidad interno (cuando Pn02 está configurado en 1). ON: El número de rotaciones del servomotor excede el valor establecido para Velocidad de detección de rotación del servomotor (Pn62).
11	BKIR	Salida de bloqueo de freno	Emite las señales de sincronización del freno de parada. Suelte el freno de parada cuando esta señal esté activada.
12	WARN	Salida de advertencia	La señal seleccionada en la Selección de salida de advertencia (Pn09) se emite
13	OGND	Tierra de salida común	Tierra común para salidas de secuencia (pines 9, 10, 11 y 12).
14	GND	Comando de GND	Común para salida de codificador y salida de fase Z (pin 21).
15	+A	Salida de fase A del Encoder	Estas señales emiten pulsos del codificador de acuerdo con la configuración de relación de división del codificador (Pn44). Esta es la salida del controlador de línea (equivalente a RS-422).
16	- A		
17	-B	Salida de fase B del Encoder	
18	+B		
19	+Z	Salida de fase Z del Encoder	
20	-Z		
21	Z	Salida de fase Z	Emite la fase Z para el codificador (1 pulso/rotación). Esta es la salida del colector abierto.

Cuadro 7: Control de salidas del puerto CN1

Conexiones

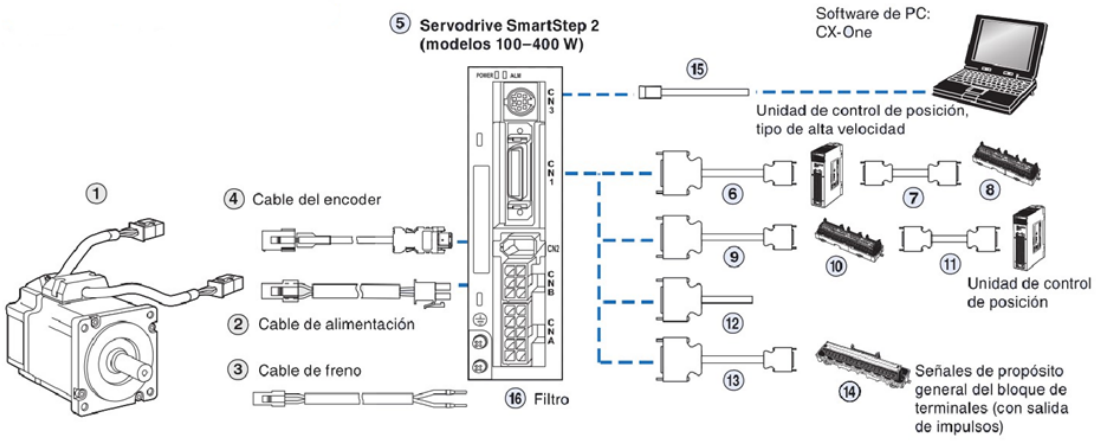


Figura 7: Diagrama de configuración de equipo general

Símbolos	Especificaciones			
1	Servomotores compatibles	Modelos	Tipo cilíndrico	Tipo plano
			R88M-GP10030H- R88M-GP20030H- R88M-GP40030H-	R88M-G05030H- R88M-G10030H- R88M-G20030H- R88M-G40030H-
2	Cable de Alimentación	Modelos	R7A-CAB003S	
			R7A-CAB005S	
			R7A-CAB010S	
			R7A-CAB015S	
			R7A-CAB020S	
3	Cable de Freno	Modelos		
4	Cable de Encoder	Modelos		
5	R7D-BP04H			
6	Cable de control (salida de line driver para 1 eje)	Unidad de control de posición (tipo de alta velocidad) CJ1W-NC234 CJ1W-NC434		XW2Z-100J-G12
				XW2Z-500J-G12
				XW2Z-10MJ-G1
	Cable de control (salida de colector abierto para 1 eje)	Unidad de control de posición (tipo de alta velocidad) CJ1W-NC214 CJ1W-NC414		XW2Z-100J-G16
				XW2Z-300J-G16
	Cable de control (salida de line driver para 2 ejes)	Unidad de control de posición (tipo de alta velocidad) CJ1W-NC234 CJ1W-NC434		XW2Z-100J-G4 XW2Z-500J-G4 XW2Z-10MJ-G4
	Cable de control (salida de colector abierto para 2 ejes)	Unidad de control de posición (tipo de alta velocidad) CJ1W-NC214 CJ1W-NC414		XW2Z-100J-G8 XW2Z-300J-G8
7	Cable del bloque de terminales para señales externas (para común de entradas, entradas de marcha directa/inversa prohibidas, entrada de parada de emergencia, entrada de proximidad de origen y entrada de interrupción)		Unidades de control de posición (tipo de alta velocidad) CJ1W-NC234 CJ1W-NC434 CJ1W-NC214 CJ1W-NC414	XW2Z-C50X
				XW2Z-100X
				XW2Z-200X
8	Bloque de terminales para señales externas (con tornillo M3 y para terminales de pines)			XW2Z-300X
				XW2Z-500X
	Bloque de terminales para señales externas (con tornillo M3.5 y para terminales tipo horquilla/ redondos)			XW2Z-010X
				XW2B-20G4
	Bloque de terminales para señales externas (con tornillo M3 y para terminales de pin tipo horquilla/redondos)			XW2B-20G5
				XW2B-20G6
9	Cable desde bloque de terminales hasta servodrive		Modelo	XW2Z-100J-B29
				XW2Z-200J-B29
10	Interfaz pasiva	Unidad de control de posición CS1W-NC1_3, CJ1W-NC1_3 o C200HW-NC113		XW2B-20J6-1B (1 eje)
		Unidad de control de posición CS1W-NC2_3/4_3, CJ1W-NC2_3/4_3 o C200HW-NC213/413		XW2B-40J6-2B (2 ejes)
		CQM1H-PLB21 o CQM1-CPU43-V1		XW2B-20J6-3B (1 eje)

Cuadro 8: Inter-conexiones para el servo motor R88M-G10030H/T y el controlador R7D-BP04H

Símbolos	Especificaciones			
11	Cable de conexión de Unidad de control de posición	CJ1W-NC133		XW2Z-050J-A18 XW2Z-100J-A18
		CJ1W-NC233/433		XW2Z-050J-A19 XW2Z-100J-A19
		CS1W-NC133		XW2Z-050J-A10 XW2Z-100J-A10
		CS1W-NC233/433		XW2Z-050J-A11 XW2Z-100J-A11
		CJ1W-NC113		XW2Z-050J-A14 XW2Z-100J-A14
		CJ1W-NC213/413		XW2Z-050J-A15 XW2Z-100J-A15
		CS1W-NC113 C200HW-NC113		XW2Z-050J-A6 XW2Z-100J-A6
		CS1W-NC213/413 C200HW-NC213/413		XW2Z-050J-A7 XW2Z-100J-A7
		CJ1M-CPU21/22/23		XW2Z-050J-A33 XW2Z-100J-A33
		CQM1H-PLB21 CQM1-CPU43-V1		XW2Z-050J-A3 XW2Z-100J-A3
12	Cable de empleo general	Para controladores de empleo general		R7A-CPB001S
				R7A-CPB002S
13	Cable del bloque de terminales		Para controladores de empleo general	XW2Z-100J-B28
				XW2Z-200J-B28
14	Bloque de terminales (con tornillo M3 y para terminales de pines)			XW2B-34G4
	Bloque de terminales (con tornillo M3.5 y para terminales tipo horquilla/redondos)			XW2B-34G5
	Bloque de terminales (con tornillo M3 y para terminales tipo horquilla/redondos)			XW2B-34G6
15	Cable de monitor de ordenador personal			R88A-CCG002P2
16	R7D-BP01H/02HH/04H			R7A-FIB104-RE

Cuadro 9: Inter-conexiones para el servo motor R88M-G10030H/T y el controlador R7D-BP04H 2

6.3. Funciones de operación

1. Alto control de posición (*High Function Position control*)

La posición se puede realizar en función de los pulsos de entrada en los puertos CW y CCW (CN1-22 a 25). El servomotor gira utilizando el valor de los pulsos de entrada de cadena multiplicado por el valor del engranaje electrónico (Pn46, Pn47, Pn4A y Pn4B). Los Servo Drives de la serie SMARTSTEP2 tienen dos modos de control de posición: control de posición de alta respuesta y control de posición avanzado.[5]

2. Control de velocidad establecido internamente (*Internal velocity control*)

La velocidad del servomotor se puede controlar utilizando las velocidades configuradas en los parámetros de ajuste de velocidad interna del número 1 al 4. Después de que la entrada EJECUCIÓN (RUN) y, consecutivamente, la entrada de designación de velocidad cero (VZERO) sean activadas, el servomotor acelerará de acuerdo con el tiempo de aceleración de arranque suave (Pn58).

Cuando la entrada de designación de velocidad cero (VZERO) sea desactivada, el servomotor desacelerará hasta detenerse de acuerdo con el tiempo de desaceleración de arranque suave (Pn59). El cambio entre las velocidades configuradas internamente se controla mediante las entradas de Selección de Velocidad Configurada Internamente 1 y 2 (VSEL1: CN1-6, VSEL2: CN1-4).[5]

Servo motor R88M-G20030H

Figura 8: Servo motor R88M-G20030H marca Omron

Tensiones aplicadas	Magnitud	Valor
Salida nominal	W	200
Par nominal	Nm	0.64
Par máximo instantáneo	Nm	1.78
Corriente nominal	A	1.6
Corriente máx. instantánea	A	4.9
Velocidad nominal	mín.–1	3
Velocidad máx.	rpm	3
Momento de inercia del rotor	kg·m2x10–4	0.14
Momento de inercia de la carga admisible	JL	30
Relación de potencia nominal	kW/s	30.3

Cuadro 10: Especificaciones de servomotor R88M-G20030H cilíndricos

Dimensiones

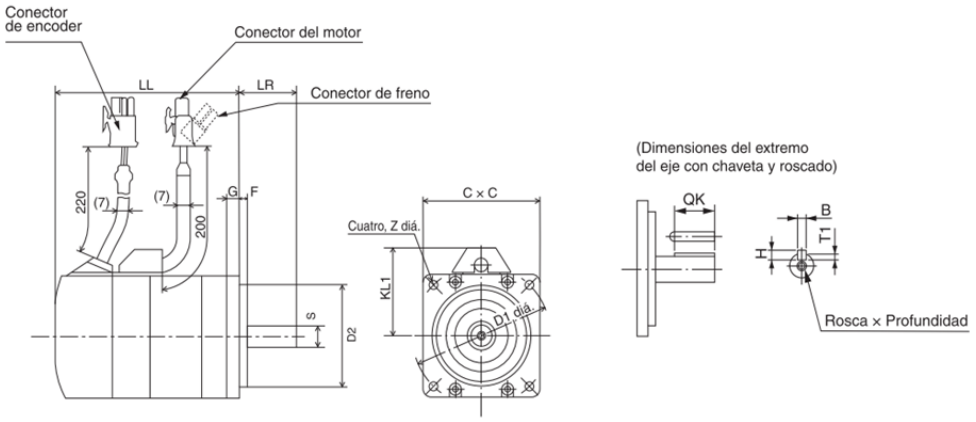


Figura 9: Diagrama de dimensiones de servo motor R88M-20030H

R88M-20030H	LR	KL1	Superficie de Brida						Extremo del eje						Peso aprox (kg)
			D1	D2	C	F	G	Z	S	QK	B	H	T1		
Dimensiones(mm)	30	53	90	70	80	5	8	5.5	11	18	4	4	2.5	M4 X8L	1.3

Cuadro 11: Tabla de dimensiones

Conexiones

Existen dos conexiones entre el servomotor y el controlador: alimentación del motor y encoder. A continuación, se detalla los cables que hacen esto posible:

Cable de alimentación del servo

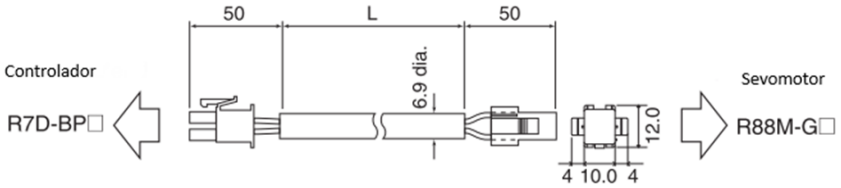


Figura 10: Cable de alimentación del servo

Controlador de servomotor		Color de cable	Servomotor	
No. Puerto	Señal		No. Puerto	Señal
1	Fase-U	Rojo	1	Fase-U
4	Fase-V	Blanco	2	Fase-V
6	Fase-W	Azul	3	Fase-W
3	FG	Verde/Amarillo	4	FG

Cuadro 12: Cableado de conectores del puerto de alimentación

Cable de comunicación del encoder

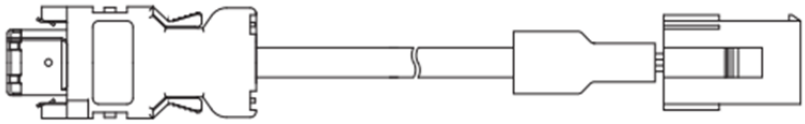


Figura 11: Diagrama de cable de comunicación del encoder

El contenido las secciones anteriores fue un extracto de [6] y[7]

6.4. PROFIBUS

PROFIBUS es una red de comunicación digital que conecta y permite el intercambio de datos entre diversos dispositivos y equipos industriales, como sensores, actuadores, controladores y otros dispositivos de campo en sistemas de automatización industrial y control. Este facilita la transferencia sin problemas de datos, comandos e información entre diferentes componentes de un sistema de fabricación o proceso.

Implementación:

PROFIBUS se implementa principalmente como PROFIBUS DP (Periféricos Descentralizados). Esta versión se utiliza en la automatización de fábricas y plantas, conectando dispositivos como PLCs (Controladores Lógicos Programables), módulos de E/S y otros dispositivos de campo de manera descentralizada. Normalmente, opera sobre cables de par trenzado o fibra óptica y se estructura en una configuración maestro-esclavo, donde un controlador central (maestro) se comunica con múltiples dispositivos periféricos (esclavos). La implementación de PROFIBUS generalmente involucra los siguientes componentes y pasos: 1) Cables de Red PROFIBUS: se utilizan diferentes tipos de cables según la aplicación específica y los requisitos. 2) Maestros PROFIBUS (o Acopladores DP/PA): estos son dispositivos responsables de gestionar y controlar la comunicación en la red. 3) Esclavos PROFIBUS: estos son los dispositivos de campo o instrumentos que se comunican con el maestro. 4) Herramientas de Configuración: se utilizan herramientas de software especializadas para configurar la red y establecer los parámetros de comunicación. PROFIBUS es conocido por su alta velocidad de transmisión de datos, fiabilidad y versatilidad en la automatización industrial, lo que lo convierte en una tecnología crucial para la conexión y el control de dispositivos en entornos de fabricación y control de procesos.[8]

Archivo *General Station Description (GDS)*

GDS es una descripción de un dispositivo de E/S proporcionada por el fabricante del dispositivo. El contenido del archivo GSD consta de información de configuración, parámetros, módulos, diagnósticos, alarmas, así como la identificación del fabricante y del dispositivo. Para profundizar un poco más en los últimos dos elementos, la identificación del fabricante (ID del fabricante) es un número proporcionado por PI (PROFIBUS y PROFINET International) y es único para cada fabricante. La identificación del dispositivo (ID del dispositivo) es establecida por el fabricante del dispositivo y es única para cada familia de dispositivos. El archivo GSD es una forma estandarizada de describir la información del dispositivo para la herramienta de ingeniería y el controlador de E/S (PLC/PAC/DCS). Puede utilizarse con una variedad de herramientas de ingeniería como un conjunto estándar de información de dispositivo. El archivo GSD sera importada en la herramienta de configuración PROFINET del controlador. Una vez haya sido importado el archivo GSD, podrá instalar el dispositivo de E/S en PROFINET, y será capaz de configurar, parametrizar y elegir opciones de diagnóstico en la herramienta.

CAPÍTULO 7

Desarrollo de proyecto

7.1. Desarrollo de prototipo

Siguiendo las instrucciones del manual la Figura6 del conector CN1 indica que, para energizar las entradas del servo controlador, es necesario el uso de 24V como alimentación general del circuito de entradas. Podemos observar que todas las entradas utilizan un circuito con una resistencia de 4.7 kiloohmio y un optoacoplador para la transmisión segura de señales internas del servo controlador. El circuito mencionado se encuentra repetido y en paralelo con los 24V positivos, siendo este su fuente de alimentación, y las entradas RUN, RESET, ECRST, GSEL, GESEL, NOT y POT se accionan para el cierre del circuito, haciendo estas entradas negadas.

La imagen también revela que las salidas del conector CN1 tienen su propia alimentación y estas van a diversas sub-conexiones. La salida ALM es una salida activa, ya que esta debería conectarse a un contactor, el cual cuando esté apagada, RUN debería de estar en ON, y viceversa. La salida INP indica cuando la posición esta completa de acuerdo con los pulsos recibidos en las entadas CW y CCW. Por otro lado, la salida BKIR está directamente conectada al control de freno, mientras WARN es una salida pasiva, la cual en el manual es recomendado que este conectada a un tipo de dispositivo de señalización para indicación a los usuarios (Luz de emergencia). Adicionalmente, OGND es la tierra común del circuito y, por último Z y su GND local que es una señal que emite solo un pulso por cada rotación del eje y se utiliza como señal de reinicio y señal de inicio.

La conexión entre el CN1 y el prototipo se llevó a cabo por medio del conector R88A-CTU001N que se encuentra en Figura 13 y el módulo XW2B-34G5, este simplifica la gestión del cableado de control de las entradas y salidas del R88A-CNW01C, estableciendo la conexión con el puerto CN1. Este adaptador posibilita el aprovechamiento total de las 12 entradas y 13 salidas disponibles. Aunque el manual sugiere el uso de terminales de engarce redondas para aplicaciones industriales, con propósitos educativos se recomienda emplear terminales de horquilla para una instalación más accesible y didáctica.

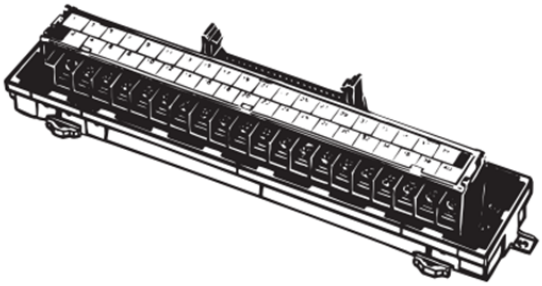


Figura 12: Módulo XW2B-34G5

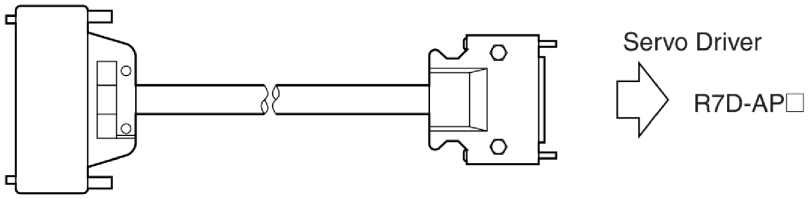


Figura 13: Cable de acople entre terminal CN1 y módulo XW2B-34G5

En mi investigación de las necesidades de las 12 entradas disponibles, únicamente 8 están habilitadas para asegurar el funcionamiento óptimo del servo controlador. Esto se debe a que CW y CCW son entradas de señales de PWM y sus respectivas referencias. Esta limitación me permitió emplear un módulo de 8 relés para establecer la comunicación entre el voltaje de bajo nivel proveniente del microcontrolador y los 24V necesarios para el correcto funcionamiento del servo controlador14.

Figura 14: Módulo de relés de 8 canales

Por medio de la Universidad del Valle de Guatemala se adquirió 25 relés de estado sólido de la marca Foteck. Los relés de estado sólido son los SSR-40 DD, estos están diseñados para el uso exclusivo de voltajes DC. Este relé tiene la capacidad en su lado de conexión tolerar voltajes entre 5V hasta 60V y del lado de accionamiento de 2.3V hasta 32V.

Figura 15: Relé de estado sólido

El Arduino Uno fue elegido para este proyecto debido a su versatilidad y capacidad. Con 14 puertos I/Os, una velocidad de reloj de 16MHz y la capacidad de generar señales PWM, es ideal para la interfaz entre el servo controlador y el PLC. Su flexibilidad, potencia y capacidad de control lo convierten en la elección idónea para garantizar una integración

eficiente y económica en el sistema. De forma especifica, se utiliza una salida PWM, 8 salidas digitales para el uso del relé, y su compatibilidad con el módulo RS-485 shield para Arduino Uno.

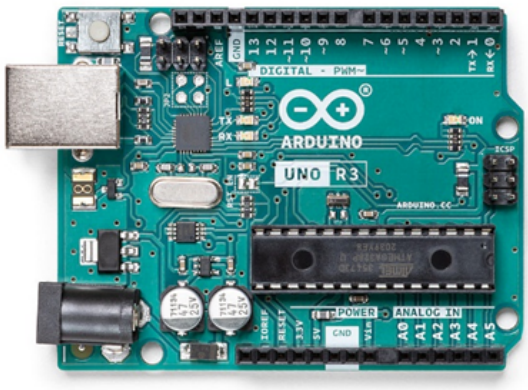


Figura 16: Arduino Uno

Al integrar todos estos componentes, se obtiene un prototipo adecuado para realizar pruebas de manejo. En este sistema, el Arduino se encarga, a través de la comunicación serial, de modificar las distintas entradas del módulo de relés y de generar las señales PWM necesarias para controlar el movimiento del servo motor.

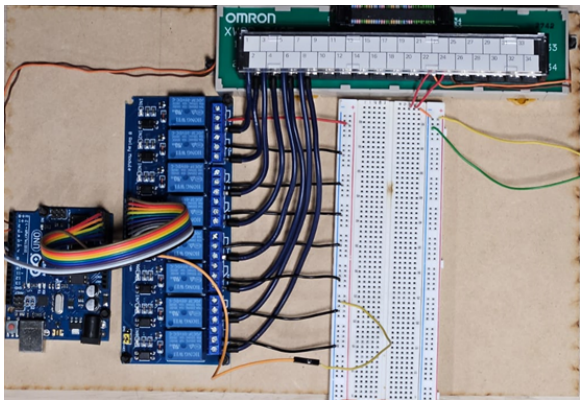


Figura 17: Primer prototipo y conexiones con módulo de relés

El prototipo representado en la Figura 17 originalmente utilizaba el módulo de relé de ocho canales; sin embargo, se optó por sustituirlo por relés de estado sólido debido a su mayor vida útil. Esta modificación implicó un aumento en las dimensiones del prototipo, lo que llevó a la decisión de fabricar dos placas de circuito impreso (PCB). Una de estas PCB se diseñó para facilitar la interconexión entre el Arduino Uno y los relés de estado sólido (Figura 18,19), mientras que la segunda se diseñó para simplificar la conexión de los pines de 24V y GND del tablero con los relés (Figura 20).

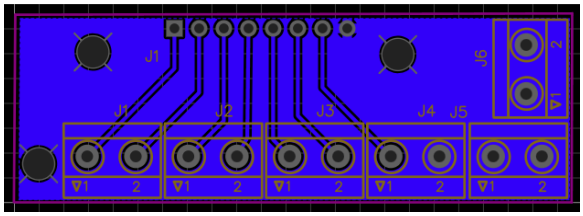


Figura 18: PCB del circuito de conexión entre Arduino Uno y relés de estado Solido

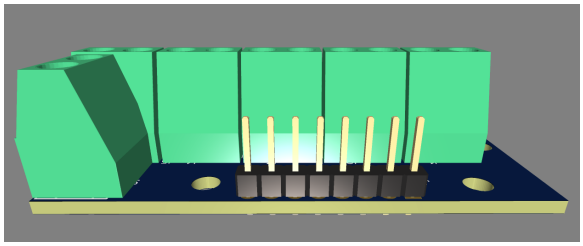


Figura 19: Modelo 3D de circuito de conexión entre Arduino Uno y relés de estado sólido

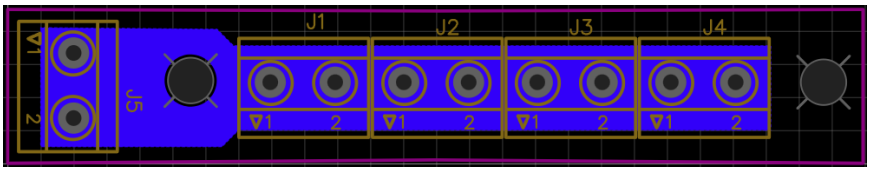


Figura 20: PCB del circuito de conexión de 24V y GND

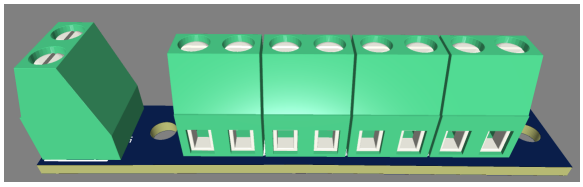


Figura 21: Modelo 3D de PCB del circuito de conexión de 24V y GND

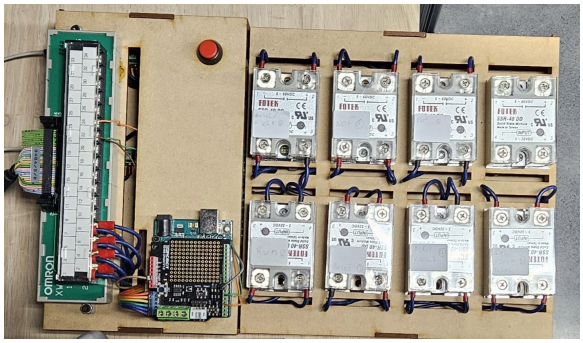
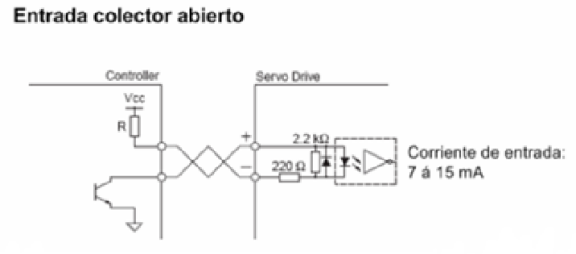


Figura 22: Segundo prototipo y conexiones con relés de estado sólido

7.2. Señal PWM

El manual indica que la señal PWM de las entradas CW y CCW deben ser de 24V o 12V, e ilustra en la Figura 23 cómo calcular la resistencia de protección y el circuito adecuado para generar la misma.



Vcc	R
24V	2k
12V	1k

Figura 23: Generación de señal PWM indicada en el manual

Después de numerosas pruebas fallidas se decidió utilizar el equipo propietario de Omron para identificar la amplitud y velocidad apropiada para el funcionamiento tanto del servo controlador como del servo motor. Se utilizó el PLC CP1L que incorpora una salida PWM, la cual fue medida y analizada.

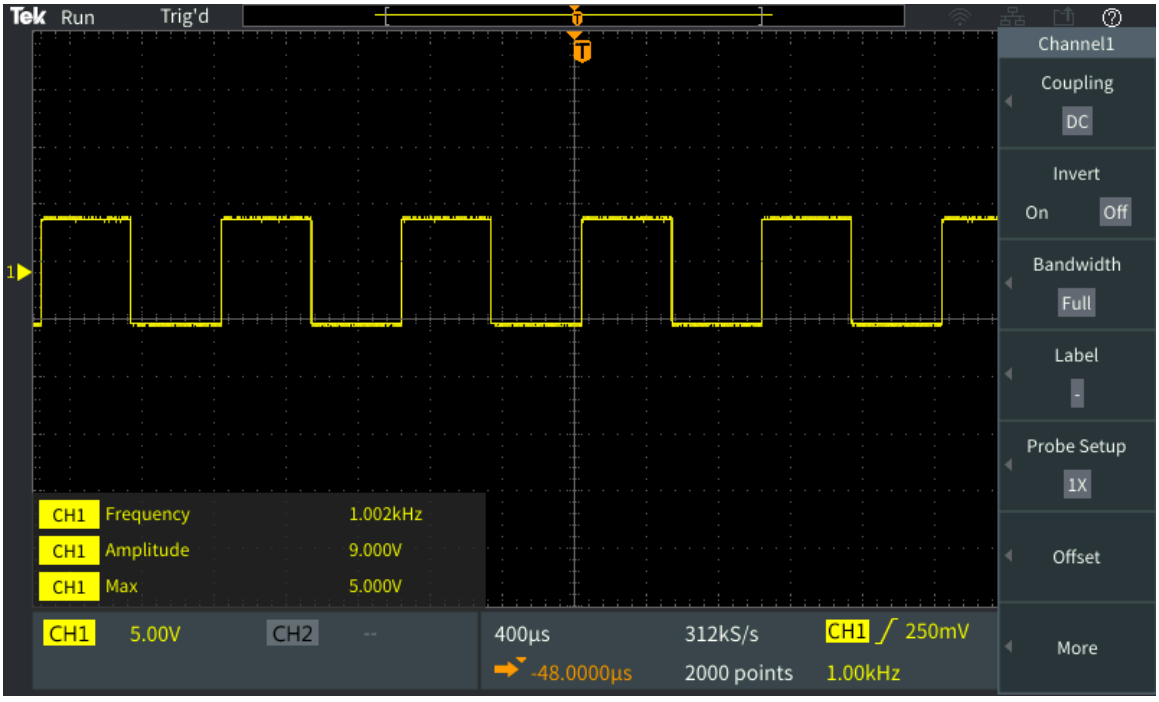


Figura 24: Señal generada por el PLC CP1L OMRON

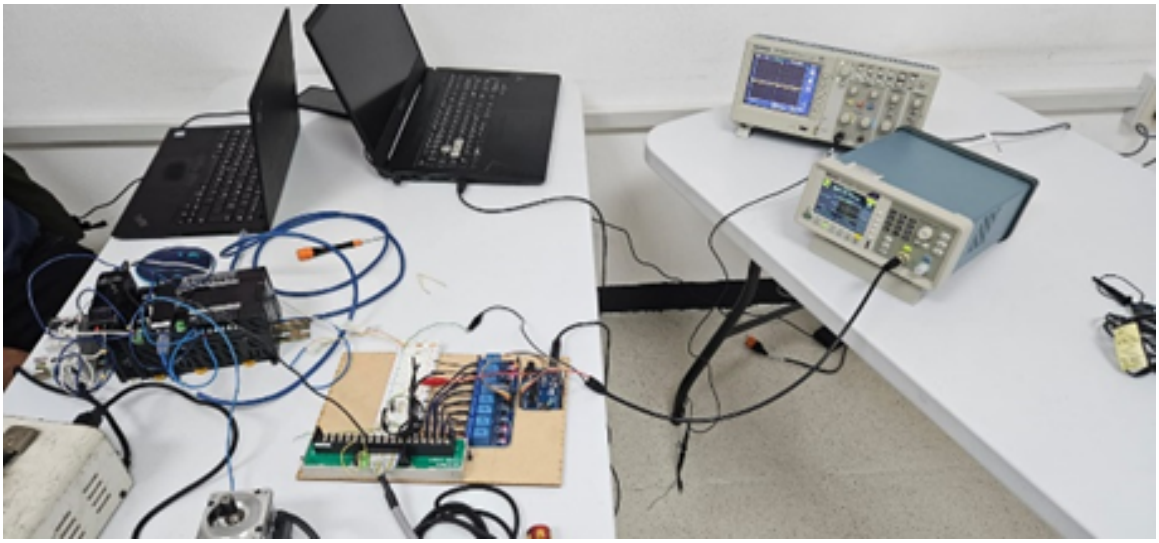


Figura 25: Verificación del prototipo con proveedor de OMRON en Guatemala

Como resultado, la amplitud adecuada puede variar entre un rango de 3V a 6V y la velocidad máxima es de 2.1MHz, lo cual reafirma que la amplitud utilizada no era la correcta, lo cual produjo las fallas del prototipo. Adicionalmente se puede concluir que la señal PWM que el servo controlador necesita está integrado por dos componentes, la frecuencia que dicta la velocidad a la cual se quiere efectuar el giro en el motor y la cantidad de pulsos dicta cuánto puede girar el servo motor. De forma específica, la frecuencia afecta la velocidad de giro y la cantidad de pulsos dicta el ángulo al cual el servo motor se debe de mover. Por último, con fines demostrativos se utilizó un generador de funciones para replicar la amplitud y velocidad de lo probado con el PLC CP1, enviando a la entrada CW/CCW una señal continua de pulsos para que el motor girara indefinidamente lo cual funcionó en su totalidad. Haciendo uso del Arduino, se generó una señal PWM y fueron examinados para contrastar dichas señales y, por otro lado, tener una señal idéntica en el microcontrolador Arduino Uno.

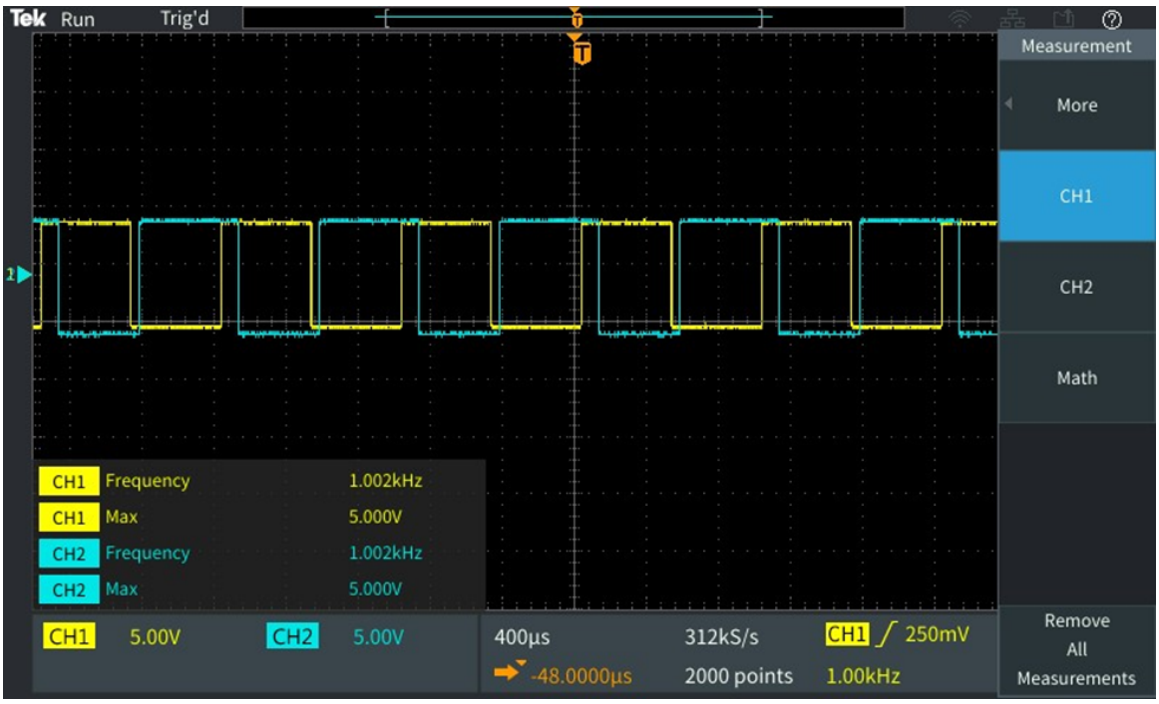


Figura 26: Comparación de Canal 1 (PLC CP1L) VS Canal 2 (Generador de señales)

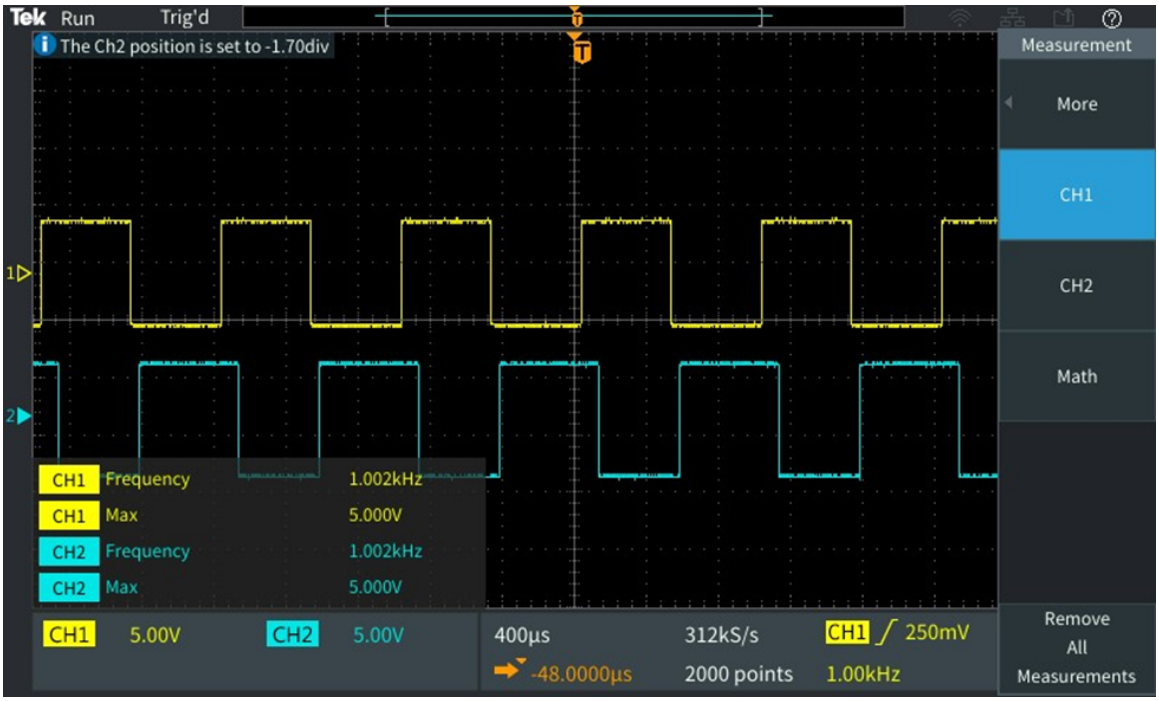


Figura 27: Comparación entre Canal 1 (Señal PWM de Arduino) y Canal 2 (Generador de señales)

7.3. Instalación en laboratorios de automatización

En la instalación en los tableros de la Universidad del Valle de Guatemala se hizo uso de la alimentación trifásica que se posee en las conexiones del guardamotor y el contactor para alimentar el servo controlador. Por otro lado, 24V fueron necesarios para energizar las entradas del puerto CN1, los cuales se obtuvieron de los módulos internos del PLC. Con fines estéticos, los cables fueron resguardados dentro de las canaletas para dar un sentido de orden y separación.

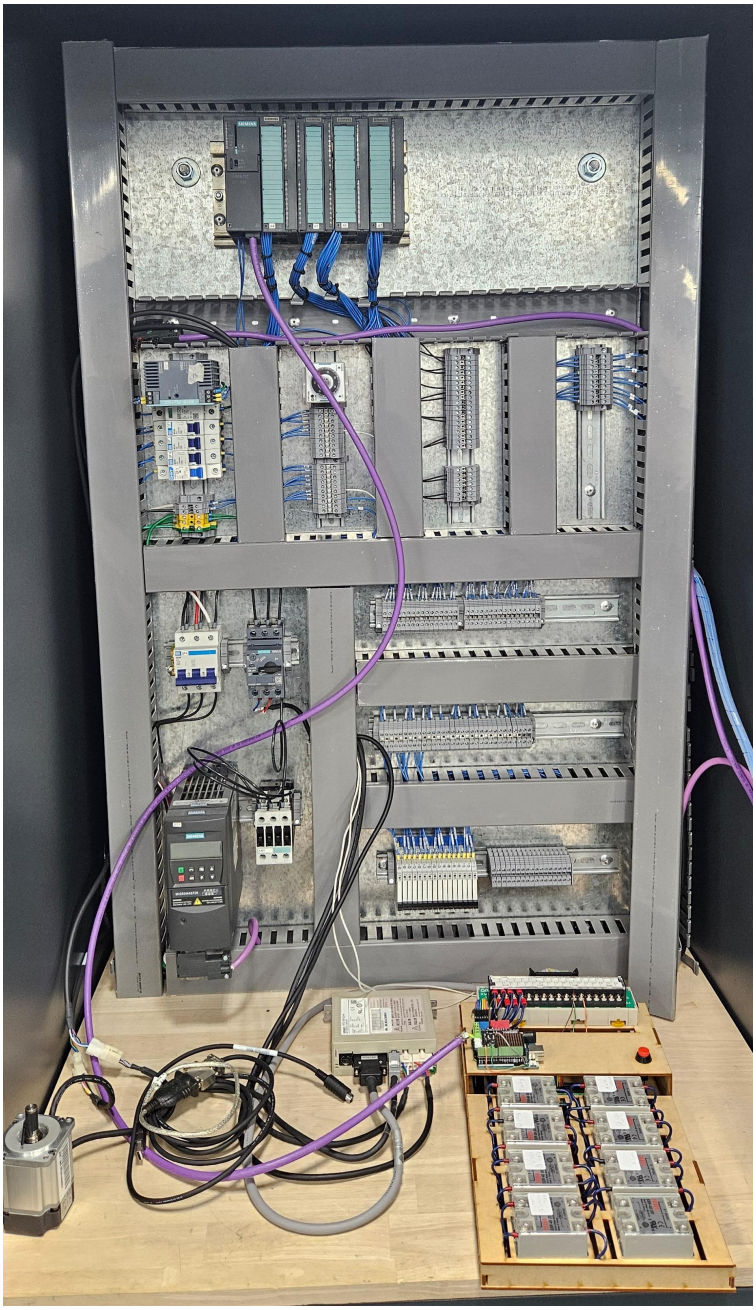


Figura 28: Instalación del prototipo en el tablero del Laboratorio de Automatización

7.4. Comunicación entre PLC y Arduino

Se utilizó el módulo RS-485 Shield para Arduino, hecho por DFROBOT, el cual está diseñado especialmente para la placa de Arduino Uno. Este puede convertir de UART a RS-485. Este escudo integra un puerto RS-485 estándar, un puerto mini RS-485 (interfaz PH2.0), encabezados RS-485 y proporciona áreas de soldadura, por lo que es conveniente para su diseño de bricolaje. El interruptor es el encargado de cambiar entre el modo de transmisión automática y manual, lo que amplía el alcance de la aplicación. El interruptor de modo de operación se coloca en OFF cuando se desea descargar el programa hacia el microcontrolador Arduino Uno y en ON para que el *shield* ejecute la tradición decaída. Es importante notar que el protocolo PROFIBUS DP tiene dos líneas independientes conocidas como línea negativa - A y línea positiva + B, que transmiten niveles de voltaje similares pero con polaridades opuestas. Por esta razón, es importante que la red esté conectada con la polaridad correcta.

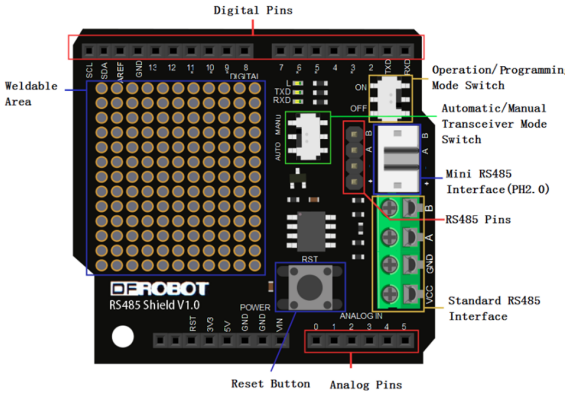


Figura 29: Módulo RS-485 Shield para Arduino UNO

PROFIBUS es un sistema de comunicación serial basado en RS-485, el cual utiliza cables de par único púrpuras y conectores estándar DB9 o M12. PROFIBUS DP puede transportar hasta 244 bytes de datos a velocidades en un rango de 9600 bits/s a 12 Mbits/s, lo que lo hace adecuado para una amplia gama de aplicaciones. Este se basa en una comunicación de maestro-esclavo, con el propósito de que la comunicación sea descentralizada, mientras que el PLC, en este caso maestro, envíe y reciba información a sus esclavos para generar una acción. La red PROFIBUS DP debe seguir estrictamente la estructura del formato del telegrama y las secuencias de telegramas definidas por la red PROFIBUS. Los diseños RS-485 utilizan el mismo UART básico, pero convierten la señal UART en una señal diferencial bidireccional. La tecnología de transmisión RS-485 utiliza un cable de dos núcleos, con el núcleo rojo positivo B+ y el núcleo verde negativo A-.

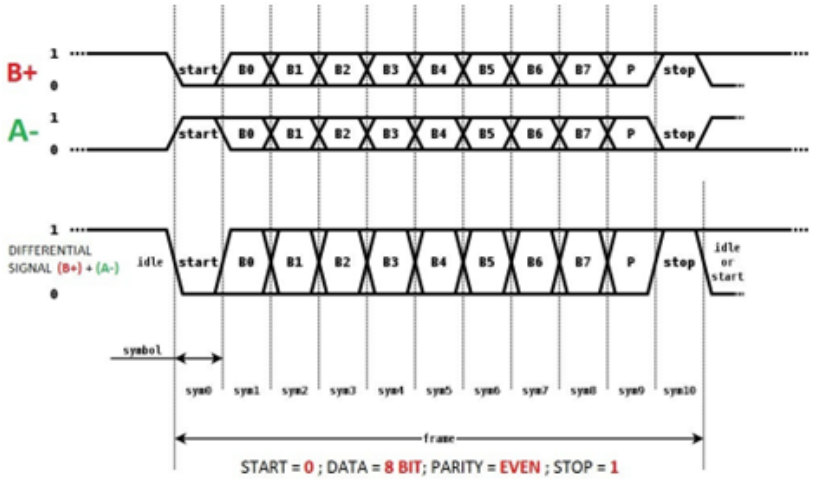


Figura 30: Codificación y formato de trama UART

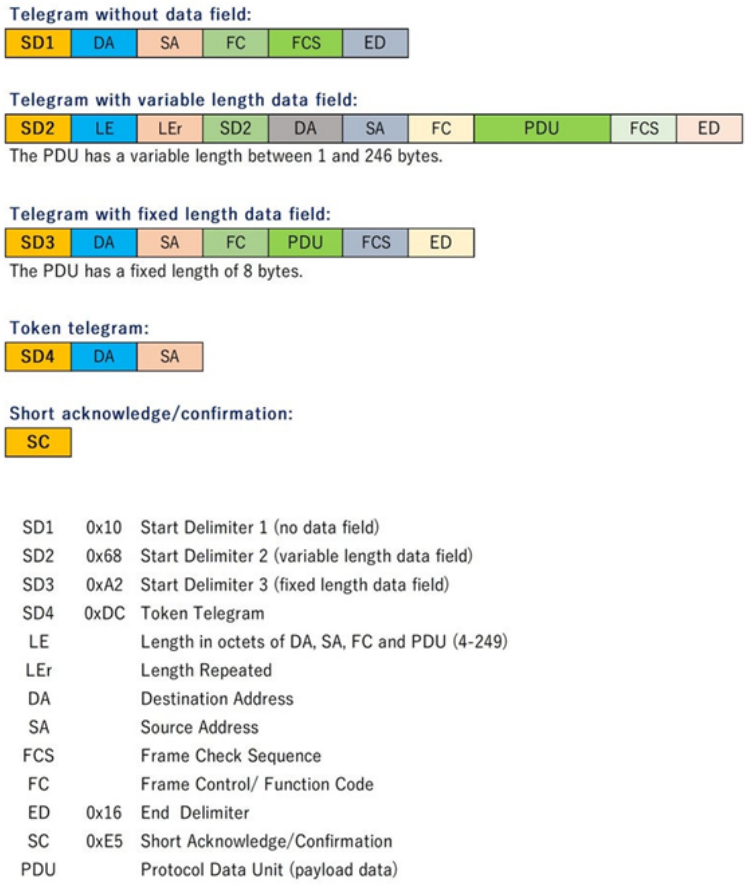


Figura 31: Formato de la trama de PROFIBUS DP

7.5. Envío de datos de PLC hacia Arduino Uno

Como mencionado anteriormente, el archivos .GDS (*"General Station Description"*) es un archivo de texto ASCII que contiene datos específicos del dispositivo, como información de identificación del proveedor, velocidades de baudios admitidas, longitud de mensaje admitida, número de datos de entrada/salida, significado de los mensajes de diagnóstico, información de sincronización, opciones y funciones admitidas, formatos de datos y señales de entradas/salidas disponibles. El archivo GDS utilizado en este proyecto fue obtenido directamente de Arduino e instalado acorde a las especificaciones del programa step 7.

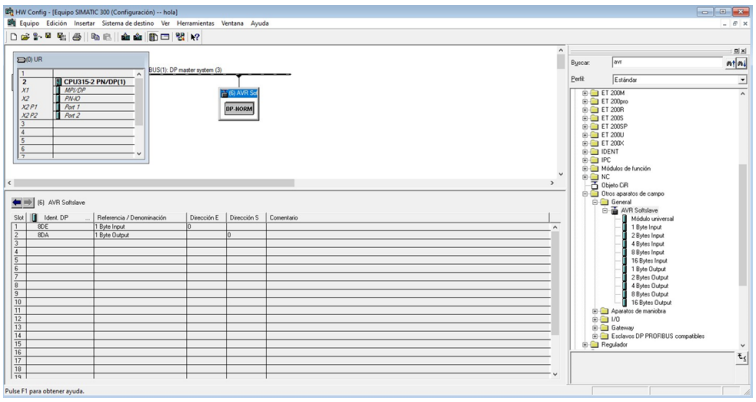


Figura 32: Configuración de hardware

En la Figura 32 se puede observar la cantidad de datos (bits/byte) que se pueden mandar o recibir y a qué espacio de memoria están asignados. Esto es importante ya que a través de estos espacios de memoria se construye la lógica de la programación del PLC.

7.6. Lógica de envío de datos

El envío de datos fue realizado por medio del programa de conexión y configuración del PLC, Simatic Manager. Se utilizando la función “MOVE” por su practicidad y bajo nivel de complejidad para descartar cualquier tipo de error. La función "MOVE", como su nombre lo indica, mueve la información de una localidad de memoria, en este caso "MB10", cuya instancia se puede observar en la Figura 33. Este espacio de memoria operativa puede ser cambiado a mano debido a que se encuentra en la tabla de variables. Luego, la información se mueve hacia la dirección “AB0”, el cual espacio de memoria indicado en la Figura 32 como la dirección de nuestro controlador, Arduino.

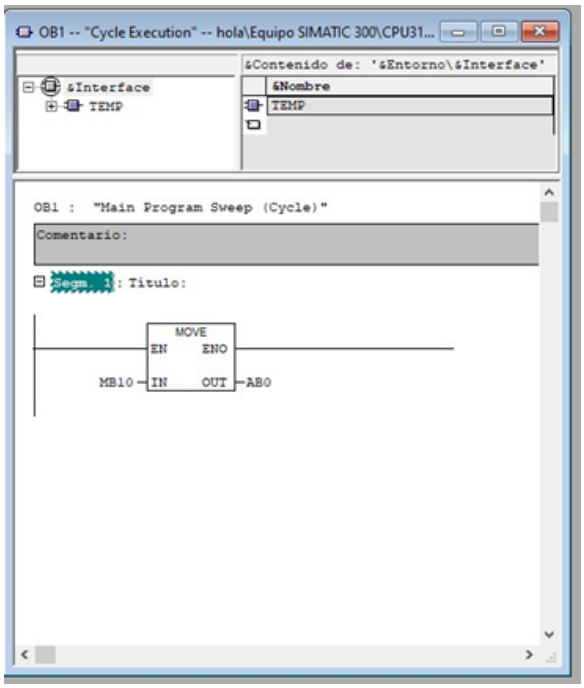


Figura 33: Lógica de envío de datos en simatic manager

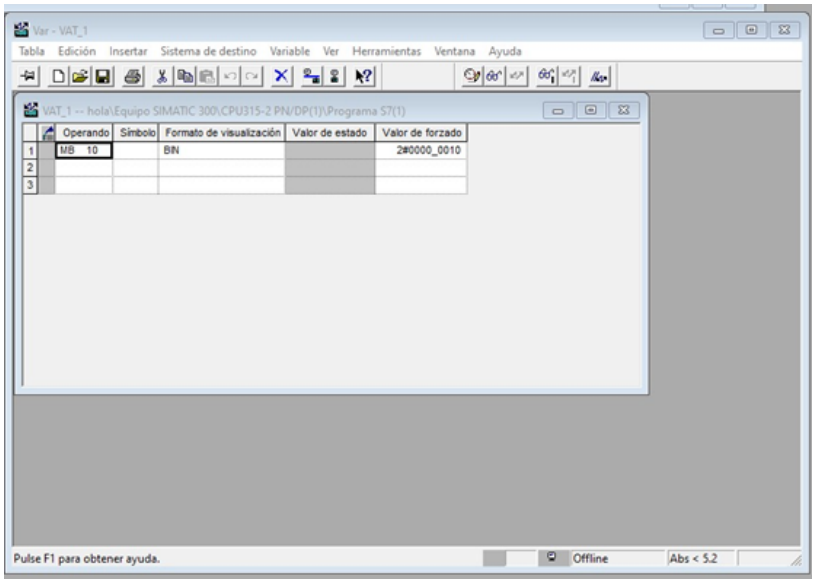


Figura 34: Tabla de variables

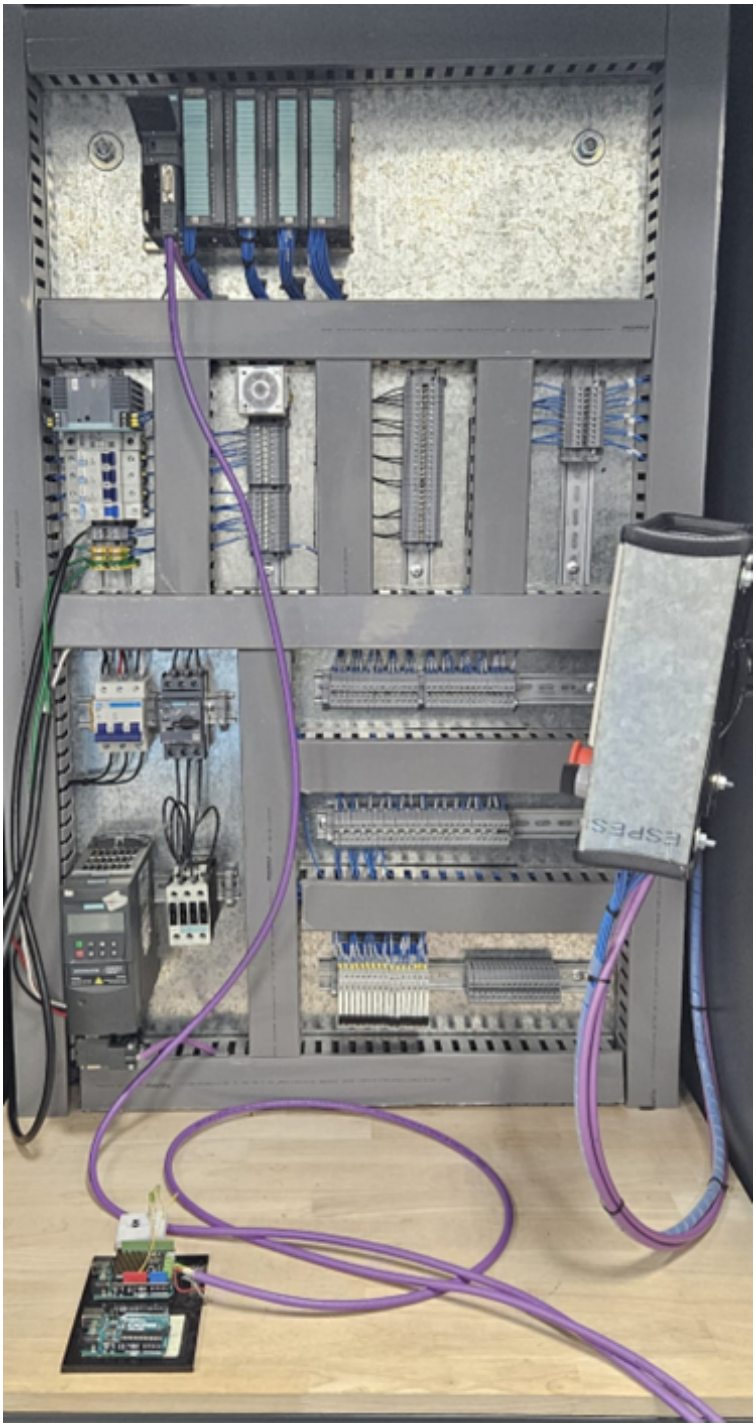


Figura 35: Prototipo de comunicación

En las pruebas iniciales se conectó a la terminal de PROFIBUS DP del PLC con su respectivo cable hacia el Shile RS-485 con un Arduino para poder realizar pruebas de comunicación. El envío de datos fue un byte cambiando únicamente el primer bit para encender y apagar un led.

7.7. Programación del Arduino Uno

La programación del microcontrolador Arduino Uno se dividió en dos partes: el control de las entradas y salidas del servo controlador, y la generación de las señales PWM, así como el protocolo de comunicación mediante PROFIBUS DP.

1) Control del sistema y generación de las señales PWM Esta primera etapa se enfocó en las señales adecuadas para el funcionamiento de las entradas en el Puerto CN1, con el fin de saber la extensión y el funcionamiento de la configuración de fábrica debido a la falta de licencia de CX Drive. La librería PWM.h fue utilizada para fabricar de la señal PWM enviada al servo controlador. Esta librería hace uso de los relojes internos del microcontrolador, Arduino Uno dedicados a las salidas PWM del mismo. En la sección de Anexos podemos ver de forma más puntual esta parte bajo la sección Control de sistema E/S.

2) Protocolo de comunicación mediante PROFIBUS DP El programa de Arduino lleva a cabo dos partes vitales en un enlace de comunicación. En primer lugar, la recepción y procesamiento de los datos. La desconstrucción del telegrama que se recibe separando y leyendo los distintos parámetros (la Figura 31 denota los parámetros a leer). Por ultimo, la construcción del telegrama con todos sus parámetros y el envió de datos.

7.8. Resultados de implementación de modos de movimiento servo controlador R7D-BP04H

Es importante notar que para la evaluación de estos modos se utilizó el programa CX-Drive, y fui acompañado de los técnicos representantes de la marca aqui en guatemala.

- **Alto Control de Posición** (*High Function Position control*)

  Este modo está preconfigurado de fábrica y no requiere modificar ningún parámetro. Este modo lee 25 pulsos ya sea en la entrada CW o CWW que se traduce a un grado de movimiento en el motor, el cambio de este modo incrementa el numero de pulsos por grado de movimiento. En conclusión, obtienes un modo listo para implementarse que se ajusta automáticamente al comportamiento tanto de la frecuencia de la señal PWM como a la cantidad de pulsos.

- **Control de velocidad establecido internamente** (*Internal velocity control*)

  Este modo hace uso de 8 parámetros los cuales no están activados por defecto estos son los siguientes:

No. de parámetro	Nombre de parámetro	Valor de parametro
Pn02	Control Mode Selection	1
Pn06	Designación de velocidad cero	1
Pn53	Configuración de velocidad interna No. 1	Rango de
Pn54	Configuración de velocidad interna No. 2	-20000
Pn55	Configuración de velocidad interna No. 3	a
Pn56	Configuración de velocidad interna No. 4	20000 r/min
Pn58	Tiempo de aceleración de arranque suave	Configuración
Pn59	Tiempo de desaceleración del arranque suave	de velocidad X 2 ms

Cuadro 13: Parámetros para el control interno de la velocidad

Con la ayuda del equipo técnico del representante de la marca en Guatemala, se realizaron los ajustes a los parámetros previamente discutidos y se alternó entre las velocidades establecidas internamente. Este proceso es controlado mediante las Entradas de Selección de Velocidad Interna 1 y 2 (VSEL1: CN1-6, VSEL2: CN1-4), lo cual ha demostrado ser un método efectivo y confiable para el cambio de velocidad sin la necesidad de utilizar cambios de frecuencia de la señal PWM. Los técnicos recomendaron dos opciones para poder utilizar todas las funciones del servo controlador: una es la compra de la licencia del programa CX Drive y la otra es la adquisición del dispositivo de unidad de parámetros R88A-PR02G.

Figura 36: Unidad de parámetros R88A-PR02G

7.9. Funcionamiento del sistema de control

En el presente trabajo, se buscó controlar el servo controlador R7D-BP04H y el servomotor R88M-G10030H, y adaptarlos al laboratorio de automatización e instrumentación industrial de la Universidad del Valle de Guatemala. Con este objetivo en mente, se decidió utilizar el protocolo PROFIBUS DP como canal de comunicación entre el PLC y el Arduino, dado que todos los PLCs del laboratorio cuentan con este tipo de comunicación. Debido a la naturaleza del protocolo PROFIBUS DP y la estructura de datos proveída del GDS, se tiene 8 bits de comunicación entre el PLC y el Arduino Uno. Como previamente mencionado el servo controlador utiliza la frecuencia de la señal PWM como indicador de velocidad y la cantidad de pulsos como la cantidad de grado a la cual este debe de girar.

Es importante destacar que la licencia del programa CX Drive fue adquirida de manera provisional gracias al período de prueba solicitado a los representantes de la marca en Guatemala. Además, el control de parámetros, gráficas, tiempos de aceleración y desaceleración son parte de los elementos educativos para el uso completo de este dispositivo. Con esta limitación en mente, se decidió ampliar el rango del prototipo con el fin de poder ejecutar cualquier función del servo controlador, basándose en la modificación de los parámetros a través del programa en caso de que se decida adquirir dicho programa.

El prototipo que utiliza el módulo de relés de 8 canales y el prototipo de relés de estado solido interpretan los bits de la siguiente manera:

Bits de comunicación	Puerto interno Arduino	Conexión al puerto CN1
Bit No. 0	A3	RUN
Bit No. 1	Cambio de Frecuencia para CCW	
Bit No. 2	Cambio de dirección de Giro	
Bit No. 3	A5	ECRST & VSEL2
Bit No. 4	7	GSEL & VZERO & TLSEL
Bit No. 5	A2	GESEL & VSEL1
Bit No. 6	A1	NOT
Bit No. 7	A0	POT

Cuadro 14: Significado de Bits de Implementación

Como se demuestra previamente en el capitulo 7.6, la facilidad del accionamiento de los puerto CN1 permite la integración de este sistema a la clase y laboratorio de automatización e instrumentación industrial.

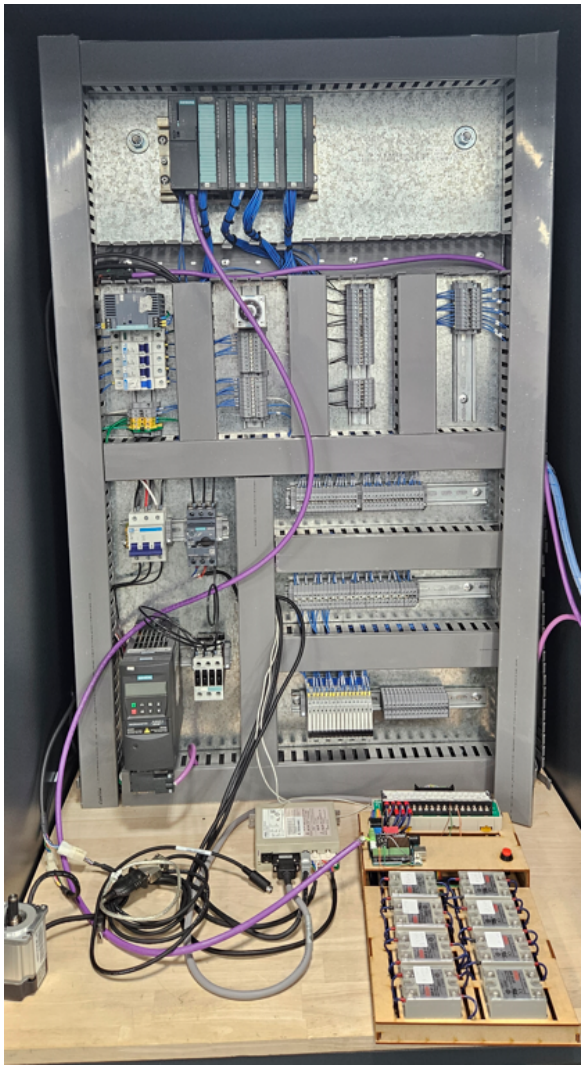


Figura 37: Prototipo en el tablero del laboratorio de automatización

CAPÍTULO 8

Conclusiones

- La instalación del servo controlador y el servomotor en el laboratorio de automatización e instrumentación industrial se llevó a cabo de manera apropiada, utilizando los componentes de seguridad disponibles en el panel. En las especificaciones del modelo del servo controlador indica que utiliza 220V Ac para su funcionalidad.
- Se estableció exitosamente el protocolo de comunicación PROFIBUS DP entre el Arduino Uno y el PLC S7-300, como se puede apreciar en la Figura 32.
- Se fabricó una señal PWM con una amplitud de 5V y un rango de frecuencia de 500Hz a 6KHz mediante el Arduino Uno para el funcionamiento del servo controlador R7D-BP04H, según se muestra en la Figura 27.
- El microcontrolador responsable de coordinar la comunicación y el accionamiento para el funcionamiento completo del servo controlador y el servomotor fue el Arduino Uno.
- La implementaron del nodo de control de posición de alta función fue exitosa, sin cambios de parámetros en el servo controlador.
- La implementación del modo de control de velocidad interna fue exitosa, y se determino que para esta función es necesario el uso de el programa CX Drive, para la activación de los parámetros adecuados.
- La elaboración del programa para el control de la comunicación y la gestión de los puertos del servo controlador resultó exitosa.
- Se desarrollar un manual de usuario para el funcionamiento de modos y uso de equipo.
- Se determinó que los diferentes modos de operación, tiempos de aceleración/desaceleración, gráficas en tiempo real, etc, pueden ser accedidas sí y solo sí, se tiene el programa CX Drive.

CAPÍTULO 9

Recomendaciones

- Se recomienda considerar la fabricación de una PCB para minimizar el uso de espacio contemplando el uso de relés de esta solidos y adaptable para el montaje del microcontrolador.
- Se recomienda explorar las distintas opciones de los del archivo GDS ya que posee las opciones de enviar datos de un bit, un byte y 2 byte tanto como input y como output.
- Es vital la adquisición del programa CX Drive, debido a que sin el mismo se tiene un muy corto alcance en las funciones del servo controlador.

CAPÍTULO 10

Bibliografía

[1] A. Salkic y H. Muhovic, «Siemens S7-1200 PLC DC Motor control capabilities,» en *IFAC CONFERENCE 2022*, vol. 5, 2022, págs. 103-108. DOI: 10.1109/ICCRD.2022.554103.

[2] D. Yulin y Z. Chunjiao, «Design and research of embedded PLC development system,» en *2011 3rd International Conference on Computer Research and Development*, vol. 3, 2011, págs. 226-228. DOI: 10.1109/ICCRD.2011.5764286.

[3] Y. Kuang, «Communication between PLC and Arduino Based on Modbus Protocol,» en *2014 Fourth International Conference on Instrumentation and Measurement, Computer, Communication and Control*, 2014, págs. 370-373. DOI: 10.1109/IMCCC.2014.83.

[4] Omron, «Smart Step 2 R7D-BP y R88D-GP08H User Manual,» en Servosistem de C.A., 2014, cap. 1.

[5] Omron, «Guía Rápida Serie Smart Step 2,» en Omron, 2014, cap. 1.

[6] Omron, «Serie Smart Step 2 R7D-BP y R88D-GP08H,» en Servosistem de C.A., 2015, cap. 1.

[7] Omron, «Servomotores de la serie G,» en Servosistem de C.A., 2014, cap. 1.

[8] M. Felser, «PROFIBUS Manual,» en Freitag, 2017.

CAPÍTULO 11

Anexos

11.1. Control de sistema E/S

```
#include <stdio.h>
#include <stdlib.h>
#include <PWM.h>

int POW                = (A5);
int RUN                = (A4);
int RESET              = (A3);
int ECRST_VSEL2        = (A2);
int GSEL_VZERO_TLSEL = (A1);
int GSEL_VSEL1         = (A0);
int NOT                = (7);
int POT                = (8);
int cww                = (2);
int PWM_port           = (3);
int32_t frequency      = 500;
int h ;
char temp               =  0 ;

void setup()
{
 Serial.begin(9600);
//=============================pwm set up=============
 InitTimersSafe();
 bool success = SetPinFrequencySafe(PWM_port , frequency);
 if (!success) {Serial.println("Failed to set frequency!");}
 else{Serial.print("yes");}
//=====================================================
```

```
 pinMode(POW   , OUTPUT);
 pinMode(RUN, OUTPUT);
 pinMode(RESET, OUTPUT);
 pinMode(ECRST_VSEL2, OUTPUT);
 pinMode(GSEL_VZERO_TLSEL, OUTPUT);
 pinMode(GSEL_VSEL1, OUTPUT);
 pinMode(NOT , OUTPUT);
 pinMode(POT, OUTPUT);
 pinMode(cww, OUTPUT);
 digitalWrite(POW,HIGH);
 digitalWrite(RUN,HIGH);
 digitalWrite(RESET,HIGH);
 digitalWrite(ECRST_VSEL2,HIGH);
 digitalWrite(GSEL_VZERO_TLSEL,HIGH);
 digitalWrite(GSEL_VSEL1,HIGH);
 digitalWrite(NOT ,HIGH);
 digitalWrite(POT,HIGH);
 digitalWrite(RESET,LOW);
 delay(1000);
 digitalWrite(RESET,HIGH);
 digitalWrite(cww,LOW);
//run normal sin pulsos    //
  digitalWrite(POW,LOW);   //
  digitalWrite(RUN,LOW);   //
  digitalWrite(NOT ,LOW);  //
  digitalWrite(POT,LOW);   //
////////////////////////////
}
void loop() {
// cambio de rotacion del servo    //
  while (Serial.available()){      //
    temp=Serial.read();            //
      if(temp =='R'){              //
        digitalWrite(cww,HIGH);    //
        Serial.println("Right");   //
      }                            //
      else if(temp =='L'){         //
        digitalWrite(cww,LOW);     //
        Serial.println("left");    //
      }
// cambio de frecuencia del servo/////////////////
      else if(temp =='1'){         //
        frequency=1000;            //
        h=1;                       //
        Serial.println("VEL 1");
      }                            //
      else if(temp =='2'){         //
        frequency=2000;            //
```

```
  h=1;                          //
  Serial.println("VEL 2");
}
else if(temp =='3'){            //
  frequency=3000;               //
  h=1;                          //
  Serial.println("VEL 3");
}
else if(temp =='4'){            //
  frequency=4000;               //
  h=1;                          //
  Serial.println("VEL 4");
}
//////////////////////////////////////////
else if(temp == 'V'){
  digitalWrite(GSEL_VSEL1,LOW);
  Serial.println("GSEL_VSEL1 on");
}
else if(temp == 'v'){
  digitalWrite(GSEL_VSEL1,HIGH);
  Serial.println("GSEL_VSEL1 off");
}
//////////////////////////////////////////
else if(temp == 'B'){
  digitalWrite(ECRST_VSEL2,LOW);
  Serial.println("GSEL_VSEL2 on");
}
else if(temp == 'b'){
  digitalWrite(ECRST_VSEL2,HIGH);
  Serial.println("GSEL_VSEL2 off");
}
//////////////////////////////////////////
else if(temp == 'K'){
  digitalWrite(GSEL_VZERO_TLSEL,LOW);
  Serial.println("GSEL_VZERO_TLSEL on");
}
else if(temp == 'k'){
  digitalWrite(GSEL_VZERO_TLSEL,HIGH);
  Serial.println("GSEL_VZERO_TLSEL off");
}
//////////////////////////////////////////
else if(temp == 'P'){
  digitalWrite(POT,LOW);
  Serial.println("POT on");
}
else if(temp == 'p'){
  digitalWrite(POT,HIGH);
  Serial.println("POT OFF");
```

```
      }
      ////////////////////////////////////////
      else if(temp == 'N'){
        digitalWrite(NOT,LOW);
        Serial.println("NOT on");
      }
      else if(temp == 'n'){
        digitalWrite(NOT,HIGH);
        Serial.println("NOT OFF");
      }
      ////////////////////////////
    Serial.println(temp);
    }
//////////////////////////////////////////////////////////
if (h==1){                                               //
    Serial.print(" Vel change ");                        //
    bool set=SetPinFrequencySafe(PWM_port,frequency);    //
    if(!set){Serial.println("NOT set ");}                //
    else{Serial.println("set RIGHT");}                   //
    h=0;                                                 //
  }                                                      //
pwmWrite(PWM_port, 155);                                 //
//////////////////////////////////////////////////////////
}
```

11.2. Programa de comunicación

```
//  DELAY_TBIT = 44 // Working on Arduino @16MHz
#define BAUD 45450

#include <SoftwareSerial.h>
SoftwareSerial myserial(8,9);


//#define VCC_PIN     8
//#define GND_PIN     9
#define LED_PIN     5
#define TOUCH_PIN   7

bool bit1;
bool bit2;


unsigned long samplingtime = 0;
// This is timer delay for 1 TBIT
#define DELAY_TBIT      44
```

```
// This pin toggles the MAX485 IC from Transmit to Recieve
#define TX_ENABLE_PIN   2
// This is the address of this slave device
#define SLAVE_ADDRESS   6
// This is the built in led on pin 13 Arduino Uno - PB5
#define LED_ERROR_PIN   13

////////////////////////////////////////////////
////////////////////////////////////////////////
#define LED_ERROR_ON    PORTB |= (1<<5);
#define LED_ERROR_OFF   PORTB &= ~(1<<5);

#define TX_ENABLE_ON    PORTD |= (1<<TX_ENABLE_PIN);
#define TX_ENABLE_OFF   PORTD &= ~(1<<TX_ENABLE_PIN);
////////////////////////////////////////////
#define TIMER1_RUN TCCR1B |= (1 << CS11);
#define TIMER1_STOP TCCR1B &= ~(1 << CS11);
////////////////////////////////////////////////////
// Ident Nummer DP Slave. Arbitrarily chosen.
// This ID not found in my .GSD library.
////////////////////////////////////////////////
#define IDENT_HIGH_BYTE       0xC0
#define IDENT_LOW_BYTE        0xDE
////////////////////////////////////////////////
////////////////////////////////////////////////
// Addresses
////////////////////////////////////////////////////
// Service Access Point Adress Offset
#define SAP_OFFSET            128
#define BROADCAST_ADD         127
#define DEFAULT_ADD           126   // Delivery address
////////////////////////////////////////////////

////////////////////////////////////////////////
// Command types
////////////////////////////////////////////////
#define SD1                   0x10  // Telegram without data field
#define SD2                   0x68  // Data telegram variable
#define SD3                   0xA2  // Data telegram fixed
#define SD4                   0xDC  // Token
#define SC                    0xE5  // Short acknowledgment
#define ED                    0x16  // End
////////////////////////////////////////////

////////////////////////////////////////////
// Function Codes
////////////////////////////////////////////
/* FC Request */
```

```
// SPS: Status query
#define FDL_STATUS              0x09
// SPS: Set outputs, read inputs
#define SRD_HIGH                0x0D
#define FCV_                    0x10
#define FCB_                    0x20
#define REQUEST_                0x40

/* FC Response */
#define FDL_STATUS_OK           0x00  // SLA: OK
// SLA: (Data low) Send data inputs
#define DATA_LOW                0x08
// SLA: (Data high) Diagnosis pending
#define DIAGNOSE                0x0A
//////////////////////////////////////////////////

//////////////////////////////////////////////
// Service Access Points (DP Slave) MS0
//////////////////////////////////////////
// Master sets slave address, slave responds with SC
#define SAP_SET_SLAVE_ADR       55
// Master requests input data, slave sends input data
#define SAP_RD_INP              56
// Master requests output data, slave sends output data
#define SAP_RD_OUTP             57
// Master Control, Slave Does not answer
#define SAP_GLOBAL_CONTROL      58
// Master requests config., Slave sends configuration
#define SAP_GET_CFG             59
// Master requests diagnosis, slave sends diagnosis Daten
#define SAP_SLAVE_DIAGNOSIS     60
 // Master sends parameters, slave sends SC
#define SAP_SET_PRM             61
// Master sends configuration, Slave sends SC
#define SAP_CHK_CFG             62
//////////////////////////////////////////

//////////////////////////////////////////
// SAP: Global Control (Data Master)
//////////////////////////////////////////
#define CLEAR_DATA_             0x02
#define UNFREEZE_               0x04
#define FREEZE_                 0x08
#define UNSYNC_                 0x10
#define SYNC_                   0x20
//////////////////////////////////////

//////////////////////////////////////
```

```
// SAP: Diagnosis (Answer slave)
//////////////////////////////////////////
/* Status Byte 1 */
#define STATION_NOT_EXISTENT_ 0x01
#define STATION_NOT_READY_    0x02
#define CFG_FAULT_            0x04
#define EXT_DIAG_             0x08
#define NOT_SUPPORTED_        0x10
#define INV_SLAVE_RESPONSE_   0x20
#define PRM_FAULT_            0x40
#define MASTER_LOCK           0x80

/* Status Byte 2 */
#define STATUS_2_DEFAULT      0x04
#define PRM_REQ_              0x01
#define STAT_DIAG_            0x02
#define WD_ON_                0x08
#define FREEZE_MODE_          0x10
#define SYNC_MODE_            0x20
#define DEACTIVATED_          0x80

/* Status Byte 3 */
#define DIAG_SIZE_OK          0x00
#define DIAG_SIZE_ERROR       0x80

/* Address */
#define MASTER_ADD_DEFAULT    0xFF

/* Extended diagnosis (EXT_DIAG_ = 1) */
// Bit 6-7 ist Diagnose Typ
#define EXT_DIAG_TYPE_        0xC0
// Wenn Bit 7 und 6 = 00, dann Geraetebezogen
#define EXT_DIAG_GERAET       0x00
 // Wenn Bit 7 und 6 = 01, dann Kennungsbezogen
#define EXT_DIAG_KENNUNG      0x40
 // Wenn Bit 7 und 6 = 10, dann Kanalbezogen
#define EXT_DIAG_KANAL        0x80
// Bit 0-5 sind Anzahl der Diagnose Bytes
#define EXT_DIAG_BYTE_CNT_    0x3F
//////////////////////////////////////////
// SAP: Set Parameters Request (Data master)
//////////////////////////////////////////
/* Command */
// Slave fuer andere Master gesperrt
#define LOCK_SLAVE_           0x80
// Slave fuer andere Master freigegeben
#define UNLOCK_SLAVE_         0x40
#define ACTIVATE_SYNC_        0x20
```

```
#define ACTIVATE_FREEZE_        0x10
#define ACTIVATE_WATCHDOG_      0x08

/* DPV1 Status Byte 1 */
#define DPV1_MODE_              0x80
#define FAIL_SAVE_MODE_         0x40
#define PUBLISHER_MODE_         0x20
#define WATCHDOG_TB_1MS         0x04

/* DPV1 Status Byte 2 */
#define PULL_PLUG_ALARM_        0x80
#define PROZESS_ALARM_          0x40
#define DIAGNOSE_ALARM_         0x20
#define VENDOR_ALARM_           0x10
#define STATUS_ALARM_           0x08
#define UPDATE_ALARM_           0x04
#define CHECK_CONFIG_MODE_      0x01

/* DPV1 Status Byte 3 */
#define PARAMETER_CMD_ON_       0x80
#define ISOCHRON_MODE_ON_       0x10
#define PARAMETER_BLOCK_        0x08
////////////////////////////////
// SAP: Check Config Request (Data Master)
////////////////////////////////
// Bit 4-5 is direction.
// 01 = input, 10 = output, 11 = input / output
#define CFG_DIRECTION_          0x30
#define CFG_INPUT               0x10  // Input
#define CFG_OUTPUT              0x20  // Output
#define CFG_INPUT_OUTPUT        0x30  // Input/Output
// Special format if more than 16/32 bytes are to be transferred
#define CFG_SPECIAL             0x00
// Bit 7 is consistency. 0 = byte or word, 1 = over
#define CFG_KONSISTENZ_         0x80   entire module
#define CFG_KONS_BYTE_WORT      0x00  // Byte or word
#define CFG_KONS_MODUL          0x80  // Modul
// Bit 6 is IO width. 0 = byte (8bit), 1 = word (16bit)
#define CFG_WIDTH_              0x40
#define CFG_BYTE                0x00  // Byte
#define CFG_WORD                0x40  // Word

/* Compact format */
// Bit 0-3 are number of bytes or words.
// 0 = 1 byte, 1 = 2 bytes etc.
#define CFG_BYTE_CNT_           0x0F

/* Special format */
```

```
// Bit 6-7 is direction.
// 01 = input, 10 = output, 11 = input / output
#define CFG_SP_DIRECTION_     0xC0
#define CFG_SP_VOID           0x00  // Empty space
#define CFG_SP_INPUT          0x40  // Input
#define CFG_SP_OUTPUT         0x80  // Output
#define CFG_SP_INPUT_OPTPUT   0xC0  // Input/Output
// Bits 0-3 are the number of manufacturer-specific
#define CFG_SP_VENDOR_CNT_    0x0F   bytes. 0 = none

/* Special format / length byte */
// Bit 0-5 are number of bytes or words.
// 0 = 1 byte, 1 = 2 bytes etc.
#define CFG_SP_BYTE_CNT_      0x3F
//////////////////////////////////////////////////////
#define TIMEOUT_MAX_SYN_TIME  33 * DELAY_TBIT // 33 TBit = TSYN
#define TIMEOUT_MAX_RX_TIME   15 * DELAY_TBIT
#define TIMEOUT_MAX_TX_TIME   15 * DELAY_TBIT
#define TIMEOUT_MAX_SDR_TIME  15 * DELAY_TBIT // 15 Tbit = TSDR
//////////////////////////////////////////

//////////////////////////////////////////
#define MAX_BUFFER_SIZE       32
// Number of bytes coming from the master
#define INPUT_DATA_SIZE       16
// Number of bytes going to master
#define OUTPUT_DATA_SIZE      16
// Number of bytes for manufacturer-specific data
#define VENDOR_DATA_SIZE      0
// Number of bytes for extended diagnostics
#define EXT_DIAG_DATA_SIZE    0

#define OUTPUT_MODULE_CNT     1     // Number of output modules
#define INPUT_MODULE_CNT      1     // Number of input modules
//////////////////////////////////////

//////////////////////////////////////
// Profibus process control
//////////////////////////////////////
#define PROFIBUS_WAIT_SYN     1
#define PROFIBUS_WAIT_DATA    2
#define PROFIBUS_GET_DATA     3
#define PROFIBUS_SEND_DATA    4
//////////////////////////////////////
unsigned char uart_buffer[MAX_BUFFER_SIZE];
unsigned int  uart_byte_cnt = 0;
unsigned int  uart_byte_cnt0 = 0;
unsigned int  uart_transmit_cnt = 0;
```

```
unsigned int  uart_transmit_cnt0 = 0;
unsigned long lastmillis;
// Profibus Flags and Variables
unsigned char profibus_status;
unsigned char diagnose_status;
unsigned char slave_addr;
unsigned char master_addr;
unsigned char group;
volatile unsigned char new_data;

#if (OUTPUT_DATA_SIZE > 0)
volatile unsigned char Profibus_out_register[OUTPUT_DATA_SIZE];
#endif
#if (INPUT_DATA_SIZE > 0)
unsigned char Profibus_in_register [INPUT_DATA_SIZE];
#endif
#if (VENDOR_DATA_SIZE > 0)
unsigned char Vendor_Data[VENDOR_DATA_SIZE];
#endif
#if (EXT_DIAG_DATA_SIZE > 0)
unsigned char Diag_Data[EXT_DIAG_DATA_SIZE];
#endif
unsigned char Input_Data_size;
unsigned char Output_Data_size;
// Number of read-in manufacturer-specific bytes
unsigned char Vendor_Data_size;

void setup() {

  pinMode(LED_ERROR_PIN,OUTPUT); // CPU RUN/STOP LED
  pinMode(TX_ENABLE_PIN,OUTPUT); // TX enable output

  pinMode(LED_PIN, OUTPUT);
  pinMode(TOUCH_PIN, OUTPUT);

  myserial.begin(9600);
  myserial.flush();
  init_Profibus();
  TX_ENABLE_OFF; // Disable Transmit (Switch to Recieve)
}
////////////////////////////////////
////////////////////////////////////
void loop() {

  if(new_data==1)
  {
      //Profibus_out_register[0] +=new_data;
      new_data=0;
```

```
      //Profibus_out_register[0] = Profibus_in_register[0];
      bit1=bitRead(Profibus_in_register[0],0);
      bit2=bitRead(Profibus_in_register[0],4);
 }
    // Correct the led state on Touch button
  digitalWrite(LED_PIN,  bit1);
  digitalWrite(TOUCH_PIN, bit2);


}
//////////////////////////////////////////////////////
//////////////////////////////////////////////////////

void init_Profibus (void)
{
  unsigned char cnt;
  // Variable init
  // Wait at least Tsyn until allowing RXdata
  profibus_status = PROFIBUS_WAIT_SYN;
  diagnose_status = false;
  Input_Data_size = 0;
  Output_Data_size = 0;
  Vendor_Data_size = 0;
  group = 0;

  slave_addr = SLAVE_ADDRESS;// Temporary address assignment.
  // TODO: Read address from EEPROM or switches.
  // Illegal addresses are forced to DEFAULT (126).
  // Set Address can be used to change it.
  if((slave_addr == 0) || (slave_addr > 126))
    slave_addr = DEFAULT_ADD;

  // Clear data
  #if (OUTPUT_DATA_SIZE > 0)
  for (cnt = 0; cnt < OUTPUT_DATA_SIZE; cnt++)
  {
    Profibus_out_register[cnt] = 0xFF;
  }
  #endif
  #if (INPUT_DATA_SIZE > 0)
  for (cnt = 0; cnt < INPUT_DATA_SIZE; cnt++)
  {
    Profibus_in_register[cnt] = 0x00;
  }
  #endif
  #if (VENDOR_DATA_SIZE > 0)
  for (cnt = 0; cnt < VENDOR_DATA_SIZE; cnt++)
  {
```

```
    Vendor_Data[cnt] = 0x00;
  }
  #endif
  #if (DIAG_DATA_SIZE > 0)
  for (cnt = 0; cnt < DIAG_DATA_SIZE; cnt++)
  {
    Diag_Data[cnt] = 0x00;
  }
  #endif
  new_data=0;
  noInterrupts();           // Disable all interrupts
  init_UART0(BAUD);
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0;
  OCR1A = TIMEOUT_MAX_SYN_TIME;
  TCCR1B |= (1 << WGM12);   // CTC mode
  TCCR1B |= (1 << CS11);    // Prescaler
  TIMSK1 |= (1 << OCIE1A);  // Enable timer compare interrupt
  interrupts();             // Enable all interrupts
}

void profibus_RX (void)
{
  unsigned char cnt;
  unsigned char telegramm_type;
  unsigned char process_data;

  // Profibus data types
  unsigned char destination_add;
  unsigned char source_add;
  unsigned char function_code;
  unsigned char FCS_data;   // Frame Check Sequence
  unsigned char PDU_size;   // PDU Size
  unsigned char DSAP_data;  // SAP Destination
  unsigned char SSAP_data;  // SAP Source

  process_data = false;
  telegramm_type = uart_buffer[0];

  switch (telegramm_type)
  {
    case SD1: // Telegram without data, max. 6 bytes

        if (uart_byte_cnt != 6) break;

        destination_add = uart_buffer[1];
        source_add      = uart_buffer[2];
```

```
        function_code   = uart_buffer[3];
        FCS_data        = uart_buffer[4];

        if (addmatch(destination_add)       == false) break;
        if (checksum(&uart_buffer[1], 3) != FCS_data) break;

        process_data = true;

        break;

    case SD2: // Telegram with variable data length

        if (uart_byte_cnt != uart_buffer[1] + 6) break;

        PDU_size        = uart_buffer[1];
        destination_add = uart_buffer[4];
        source_add      = uart_buffer[5];
        function_code   = uart_buffer[6];
        FCS_data        = uart_buffer[PDU_size + 4];

        if (addmatch(destination_add)              == false) break;
        if (checksum(&uart_buffer[4], PDU_size) != FCS_data) break;

        process_data = true;

        break;

    case SD3: // Telegram with 5 bytes data, max. 11 bytes
        if (uart_byte_cnt != 11) break;
        PDU_size        = 8;              // DA + SA + FC + PDU
        destination_add = uart_buffer[1];
        source_add      = uart_buffer[2];
        function_code   = uart_buffer[3];
        FCS_data        = uart_buffer[9];
        if (addmatch(destination_add)       == false) break;
        if (checksum(&uart_buffer[1], 8) != FCS_data) break;
        process_data = true;
        break;
    case SD4: // Token with 3 Byte Data
        if (uart_byte_cnt != 3) break;
        destination_add = uart_buffer[1];
        source_add      = uart_buffer[2];
        if (addmatch(destination_add)       == false) break;
        break;
    default:
        break;
}
```

```
// Only evaluate if data is OK
if (process_data == true)
{
  master_addr = source_add; // Master address is Source address
  //Service Access Point detected?
  if ((destination_add & 0x80) && (source_add & 0x80))
  {
    DSAP_data = uart_buffer[7];
    SSAP_data = uart_buffer[8];
    // 1) SSAP 62 -> DSAP 60 (Get Diagnostics Request)
    // 2) SSAP 62 -> DSAP 61 (Set Parameters Request)
    // 3) SSAP 62 -> DSAP 62 (Check Config Request)
    // 4) SSAP 62 -> DSAP 60 (Get Diagnostics Request)
    // 5) Data Exchange Request (normal cycle)
    switch (DSAP_data)
    {
    // Set Slave Address (SSAP 62 -> DSAP 55)
      case SAP_SET_SLAVE_ADR:
    // Only possible in the "Wait Parameter" (WPRM) state
          slave_addr = uart_buffer[9];
          profibus_send_CMD(SC, 0, SAP_OFFSET, &uart_buffer[0], 0);
          break;
          // Global Control Request (SSAP 62 -> DSAP 58)
      case SAP_GLOBAL_CONTROL:
          // If "Clear Data" high, then PLC CPU on "Stop"
          if (uart_buffer[9] & CLEAR_DATA_)
          {
            LED_ERROR_ON;  // Status "PLC not ready"
          }
          else
          {
            LED_ERROR_OFF; // Status "PLC OK"
          }

          // Calculate group
          for (cnt = 0;  uart_buffer[10] != 0; cnt++)
          uart_buffer[10]>>=1;

          // If command is for us
          if (cnt == group)
          {
            if (uart_buffer[9] & UNFREEZE_)
            {
              // Delete FREEZE state
            }
            else if (uart_buffer[9] & UNSYNC_)
            {
              // Delete SYNC state
```

```
                }
                else if (uart_buffer[9] & FREEZE_)
                {
                  // Do not read inputs again
                }
                else if (uart_buffer[9] & SYNC_)
                {
                  // Set outputs only with SYNC command
                }
              }

              break;
// Get Diagnostics Request (SSAP 62 -> DSAP 60)
        case SAP_SLAVE_DIAGNOSIS:

    /* After receiving the diagnosis, the DP slave changes state
    "Power on Reset" (POR) in the state "Wait Parameter" (WPRM)
    At the end of initialization ("Data Exchange" state (DXCHG))
    the master sends a Diagnostics Request a second time to
    check correct configuration*/
              if (function_code == (REQUEST_ + FCB_ + SRD_HIGH))
              {
               // Target SAP Master
                uart_buffer[7]  = SSAP_data;
                // Source SAP Slave
                uart_buffer[8]  = DSAP_data;
                // Status 1
                uart_buffer[9]  = STATION_NOT_READY_;
                // Status 2
                uart_buffer[10] = STATUS_2_DEFAULT + PRM_REQ_;
                // Status 3
                uart_buffer[11] = DIAG_SIZE_OK;
                // Address Master
                uart_buffer[12] = MASTER_ADD_DEFAULT;
                // Ident high
                uart_buffer[13] = IDENT_HIGH_BYTE;
                // Ident low
                uart_buffer[14] = IDENT_LOW_BYTE;

                #if (EXT_DIAG_DATA_SIZE > 0)
                 // Device-related diagnosis (number of bytes)
                uart_buffer[15] = EXT_DIAG_DATA_SIZE;
                for (cnt = 0; cnt < EXT_DIAG_DATA_SIZE; cnt++)
                {
                  uart_buffer[16+cnt] = Diag_Data[cnt];
                }
                #endif
                profibus_send_CMD(SD2, DATA_LOW, SAP_OFFSET,
```

```
            &uart_buffer[7], 8 + EXT_DIAG_DATA_SIZE);
          }
          else if (function_code ==
          (REQUEST_ + FCV_ + SRD_HIGH) ||
          function_code ==
          (REQUEST_ + FCV_ + FCB_ + SRD_HIGH))
          {
            // Diagnostic request to check data
            //from Check Config Request
            // Target SAP Master
            uart_buffer[7]  = SSAP_data;
            // Source SAP slave
            uart_buffer[8]  = DSAP_data;
            // Status 1
            if (diagnose_status == true)
              uart_buffer[9]  = EXT_DIAG_;
            else
              uart_buffer[9]  = 0x00;
            // Status 2
            uart_buffer[10] = STATUS_2_DEFAULT;
            // Status 3
            uart_buffer[11] = DIAG_SIZE_OK;
            // Address Master
            uart_buffer[12] = master_addr - SAP_OFFSET;
            // Ident high
            uart_buffer[13] = IDENT_HIGH_BYTE;
            // Ident low
            uart_buffer[14] = IDENT_LOW_BYTE;
            #if (EXT_DIAG_DATA_SIZE > 0)
            // Device-related diagnosis (number of bytes)
            uart_buffer[15] = EXT_DIAG_DATA_SIZE;
            for (cnt = 0; cnt < EXT_DIAG_DATA_SIZE; cnt++)
            {
              uart_buffer[16+cnt] = Diag_Data[cnt];
            }
            #endif

            profibus_send_CMD
            (SD2, DATA_LOW, SAP_OFFSET,
            &uart_buffer[7], 8 + EXT_DIAG_DATA_SIZE);
          }

          break;
// Set Parameters Request (SSAP 62 -> DSAP 61)
        case SAP_SET_PRM:

// After receiving the parameters, the DP slave changes state
// "Wait Parameter" (WPRM) in the state
```

```
// "Wait Configuration" (WCFG)

 // Only accept data for our device
            if ((uart_buffer[13] == IDENT_HIGH_BYTE) &&
            (uart_buffer[14] == IDENT_LOW_BYTE))
            {
// User Parameter Size = Length -
// DA, SA, FC, DSAP, SSAP, 7 Parameter Bytes
              Vendor_Data_size = PDU_size - 12;
// Read in user parameters
              #if (VENDOR_DATA_SIZE > 0)
              for (cnt = 0; cnt < Vendor_Data_size; cnt++)
              Vendor_Data[cnt] = uart_buffer[16+cnt];
              #endif
              // Read group
              for (group = 0; uart_buffer[15] != 0; group++)
              uart_buffer[15]>>=1;
              profibus_send_CMD(SC, 0, SAP_OFFSET,
              &uart_buffer[0], 0);
            }
            break;
// Check Config Request (SSAP 62 -> DSAP 62)
        case SAP_CHK_CFG:

/* After receiving the configuration, the DP slave changes state
"Wait Configuration" (WCFG) in the "Data Exchange" state (DXCHG)
 IO configuration:
 Compact format for max. 16/32 bytes IO
 special format for max. 64/132 bytes IO
 evaluate several bytes depending on the PDU data size
 LE / LEr - (DA + SA + FC + DSAP + SSAP) =
 number of config bytes*/

      Output_Data_size=0;
      Input_Data_size=0;
            for (cnt = 0; cnt < uart_buffer[1] - 5; cnt++)
            {
              switch (uart_buffer[9+cnt] & CFG_DIRECTION_)
              {
                case CFG_INPUT:
                     Input_Data_size +=
                     (uart_buffer[9+cnt] & CFG_BYTE_CNT_) + 1;
                     if (uart_buffer[9+cnt] & CFG_WIDTH_ & CFG_WORD)
                       Input_Data_size += Input_Data_size*2;
                     break;

                case CFG_OUTPUT:
                     Output_Data_size +=
```

```
        (uart_buffer[9+cnt] & CFG_BYTE_CNT_) + 1;
        if (uart_buffer[9+cnt] & CFG_WIDTH_ & CFG_WORD)
          Output_Data_size += Output_Data_size*2;
        break;

    case CFG_INPUT_OUTPUT:
        Input_Data_size +=
        (uart_buffer[9+cnt] & CFG_BYTE_CNT_) + 1;
        Output_Data_size +=
        (uart_buffer[9+cnt] & CFG_BYTE_CNT_) + 1;
        if (uart_buffer[9+cnt] & CFG_WIDTH_ & CFG_WORD)
        {
          Input_Data_size += Input_Data_size*2;
          Output_Data_size += Output_Data_size*2;
        }
        break;

    case CFG_SPECIAL:
        // Special format
        // Manufacturer-specific bytes available?
        if (uart_buffer[9+cnt] & CFG_SP_VENDOR_CNT_)
        {
          // Save the number of manufacturer data
          Vendor_Data_size = uart_buffer[9+cnt] &
         CFG_SP_VENDOR_CNT_;
          // Deduct number of total
          uart_buffer[1] -= Vendor_Data_size;
        }

        // I/O Data
        switch (uart_buffer[9+cnt] & CFG_SP_DIRECTION_)
        {
          case CFG_SP_VOID:
              // Empty data field
              break;

          case CFG_SP_INPUT:
              Input_Data_size +=
              (uart_buffer[10+cnt] & CFG_SP_BYTE_CNT_) + 1;
              if (uart_buffer[10+cnt] & CFG_WIDTH_ & CFG_WORD)
                Input_Data_size += Input_Data_size*2;
              cnt++;  // We already have this byte
              break;

          case CFG_SP_OUTPUT:
              Output_Data_size +=
              (uart_buffer[10+cnt] & CFG_SP_BYTE_CNT_) + 1;
              if (uart_buffer[10+cnt] & CFG_WIDTH_ & CFG_WORD)
```

```
                    Output_Data_size += Output_Data_size*2;
                cnt++;  //We already have this byte
                break;

            case CFG_SP_INPUT_OPTPUT:
                // Erst Ausgang...
                Output_Data_size +=
                (uart_buffer[10+cnt] &
                CFG_SP_BYTE_CNT_) + 1;
                if (uart_buffer[10+cnt] &
                CFG_WIDTH_ & CFG_WORD)
                  Output_Data_size +=
                  Output_Data_size*2;
                // Dann Eingang
                Input_Data_size += (uart_buffer[11+cnt] &
                CFG_SP_BYTE_CNT_) + 1;
                if (uart_buffer[11+cnt] &
                CFG_WIDTH_ & CFG_WORD)
                  Input_Data_size += Input_Data_size*2;
                cnt += 2;  // We already have these bytes
                break;

            } // Switch End
            break;

        default:
            Input_Data_size = 0;
            Output_Data_size = 0;
            break;

      } // Switch End
    } // For End

    if (Vendor_Data_size != 0)
    {
      // Evaluate
    }
    //In case of error -> send CFG_FAULT_ to diagnosis
    // short acknowledgment
    profibus_send_CMD(SC, 0, SAP_OFFSET, &uart_buffer[0], 0);
    break;

  default:
    // Unknown SAP
    break;
 } //Switch DSAP_data end
}
// Destination: Slave address
```

```
else if (destination_add == slave_addr)
{
  // Status query
  if (function_code == (REQUEST_ + FDL_STATUS))
  {
    profibus_send_CMD(SD1, FDL_STATUS_OK, 0, &uart_buffer[0], 0);
  }
  // Master sends output data and
  // requests input data(Send and Request Data)
  else if (function_code == (REQUEST_ + FCV_ + SRD_HIGH) ||
           function_code == (REQUEST_ + FCV_ + FCB_ + SRD_HIGH))
  {

    // Read data from master
    #if (INPUT_DATA_SIZE > 0)
    for (cnt = 0; cnt < INPUT_DATA_SIZE; cnt++)
    {
      Profibus_in_register[cnt] = uart_buffer[cnt + 7];
  new_data=1;
    }
    #endif


    // Write data for master in buffer
    #if (OUTPUT_DATA_SIZE > 0)
    for (cnt = 0; cnt < OUTPUT_DATA_SIZE; cnt++)
    {
      uart_buffer[cnt + 7] = Profibus_out_register[cnt];
    }
    #endif


    #if (OUTPUT_DATA_SIZE > 0)
    if (diagnose_status == true)
    // Request a diagnosis
      profibus_send_CMD(SD2, DIAGNOSE, 0, &uart_buffer[7], 0);
    else
    // send data
      profibus_send_CMD(SD2, DATA_LOW, 0,
      &uart_buffer[7], Input_Data_size);
    #else
    if (diagnose_status == true)
    // Request a diagnosis
      profibus_send_CMD(SD1, DIAGNOSE, 0, &uart_buffer[7], 0);
    else
     // short acknowledgment
      profibus_send_CMD(SC, 0, 0, &uart_buffer[7], 0);
    #endif
```

```
      }
    }

  } //Data valid at the end

}

////////////////////////////////////////////////////////
/*!
  Compile and send * \ brief Profibus telegram
  * \ param type telegram type (SD1, SD2 etc.)
  * \ param function_code Function code to be transmitted
  * \ param sap_offset Value of the SAP offset
  * \ param pdu pointer to data field (PDU)
  * \ param length_pdu Length of pure PDU without DA, SA or FC
  */
void profibus_send_CMD (unsigned char type,
                        unsigned char function_code,
                        unsigned char sap_offset,
                        char *pdu,
                        unsigned char length_pdu)
{
  unsigned char length_data;


  switch(type)
  {
    case SD1:
      uart_buffer[0] = SD1;
      uart_buffer[1] = master_addr;
      uart_buffer[2] = slave_addr + sap_offset;
      uart_buffer[3] = function_code;
      uart_buffer[4] = checksum(&uart_buffer[1], 3);
      uart_buffer[5] = ED;
      length_data = 6;
      break;

    case SD2:
      uart_buffer[0] = SD2;
      // Length of the PDU incl. DA, SA and FC
      uart_buffer[1] = length_pdu + 3;
      uart_buffer[2] = length_pdu + 3;
      uart_buffer[3] = SD2;
      uart_buffer[4] = master_addr;
      uart_buffer[5] = slave_addr + sap_offset;
      uart_buffer[6] = function_code;
      //Data is already filled in before the function is called
      uart_buffer[7+length_pdu] =
```

```
      checksum(&uart_buffer[4], length_pdu + 3);
      uart_buffer[8+length_pdu] = ED;
      length_data = length_pdu + 9;
      break;

    case SD3:
      uart_buffer[0] = SD3;
      uart_buffer[1] = master_addr;
      uart_buffer[2] = slave_addr + sap_offset;
      uart_buffer[3] = function_code;
      // Data is already filled in before the function is called
      uart_buffer[9] = checksum(&uart_buffer[4], 8);
      uart_buffer[10] = ED;
      length_data = 11;
      break;

    case SD4:
      uart_buffer[0] = SD4;
      uart_buffer[1] = master_addr;
      uart_buffer[2] = slave_addr + sap_offset;
      length_data = 3;
      break;

    case SC:
      uart_buffer[0] = SC;
      length_data = 1;
      break;

    default:
      break;
  }
  profibus_TX(&uart_buffer[0], length_data);

}
////////////////////////////////////////////////////
/*!
  Send * \ letter telegram
  * \ param data pointer to data field
  * \ param length Length of the data
  */
void profibus_TX (char *data, unsigned char length)
{
  TX_ENABLE_ON;  // Enable Transmit (Switch to Transmit)
  OCR1A = TIMEOUT_MAX_TX_TIME;
  profibus_status = PROFIBUS_SEND_DATA;

  uart_byte_cnt = length;          // Number of bytes to send
  uart_transmit_cnt = 0;           // Payer for sent bytes
```

```
  UCSR0B |= _BV(UDRIE0);
}
////////////////////////////////////

unsigned char checksum(char *data, unsigned char length)
{
  unsigned char csum = 0;

  while(length--)
  {
    csum += data[length];
  }

  return csum;
}
////////////////////////////////////////
/*!
  Check the letter destination. Address must be with
  slave address or broadcast (including SAP offset)
          to match

  */
unsigned char addmatch (unsigned char destination)
{
  if ((destination != slave_addr) &&                 // Slave
      (destination != slave_addr + SAP_OFFSET) &&    // SAP Slave
      (destination != BROADCAST_ADD) &&              // Broadcast
      (destination != BROADCAST_ADD + SAP_OFFSET)){  // SAP Broadcast
        return false;
      }
  return true;
}
////////////////////////////////////////////////


ISR (TIMER1_COMPA_vect)     // Timer1 Output Compare 1A.
{
  TIMER1_STOP;  // Guard ourselves.
  switch (profibus_status)
  {
    case PROFIBUS_WAIT_SYN: // TSYN expired
        profibus_status = PROFIBUS_WAIT_DATA;
        OCR1A = TIMEOUT_MAX_SDR_TIME;
        uart_byte_cnt = 0;
        break;

    case PROFIBUS_WAIT_DATA:  // TSDR expired but no data there
        break;
```

```
    case PROFIBUS_GET_DATA:    // TSDR expired and data there
        profibus_status = PROFIBUS_WAIT_SYN;
// We have already waited TIMEOUT_MAX_RX_TIME of bus idle.
        OCR1A = TIMEOUT_MAX_SYN_TIME-TIMEOUT_MAX_RX_TIME;
        profibus_RX();
        break;
// Transmission timeout expired, go back to reception
    case PROFIBUS_SEND_DATA:
        profibus_status = PROFIBUS_WAIT_SYN;
        OCR1A = TIMEOUT_MAX_SYN_TIME;
        TX_ENABLE_OFF;  // Disable Transmit (Switch to Recieve)
        break;

    default:
        break;

  }
  // Timer 1 Start
  TIMER1_RUN;
  TCNT1 = 0;
}

/*!
 * ISR UART0 Receive
 */
ISR(USART_RX_vect)
{
  // Retrieve RXdata to buffer
  uart_buffer[uart_byte_cnt] = UDR0;
  // Only read data after Tsyn have expired
  if (profibus_status == PROFIBUS_WAIT_DATA)
  {
    profibus_status = PROFIBUS_GET_DATA;
  }

  // Read data allowed?
  if (profibus_status == PROFIBUS_GET_DATA)
  {
    uart_byte_cnt++;
    // Check for buffer overflow!
    if(uart_byte_cnt >= MAX_BUFFER_SIZE) uart_byte_cnt--;
  }
  TCNT1 = 0;
}
////////////////////////////////////////////////////////////////////////
/*!
 * ISR UART Transmit
```

```
 */
ISR(USART_UDRE_vect)
{
  // Everything sent?
  if (uart_transmit_cnt < uart_byte_cnt)
  {
    // TX Buffer
    UDR0 = uart_buffer[uart_transmit_cnt++];
  }
  else
  {
    // All sent, interrupt again
    UCSR0B &= ~( 1 << UDRIE0 );
  }
  TCNT1=0;
}
//////////////////////////////////////////////////
/*!
 * Initialize UART0
 * Even parity, 1 stop bit.
 */
/* */
void init_UART0(unsigned long baud)
{
  // Try U2X mode first
  uint16_t baud_setting = (F_CPU / 4 / baud - 1) / 2;
  UCSR0A = 1 << U2X0;
  if (((F_CPU == 16000000UL) && (baud == 57600)) || (baud_setting >4095))
  {
    UCSR0A = 0;
    baud_setting = (F_CPU / 8 / baud - 1) / 2;
  }
  // assign the baud_setting, a.k.a. ubrr (USART Baud Rate Register)
  UBRR0H = baud_setting >> 8;
  UBRR0L = baud_setting;
  UCSR0B |= ( 1 << RXEN0 ) | ( 1 << TXEN0 );
  UCSR0B |= ( 1 << RXCIE0 ) | ( 1 << UDRIE0 );
  //set the data bits, parity, and stop bits
  UCSR0C = 0x26; // 8 bits, EVEN parity and 1 Stop bit - 8E1
}
```

CAPÍTULO 12

Manual

Introducción y características principales

La Serie SMARTSTEP 2 ofrece servo controlador diseñados para sistemas de posicionamiento de baja capacidad. A pesar de su tamaño compacto, estas unidades cuentan con funciones avanzadas como ajuste automático en tiempo real y filtros adaptativos que simplifican los ajustes de ganancia. Además, incluyen un filtro de muesca para reducir la vibración mecánica durante la operación, lo que asegura un rendimiento estable en mecanismos con baja rigidez de carga. Comparada con su predecesora, la Serie SMARTSTEP A, la Serie SMARTSTEP 2 logra una reducción significativa del espacio de instalación y del tamaño, lo que la hace ideal para aplicaciones que requieren posicionamiento preciso en espacios reducidos. Una característica destacada es su capacidad para suprimir la vibración en mecanismos con baja rigidez, lo cual es crucial durante la aceleración y desaceleración. También ofrece control de resonancia para lograr posicionamiento de alta velocidad y es compatible con una amplia gama de entradas de pulsos, lo que simplifica el control de sincronización. La Serie SMARTSTEP 2 proporciona una variedad de funciones de configuración de pulsos, como multiplicación de pulsos de comando y división de codificadores, adaptándose a las necesidades específicas de cada dispositivo o sistema. Además, facilita el control de velocidad con ajustes internos y permite la personalización del número de pulsos de codificador emitidos por el servo controlador.

Especificaciones generales de servo controlador

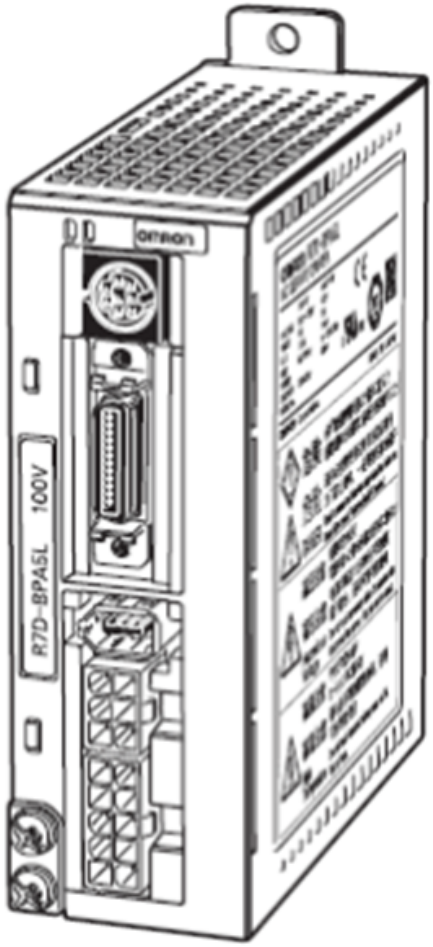


Figura 1: Servo controlador R7D-BP04H

Elementos	Especificación
Temperatura ambiente de funcionamiento	0 a 55°C
Humedad ambiente de funcionamiento	90 % máx. (sin condensación)
Temperatura ambiente de almacenamiento	–20 a 65°C
Humedad ambiente de almacenamiento	90 % máx. (sin condensación)
Atmósfera de almacenamiento/operación	Sin gases corrosivos.
Resistencia a vibraciones	10 a 60 Hz; aceleración: 5,9 m/s^2 (0,6 G) máx
Resistencia a golpes	Aceleración máx. 19,6 m/s^2, 3 veces en las direcciones X, Y y Z
Resistencia de aislamiento	Entre terminales de alimentación/línea de alimentación y marco de tierra: 0,5 M mín. (a 500 Vc.c.)
Rigidez dieléctrica	Entre fuente de alimentación/terminales de alimentación y marco de tierra: 1.500 Vc.a. a 50/60 Hz durante 1 minuto Entre cada señal de control y el marco de tierra: 500 Vc.a. durante 1 minuto
Grado de protección	Incorporado en el panel (IP10).
Normas internacionales	Aprobación CE: EMC EN55011 clase A grupo 1, EN 61000-6-2, baja tensión EN50178

Cuadro 1: Especificaciones generales

Indicador LED de alimentación (PWR)

Estado de la LED	Descripción
Verde	Fuente de poder primaria encendida
Parpadeo naranja a intervalos de 1s	Se ha producido una advertencia
Roja	Una alarma en sucedido

Cuadro 2: Estado de LED

Indicador LED de alarma (ALM)

Este indicador se ilumina cuando se ha producido una alarma. El número de destellos naranjas y rojos indican el código de la alarma. Para más información recurrir al manual del proveedor en la página 8.

Parámetros básicos para empezar a trabajar

El servo controlador tiene por defecto la mayoría de entradas deshabilitadas excepto la entra

Parámetros de funciones de selección (Pn02)

El servo SS2 dispone de dos métodos de control de posición: 1) Control de posición de alta velocidad y 2) Control de posición de alta funcionalidad, seleccionables a través de este parámetro, según la siguiente tabla:

	Valores posibles	Explicación	Valor inicial
Pn02	0	Control de posición con respuesta de alta velocidad	2
	1	Control de velocidad	
	2	Control de posición de alta funcionalidad	

Figura 2: Descripción de opciones para el parámetro Pn02

1) Control de posición de alta funcionalidad: este método de control ofrece la mayor funcionalidad a nivel de control, pero asume que la mecánica es la correcta: alta rigidez, ausencia de holguras, no resonancias, etc.

2) Control de posición con respuesta de alta velocidad: este método fue ideado para la realización de un buen control de motor "independientemente" de la mecánica. Es decir que, a través de la electrónica, se resuelven las limitaciones mecánicas. Este es el motivo de que sea posible habilitar todos los algoritmos de control.

Selección de límites de movimiento POT/NOT (Pn04)

Permite habilitar los límites de movimiento POT/NOT para definir un área de operación del sistema entre dichos límites. Cuando se activa el límite en forward (POT) no se permite el movimiento en este sentido, pero sí en reversa y viceversa. Por defecto, estas entradas están deshabilitadas, por lo que la operación se realiza independientemente del estado de estas.

	Valores posibles	Explicación	Valor inicial
Pn04	0	Habilitados	1
	1	Deshabilitados	

Figura 3: Descripción de opciones para el parámetro Pn04

Se dispone de los siguientes métodos de frenado ante la activación (entradas a OFF) de las señales de POT/NOT seleccionables según el parámetro Pn66:

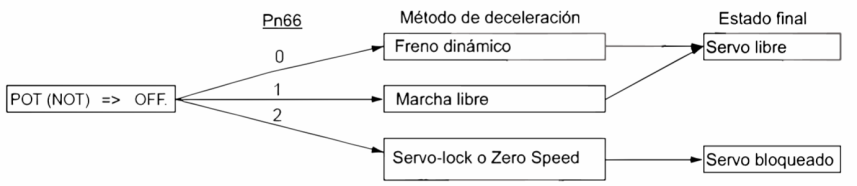


Figura 4: Descripción de opciones para el parámetro Pn66

Selección Zero Speed/Conmutación límite de par (Pn06)

En función del modo de control en el que se esté trabajando, este parámetro permite habilitar y deshabilitar dos funciones diferentes:

- Selección de Zero Speed (VZERO): esta función permite generar una consigna de velocidad cero al servo trabajando en control de velocidad.
- Conmutación de límite de par (TLSEL): esta función permite seleccionar entre dos juegos de parámetros (Pn70, Pn5E, Pn63 ó Pn71, Pn72, Pn73) que se corresponden respectivamente con Sobre velocidad/Límite de par/Contador de error. Esta función se puede utilizar en control de velocidad o de posición indistintamente.

Si se trabaja en control de velocidad, se pueden emplear ambas funciones, mientras que si se trabaja en control de posición se puede utilizar la función de conmutación de límite de par (TLSEL). En la siguiente tabla se muestran las combinaciones posibles:

	Modo de control	Valores posibles	Explicación	Valor inicial
Pn06	Pn02=1	1	Función VZERO habilitada	1
		2	Función VZERO deshabilitada	
	Pn02=0, 1, 2	0	Función TLSEL deshabilitada	
		2	Función TLSEL habilitada	

Figura 5: Descripción de opciones para el parámetro Pn06

Multiplicador de referencia de pulsos

Permite aplicar el factor de multiplicación x2, x4 al tren de pulsos de entrada cuando se selecciona como tipo de entrada "fase diferencial 90º" en el parámetro Pn42 = 0, 2.

	Valores posibles	Explicación
Pn40	1	Multiplicador x2
	2	
	3	Multiplicador x4
	4	

Figura 6: Descripción de opciones para el parámetro Pn40

Conmutación de sentido de giro (Pn41)

Permite invertir el sentido de giro del motor con respecto al indicado en la consigna de pulsos.

	Valores posibles	Explicación
Pn41	0	El servomotor gira según el sentido indicado en la referencia de pulsos
	1	El servomotor gira en sentido opuesto al indicado por la referencia de pulsos
	2	
	3	El servomotor gira según el sentido indicado en la referencia de pulsos

Figura 7: Descripción de opciones para el parámetro Pn41

Conmutación de sentido de giro (Pn42)

Permite seleccionar el formato de entrada de la referencia de pulsos al controlador trabajando en control de posición.

	Valores posibles	Explicación
Pn42	0, 2	Entrada fase diferencial 90º (A, B)
	1	Entrada de pulsos forward + entrada de pulsos reverse
	3	Entrada de pulsos + sentido

Figura 8: Descripción de opciones para el parámetro Pn42

Especificaciones generales del servomotor

Figura 9: Servo motor R88M-G20030H marca Omron

Tensiones aplicadas	Magnitud	Valor
Salida nominal	W	200
Par nominal	Nm	0.64
Par máximo instantáneo	Nm	1.78
Corriente nominal	A	1.6
Corriente máx. instantánea	A	4.9
Velocidad nominal	mín.–1	3
Velocidad máx.	rpm	3
Momento de inercia del rotor	kg·m2x10–4	0.14
Momento de inercia de la carga admisible	JL	30
Relación de potencia nominal	kW/s	30.3

Cuadro 3: Especificaciones de servomotor R88M-G20030H cilíndricos

Partes y funcionamientos

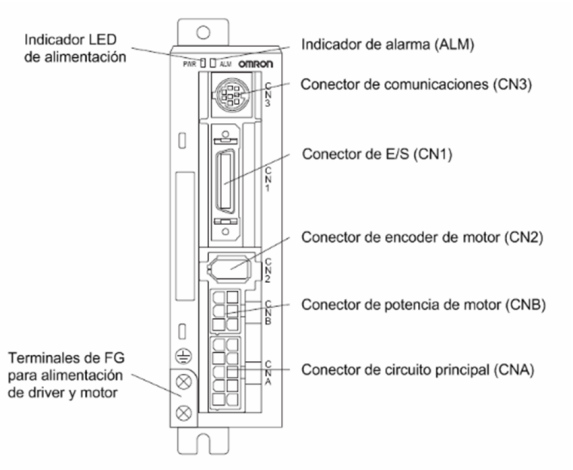


Figura 10: Nombre de puertos del servo controlador

Instalación

Conexión entre el servo controlador y el servo motor

La conexión entre estos dispositivos son dos cables CN2 y CNB, los cuales van directamente conectados al último cable del servo motor y al penúltimo cable respectivamente.

Conexión entre el circuito de acoplamiento y el Servo Controlador

El servo controlador utiliza como medio el cable XW2Z-100J-B28, que permite la conexión del puerto CN1 en el servo controlador con el módulo de separación XW2B-34G5. El módulo XW2B-34G5 facilita el acceso a las señales de las entradas y salidas del puerto CN1 para el circuito de acople.

Conexión entre el circuito de acoplamiento y el PLC

El PLC posee un puerto de conexión de PROFIBUS DP (Figura 11) para la conexión del cable que actúa como medio entre el PLC y el circuito de acoplamiento (la conexión con la placa del Arduino se haces conectado el cable verde con la entrada +B y el cable rojo con la entrada +A Figura 12). Adicionalmente se utiliza los 24V y la referencia del PLC para poder alimentar las entradas del servo controlador por medio del circuito de acoplamiento. Finalmente, se utiliza la conexión trifásica la cual se hace por medio del contactor que conecta al variador para la alimentación del servo controlador y el servo motor.

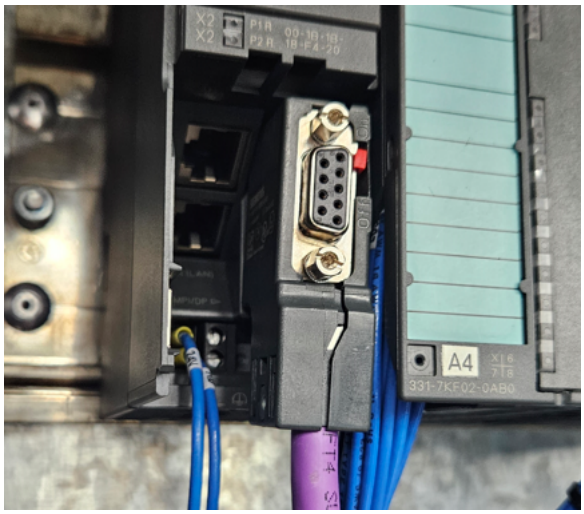


Figura 11: Puerto PROFIBUS-DP en el PLC

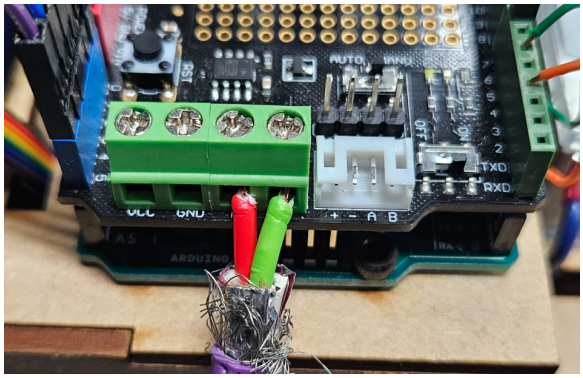


Figura 12: Conexión de cable PROFIBUS al Arduino UNO

Funcionamiento del circuito de acoplamiento

Para el funcionamiento óptimo del circuito de acople, todos los pasos en esta sección son de suma importancia.

Configuración de circuito de acoplamiento en step 7

Instalación de GSD

Los pasos para la instalación del archivo .GSD son los siguientes:

1. Abra la configuración de hardware.
2. Cierre el proyecto abierto en él.
3. Seleccione el comando del menú “Opciones > Instalar archivo GSD...”.
4. En el siguiente diálogo, busque en el directorio del archivo donde haya almacenado los archivos GSD. Puede elegir si desea buscar los archivos GSD en el directorio de archivos o en un proyecto STEP 7.
5. Luego, haga clic en “Instalar”.

   Nota: Durante el proceso de instalación, es posible que reciba mensajes de soporte. Por ejemplo, que un archivo GSD específico ya está instalado. Cuando la instalación esté completa, puede mostrar un registro.
6. Luego, actualice su catálogo de hardware utilizando el comando del menú “Opciones > Actualizar catálogo”.
7. Luego, abra nuevamente la configuración de hardware.

   Nota: Los esclavos se leen en el catálogo de módulos, cuyos archivos GSD fueron instalados después del inicio de STEP 7. Luego, puede incorporar los esclavos en su proyecto STEP 7. Los dispositivos recién instalados están en el Catálogo de Hardware. En este ejemplo para PROFIBUS DP y para PROFINET IO bajo .Otros dispositivos de campo".
8. Abra el esclavo configurado en la configuración de hardware. Nota: Puede encontrar el nombre del archivo GSD en las propiedades en la pestaña "General"junto al número de pedido.

STEP 7 almacena los archivos GSD en la unidad donde instaló STEP 7. Encontrará esto en el directorio “Archivos de programa > Siemens > STEP 7, S7DATA, GSD”.

Se adjunta hipervínculo con el fin de poder llevar este proceso de manera más visual:Link

Configuración de comunicación PROFIBUS DP

Posterior a la instalación del archivo .GSD, se puede encontrar en la parte derecha de la ventana de configuración de HARDWARE, que normalmente se encuentra en la carpeta titulada “GENERAL”, bajo el nombre “AVR software”.

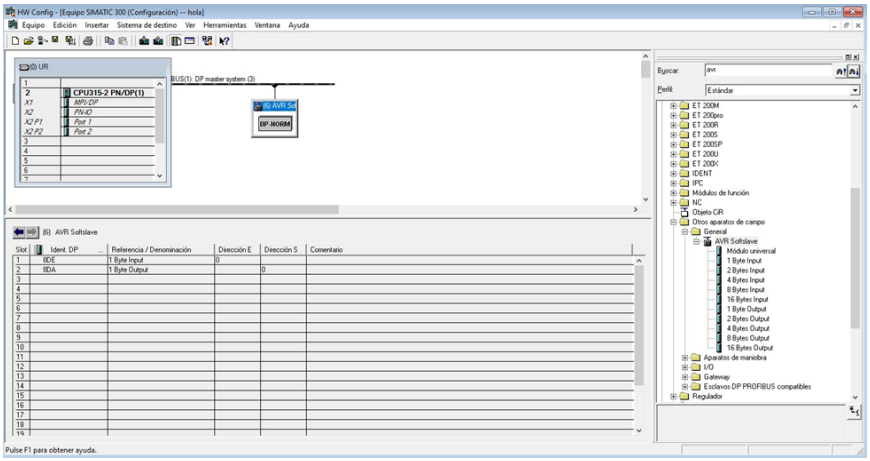


Figura 13: Configuración de hardware

Adicionalmente, se selecciona la cantidad de bits/bytes de inputs/outputs deseados para la comunicación de PRUFIBUS DP y despliega la dirección donde debemos enviar la data. Finalmente, para el envío de datos se utiliza la función “MOVE” que traslada la información la data de una locación de memoria a la dirección de envío por medio de PROFIBUS.

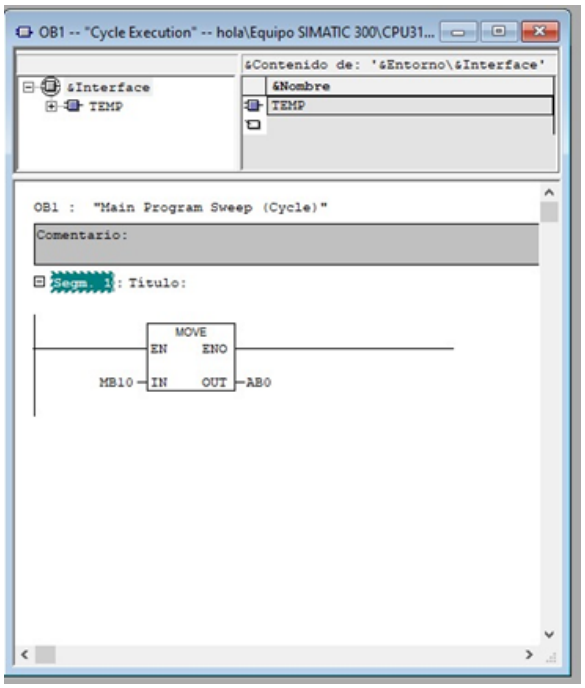


Figura 14: Lógica de envío de datos en simatic manager

Comandos de funcionamiento del circuito de acoplamiento

El circuito de acoplamiento fue diseñado para el uso versátil del servo controlador, debido a que cada entrada del puerto CN1 depende de la activación del parámetro respectivo. Por otro lado, el accionamiento de cada entrada es de forma individual.

Bits de comunicacion	Puerto interno Arduino	Conexión al puerto CN1
Bit No. 0	A4	RUN
Bit No. 1	Cambio de frecuencia para CCW	
Bit No. 2	Cambio de dirección de Giro	
Bit No. 3	A2	ECRST & VSEL2
Bit No. 4	A1	GSEL & VZERO & TLSEL
Bit No. 5	A0	GSEL & VSEL1
Bit No. 6	7	NOT
Bit No. 7	8	POT

Cuadro 4: Significado de Bits de Implementación

Funcionamiento CX Drive

CX Drive es el programa de conexión y configuración de parámetros específico de Omron para gestionar el Servo Controlador "R7D-BP04H". La Figura 15 es la imagen inicial del programa. Por otro, CX Drive con el comando F1, el programa despliega un desglose general del uso del programa.

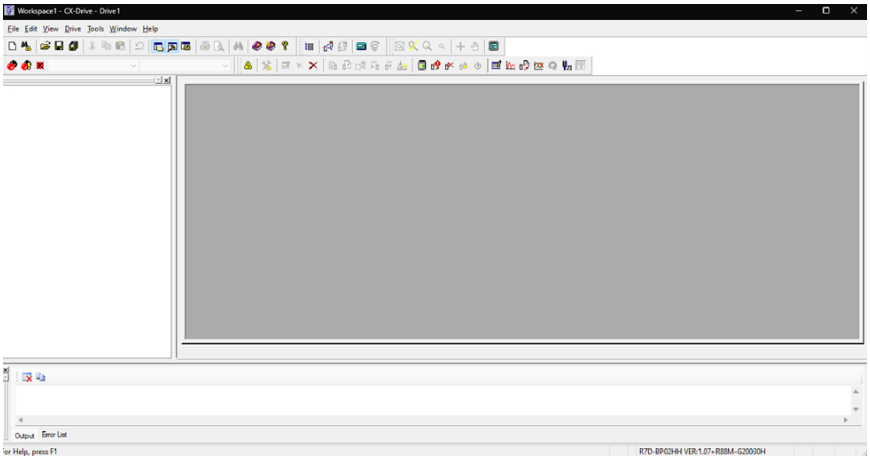


Figura 15: Pantalla de inicio de CX Drive

Para conectarse al servo controlador debe buscar en "*device manager*" el puerto de comunicación donde el servo controlador está conectado, luego se va a el menú de "*Driver*" de la aplicación y luego a "*Change...*" para poder colocar el puerto de comunicación como en la Figura 16

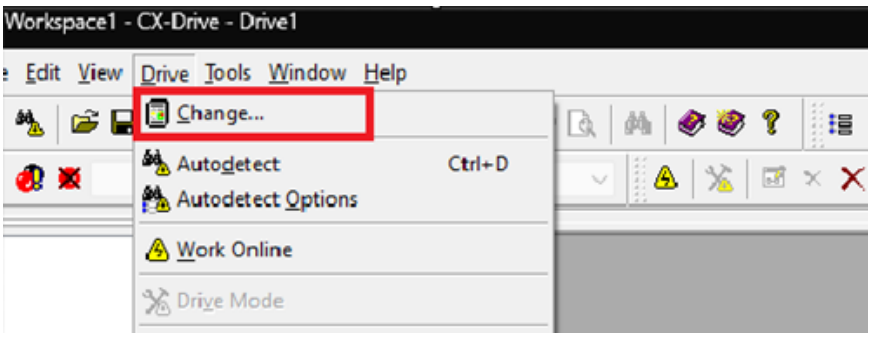


Figura 16: Paso 1 de configuración del puerto COM

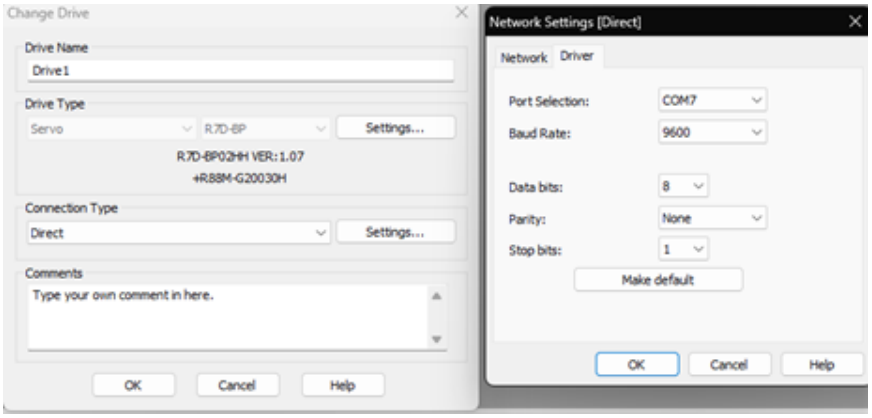


Figura 17: Paso 2 de configuración del puerto COM

En el parámetro de "*Driver Type*" se inserta la información del servo a utilizar y en la parte de "Connection type" se coloca el puerto de comunicación obtenido del "*device manager*".

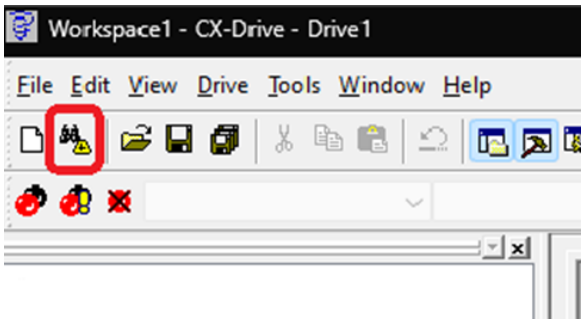


Figura 18: Paso 1 de configuración de autodetección

La función del programa "*Autodetect*" debe ser seleccionada para desplegar el menú en Figura 19

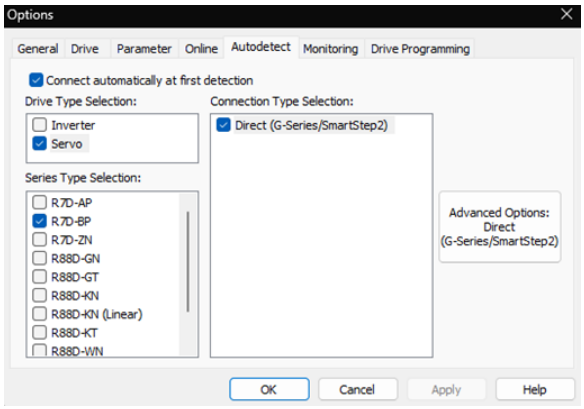


Figura 19: Paso 2 de configuración de autodetección

En esta ventana se selecciona nuevamente la información del servo al que se busca conectar, seguidamente de aceptar y seleccionar "*ok*". Consecuentemente, la ventana de la Figura 20 se despliega, y posteriormente de seleccionar "*Start*". Como resultado, el programa inicializará la conexión entre el programa y el servo.

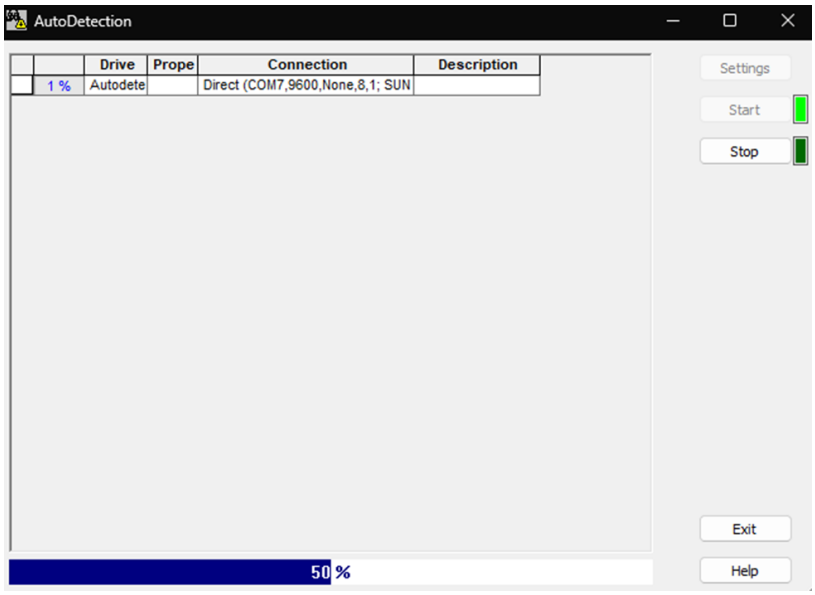


Figura 20: Paso 3 de configuración de autodetección

En el momento que esta conexión sea establecida, en la parte izquierda de la ventana tendremos algo similar a la Figura 21 .

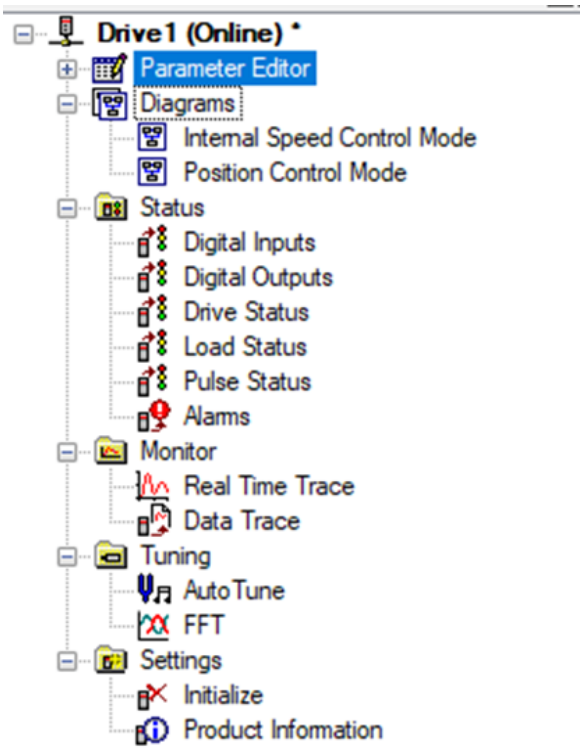


Figura 21: Opciones y funciones del servo controlador en CX Drive

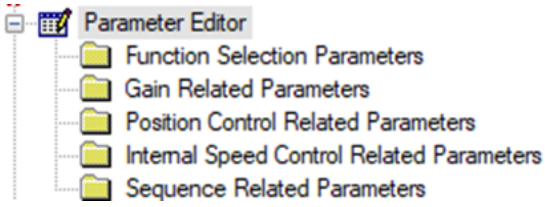


Figura 22: Contenido de editor de parámetro

"Parameter Editor" despliega un menú de las posibles opciones en las cuales el servo controlador puede operar. Dentro de cada carpeta podemos encontrar los parámetros necesarios para usar el modo de operación decido.

...	Index	Description	Value	Drive Value	Default	Range	Units	Rest...
	Pn00	Unit No. Setting	1	---	1	1 to 15		☑
	Pn01	Default Display	1: Servomotor Rotatio...	---	1	0 to 15		☑
	Pn02	Control Mode Selection	2: High function positi...	---	2	0 to 2		☑
	Pn04	Drive Prohibit Input Selection	1: Disabled	---	1	0 to 1		☑
	Pn06	Zero Speed Designation/Torque Limit Switch	1: Zero speed designa...	---	1	0 to 2		☑
	Pn09	Warning Output Selection	2: Over regeneration/...	---	2	0 to 6		☐

Figura 23: Parámetros relacionados con el funcionamiento

...	Index	Description	Value	Drive Value	Default	Range	Units	Rest...
	Pn10	Position Loop Gain (RT)	40	---	40	0 to 32767	x 1/s	
	Pn11	Speed Loop Gain (RT)	60	---	60	1 to 3500	Hz	
	Pn12	Speed Loop Integration Time Constant (RT)	20	---	20	1 to 1000	ms	
	Pn13	Speed Feedback Filter Time Constant (RT)	0	---	0	0 to 5		
	Pn14	Torque Command Filter Time Constant (RT)	1.00	---	1.00	0.00 to 25...	ms	
	Pn15	Speed Feed-forward Amount (RT)	30.0	---	30.0	-200.0 to ...	%	
	Pn16	Feed-forward Filter Time Command (RT)	1.00	---	1.00	0.00 to 64...	ms	
	Pn18	Position Loop Gain 2 (RT)	20	---	20	0 to 32767	x 1/s	
	Pn19	Speed Loop Gain 2 (RT)	80	---	80	1 to 3500	Hz	
	Pn1A	Speed Loop Integration Constant 2 (RT)	50	---	50	1 to 1000	ms	
	Pn1B	Speed Feedback Filter Time Constant 2 (RT)	0	---	0	0 to 5		
	Pn1C	Torque Command Filter Time Constant 2 (RT)	1.00	---	1.00	0.00 to 25...	ms	
	Pn1D	Notch Filter 1 Frequency	1500	---	1500	100 to 1500	Hz	
	Pn1E	Notch Filter 1 Width	2: 0.71	---	2	0 to 4		
	Pn20	Inertia Ratio (RT)	300	---	300	0 to 10000	%	
	Pn21	Realtime Autotuning Mode Selection	0: Disabled. Adaptive ...	---	0	0 to 7		
	Pn22	Realtime Autotuning Machine Rigidity Selection	2	---	2	0 to 15		
	Pn25	Autotuning Operation Setting	0: Forward to reverse,...	---	0	0 to 7		
	Pn26	Overrun Limit Setting	1.0	---	1.0	0.0 to 100.0	Rotation	
	Pn2B	Vibration Frequency	0.0	---	0.0	0.0 to 500.0	Hz	
	Pn2C	Vibration Filter Setting	0.0	---	0.0	-20.0 to 2...	Hz	
	Pn2F	Adaptive Filter Table Number	0: Notch Filter 1 disabled	---	0	0 to 64		
	Pn30	Gain Switching Input Operating Mode Select...	1: Disabled	---	1	0 to 1		
	Pn31	Gain Switch Setting	0: Always gain 1	---	0	0 to 10		
	Pn32	Gain Switch Time	30	---	30	0 to 10000	x 166 micro s	
	Pn33	Gain Switch Level Setting	600	---	600	0 to 20000		
	Pn34	Gain Switch Hysteresis Setting	50	---	50	0 to 20000		
	Pn35	Position Loop Gain Switching Time	20	---	20	0 to 10000	x 166 micro s	

Figura 24: Parámetros relacionados con la ganancia

...	Index	Description	Value	Drive Value	Default	Range	Units	Rest...
	Pn40	Command Pulse Multiplier Setting	4: Multiplier 4	---	4	1 to 4		☑
	Pn41	Command Pulse Rotation Direction Switch	0: Rotates in a directio...	---	0	0 to 3		☑
	Pn42	Command Pulse Mode	3: Feed pulses and for...	---	1	0 to 3		☑
	Pn44	Encoder Divider Setting	2500	---	2500	1 to 16384	Pulse(s)	☑
	Pn45	Encoder Output Direction Switch	0: Not reversed	---	0	0 to 1		☑
	Pn46	Electronic Gear Ratio Numerator 1	10000	---	10000	1 to 10000		☐
	Pn47	Electronic Gear Ratio Numerator 2	10000	---	10000	1 to 10000		☐
	Pn4A	Electronic Gear Ratio Numerator Exponent	0	---	0	0 to 17		☐
	Pn4B	Electronic Gear Ratio Denominator	2500	---	2500	1 to 10000		☐
	Pn4C	Position Command Filter Time Constant Set...	0	---	0	0 to 7		☐
	Pn4E	Smoothing Filter Setting	0	---	0	0 to 31		☑

Figura 25: Parámetros relacionados con el control de posición

...	Index	Description	Value	Drive Value	Default	Range	Units	Rest...
	Pn53	No. 1 Internal Speed Setting	100	---	100	-6000 to 6...	r/min	☐
	Pn54	No. 2 Internal Speed Setting	200	---	200	-6000 to 6...	r/min	☐
	Pn55	No. 3 Internal Speed Setting	300	---	300	-6000 to 6...	r/min	☐
	Pn56	No. 4 Internal Speed Setting	400	---	400	-6000 to 6...	r/min	☐
	Pn57	Jog Speed	200	---	200	0 to 500	r/min	☐
	Pn58	Soft Start Acceleration Time	0	---	0	0 to 5000	x 2 ms/(100...	☐
	Pn59	Soft Start Deceleration Time	0	---	0	0 to 5000	x 2 ms/(100...	☐
	Pn5E	Torque Limit	300	---	300	0 to 300	%	☐

Figura 26: Parámetros relacionados con velocidades internas del servo motor

...	Index	Description	Value	Drive Value	Default	Range	Units	Rest...
	Pn60	Positioning Completion Range 1	25	---	25	0 to 32767	Pulse(s)	☐
	Pn61	Zero Speed Detection	20	---	20	0 to 20000	r/min	☐
	Pn62	Rotation Speed for Motor Rotation Detection	50	---	50	0 to 20000	r/min	☐
	Pn63	Deviation Counter Overflow Level	100	---	100	1 to 32767	x 256 pulses	☐
	Pn64	Deviation Counter Overflow Alarm Invalidation	0: Enabled	---	0	0 to 1		☐
	Pn66	Stop Selection for Drive Prohibition Input	0: Invalidate torque, a...	---	0	0 to 2		☑
	Pn68	Stop Selection for Alarm Generation	0: Decel: Dynamic brak...	---	0	0 to 3		☐
	Pn69	Stop Selection with Servo OFF	0: Decel: Dynamic brak...	---	0	0 to 7		☐
	Pn6A	Brake Timing when Stopped	10	---	10	0 to 100	x 2 ms	☐
	Pn6B	Brake Timing during Operation	50	---	50	0 to 100	x 2 ms	☐
	Pn6C	Regeneration Resistor Selection	0: Built-in condenser h...	---	0	0 to 3		☑
	Pn70	Overspeed Detection Level Setting	0	---	0	0 to 6000	r/min	☐
	Pn71	No. 2 Torque Limit	100	---	100	0 to 300	%	☐
	Pn72	No. 2 Deviation Counter Overflow Level	100	---	100	1 to 32767	x 256 pulses	☐
	Pn73	No. 2 Overspeed Detection Level Setting	0	---	0	0 to 6000	r/min	☐

Figura 27: Parámetros secuenciales

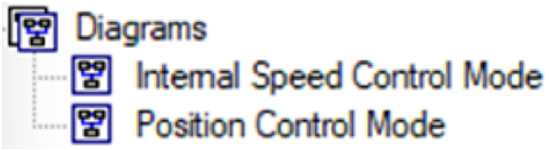


Figura 28: Contenido de carpeta de diagrama

Los parámetros pueden mostrarse en diagramas de bloques para los parámetros relevantes. Los inversores admiten diagramas de bloques PID, y los servomotores admiten diagramas de bloques de posición, velocidad y par.

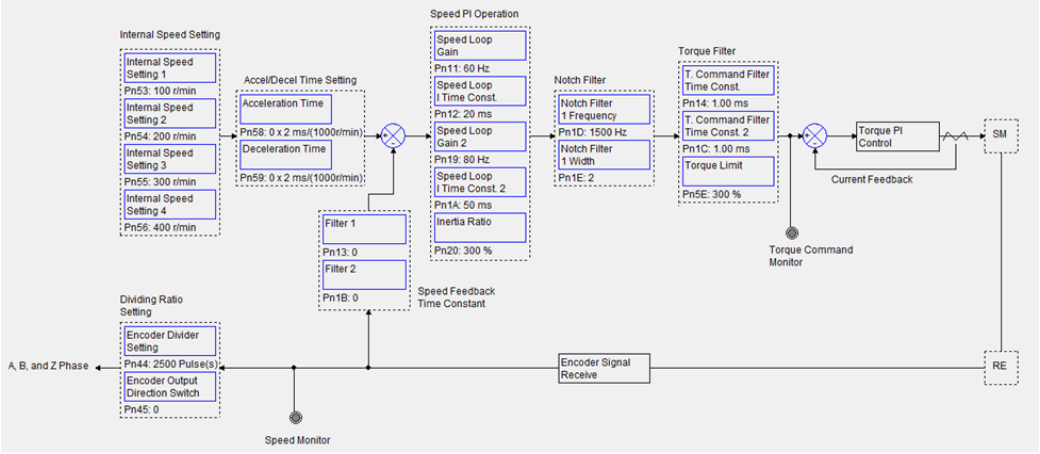


Figura 29: *Internal speed control mode*

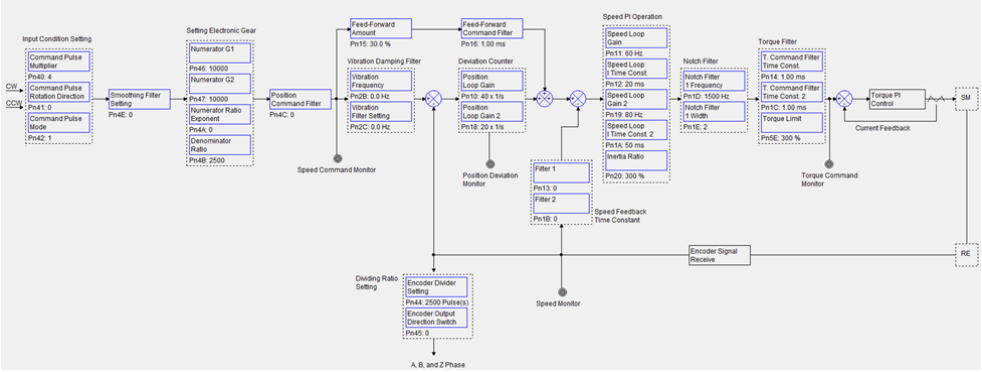


Figura 30: *Position control mode*

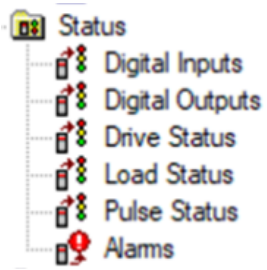


Figura 31: Contenido de carpeta de estado

El ícono con la carpeta que se titula "*Status*" en la Figura 31 , despliega información de las entradas, salida, el estado del servo controlador, la carga del servo motor las pulsaciones y ,por último, las alarmas.

	Index	Description	Value
☑	ECRST	Counter Cleared	OFF
☑	GESEL	Command Dividing/Multiplier Switchin	OFF
☑	GSEL	Gain Switching	OFF
☑	NOT	CW Overtravel Inhibited	OFF
☑	POT	CCW Overtravel Inhibited	OFF
☑	RESET	Alarm Cleared	OFF
☑	RUN	Servo-On	ON
☑	VSEL1	Internal Speed Command Selection 1	OFF
☑	VSEL2	Internal Speed Command Selection 2	OFF
☑	VZERO	Zero Speed Clamp	OFF

Figura 32: Entradas Digitales

La Figura 32 indica 10 entradas, pero el servo controlador solo tiene 7 puertos de entradas digitales. Esto se debe a que, para activar los puertos restantes es necesario cambiar los parámetros internos del servo por medio de CX Drive. Es decir, pin número 4 puede activar ECRST y VSEL 2, sí y solo sí, cuando el parámetro de esa entrada esté activo, ya sea para ECRST o VSEL 2.

Index	Description	Value
/ALM	Servo Alarm	---
BKIR	Electromagnetic Brake Released	---
DYNMC_BRK	Dynamic Brake Activated	---
INP	Positioning Complete	---
READY	Servo Ready	---
T_LIM	Torque being Limited	---
TGON	Speed Achieved	---
WRN:B5	Over-regeneration	---
WRN:B7	Overload	---
ZSPD	Zero Speed Detected	---

Figura 33: Salidas digitales

La Figura 33 indica los estados de las salidas del puerto CN1.

Index	Description	Value
CSPD	Current Speed	95 r/min
CTRLMODE	Control Mode	0: High speed r
DEV	Current Deviation Counter	283 Pulse(s)
STATUS:B2	CW Rotating	OFF
STATUS:B3	CCW Rotating	ON
TRQ	Current Torque Output	2.5 %

Figura 34: Estatus del servo controlador

...	Index	Description	Value	Drive Value	Default	Range	Units	Rest...
	Pn00	Unit No. Setting	1	---	1	1 to 15		
	Pn01	Default Display	1: Servomotor Rotatio...	---	1	0 to 15		
	Pn02	Control Mode Selection	2: High function positi...	---	2	0 to 2		
	Pn04	Drive Prohibit Input Selection	1: Disabled	---	1	0 to 1		
	Pn06	Zero Speed Designation/Torque Limit Switch	1: Zero speed designa...	---	1	0 to 2		
	Pn09	Warning Output Selection	2: Over regeneration/...	---	2	0 to 6		

Figura 35: Estatus de carga de motor

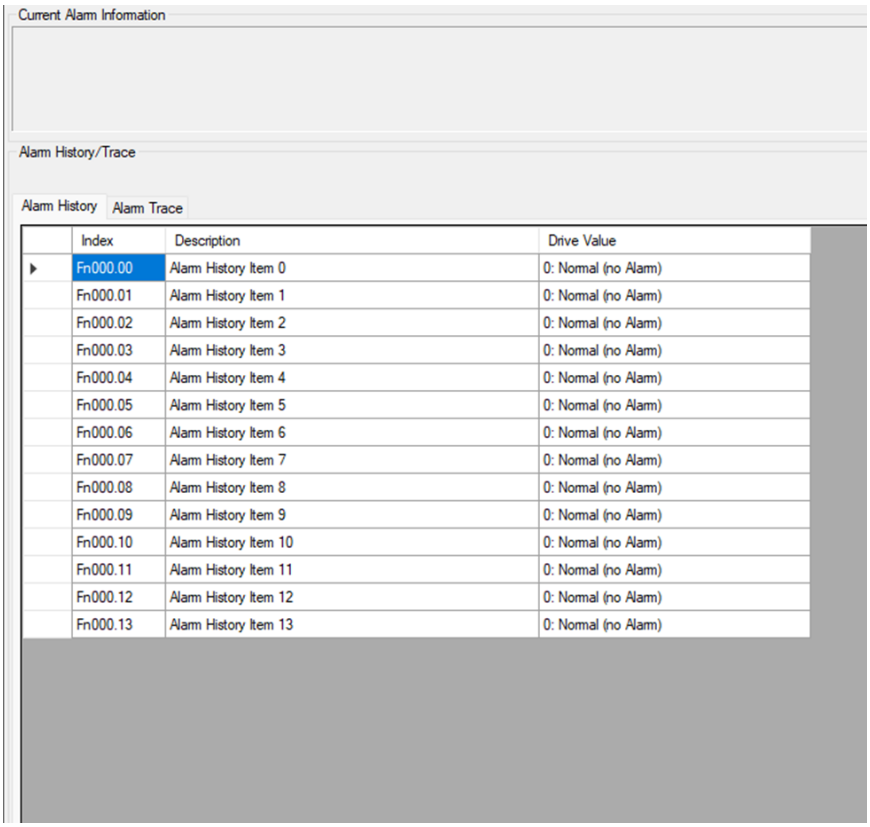


Figura 36: Alarmas y estados de las mismas

Figura 37: Contenido de carpeta de monitoreo

Los trazos en tiempo real pueden mostrarse para los parámetros seleccionados del variador en línea. Además, son posibles los trazos de datos para los servos R7D-AP y R88D-WT.

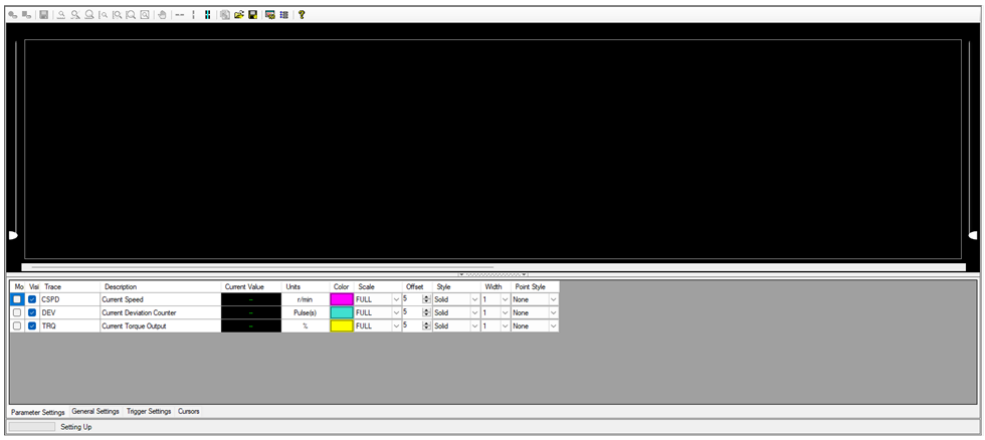


Figura 38: *Real time trace*

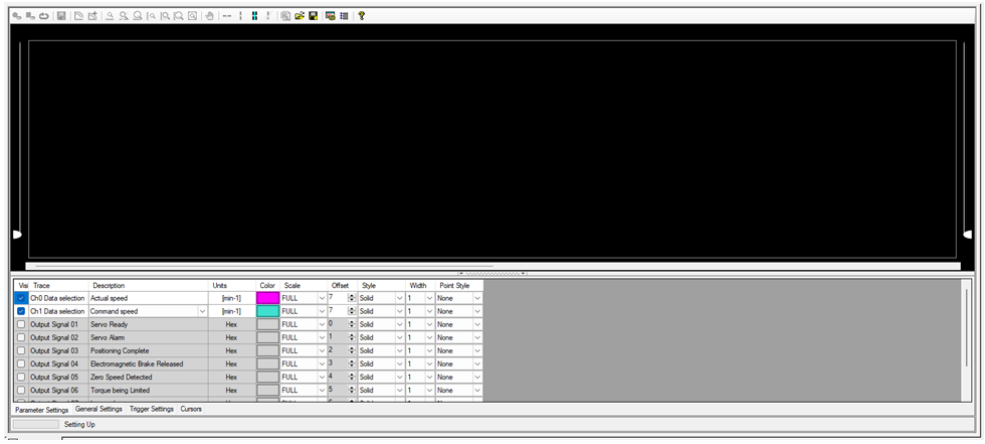


Figura 39: *Data Trace*

Rastreo de datos para obtener datos detallados del Servo controlador y mostrarlos, además de guardarlos en un archivo.

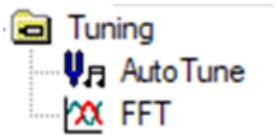


Figura 40: Contenido de carpeta *Tuning*

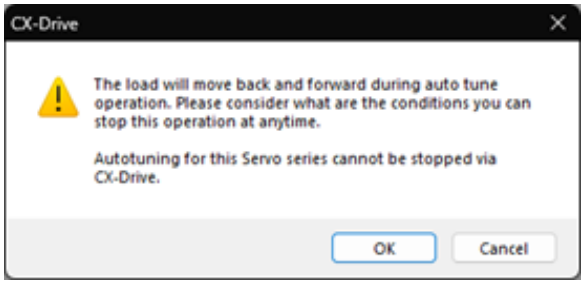


Figura 41: Alerta del movimiento del motor con la carga para la función de *Auto Tuning*

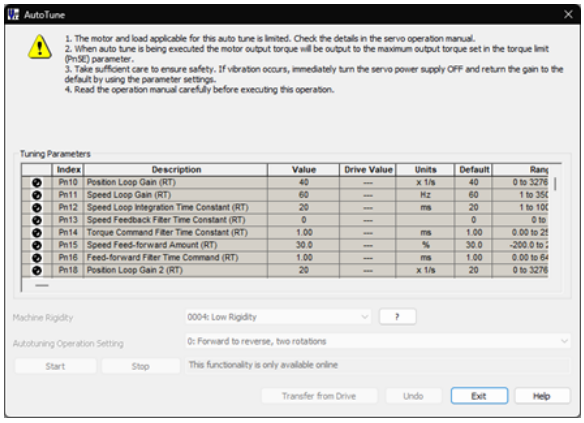


Figura 42: Parámetros para el movimiento óptimo de servo controlador y la carga del motor.

Con esta función se pueden ajustar automáticamente las ganancias para satisfacer el rendimiento requerido de la máquina al recibir comandos del servo. Especialmente con la Serie G5 (R88D-KT/R88D-KN) se pueden realizar ajustes automáticos de ganancia mediante operaciones simples con el asistente. Al seleccionar un sistema mecánico, hacer ajustes de operación y establecer condiciones de finalización según el asistente, el motor funciona de acuerdo con los ajustes y se calculan automáticamente los valores óptimos de los parámetros de ganancia.

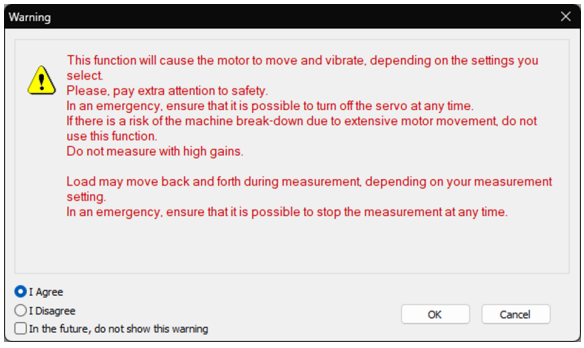


Figura 43: Alerta de movimiento del motor para la función FFT.

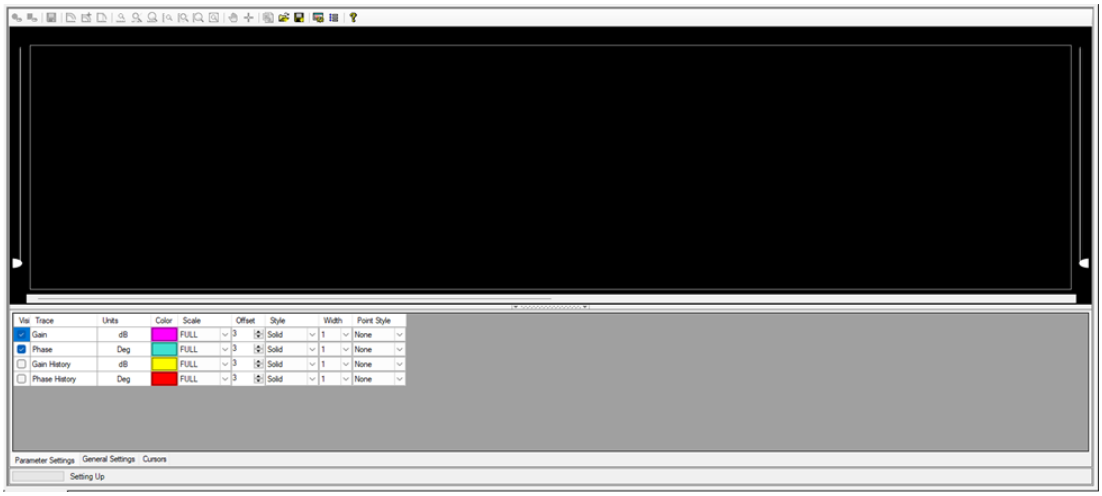


Figura 44: Gráfica de datos obtenidos del motor

El análisis FFT puede realizarse para el variador en línea, lo que permite especificar la frecuencia resonante del dispositivo mediante la verificación de la ganancia de la respuesta en frecuencia, lo cual es especialmente útil para ajustar manualmente los ajustes del filtro de muesca. Asi mismo, se puede verificar la respuesta del dispositivo al revisar la fase de la respuesta en frecuencia, lo que es efectivo para ajustar aspectos como la constante de tiempo de integración del lazo de velocidad y la cantidad de retroalimentación de velocidad.

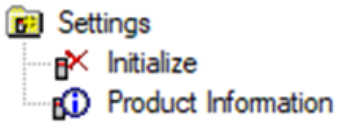


Figura 45: Contenido de carpeta *Settings*

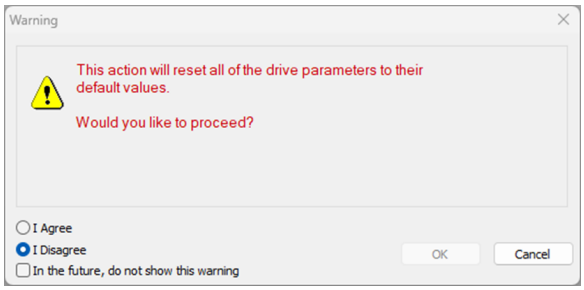


Figura 46: Alerta de reinicio de parámetros para la función de *Initialize*

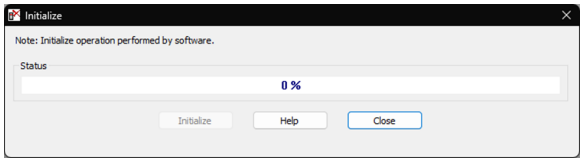


Figura 47: Pantalla de carga de estado de la función de *Initialize*

Se pueden inicializar los parámetros del variador en línea, y para algunos modelos de variadores se puede establecer la autorización de contraseña

CAPÍTULO 13

Laboratorio

Universidad del Valle de Guatemala
Facultad de Ingeniería
Depto. de Ingeniería Electrónica y Mecatrónica
IExxxx – Instrumentación y Automatización Industrial x

Laboratorio No. X: Servo controlador y servomotor.

Precaución: En este laboratorio estará utilizando Alto Voltaje, tome las medidas de seguridad necesarias para protección de su persona y de sus compañeros.
Ante cualquier duda mejor consulte si todo está bien.
SIEMPRE cumpla con las reglas del Laboratorio y de Seguridad Industrial.

Materiales:

- Servo controlador
- Servomotor
- Cables de conexión
- Tablero
- Herramienta
- Circuito de acople
- Cable PROFIBUS

Asegúrese de tener la configuración de **su PLC** correcta y de entender el inciso “Funcionamiento del circuito de acoplamiento” en la página 8 del manual de usuario.

A partir de este material realizar la siguiente configuración electrónica y su respectivo programa.

Parte 1. Conexiones del tablero:

Revise como están conectados los componentes para alto voltaje. Ver que espiga que se debe de utilizar. Revise continuidad de las fases de alto voltaje desde la espiga hasta los primeros dispositivos que estén conectados.

¿A cuántos y qué dispositivos está conectada la espiga de alto voltaje?
La espiga de alto voltaje está conectada al disyuntor, el cual se conecta al guardamotores y a través de este al contactor.
Tómeles una fotografía.?

Parte 2. Conexión entre el circuito de acoplamiento y el PLC:

1. Con el PLC apagado conectar GND del circuito de acople con el GND del PLC.
2. Con el PLC apagado conectar +24V del PLC a Vin del circuito de acople.
3. Desconectar dos de los cables del contactor del variador y conectar los cables de alimentación del servo controlador.

¿Qué línea debería ir neutro?

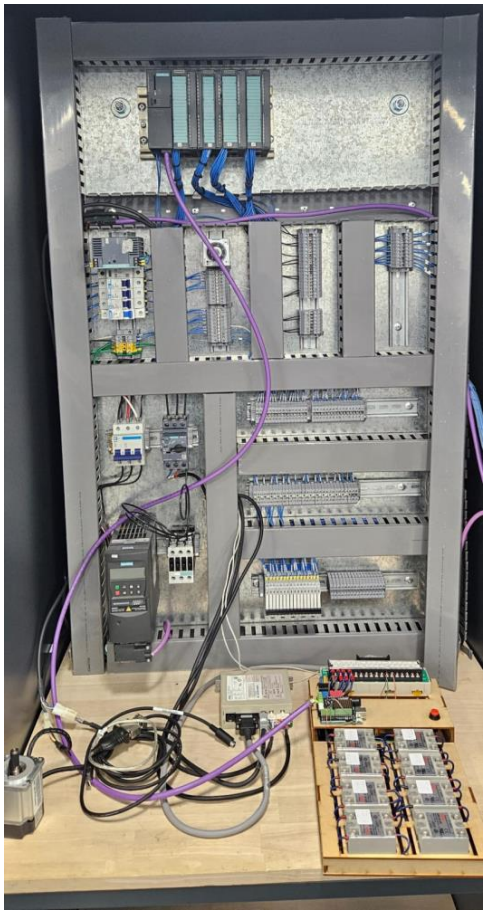
El servo controlador usa L1 y L2

4. Encender el contactor.

¿En el servo controlador cual es la secuencia de las luces?

El servo enciende la luce de PWR y ALM al mismo tiempo y después se apagan y se ilumina solo la de PWR.

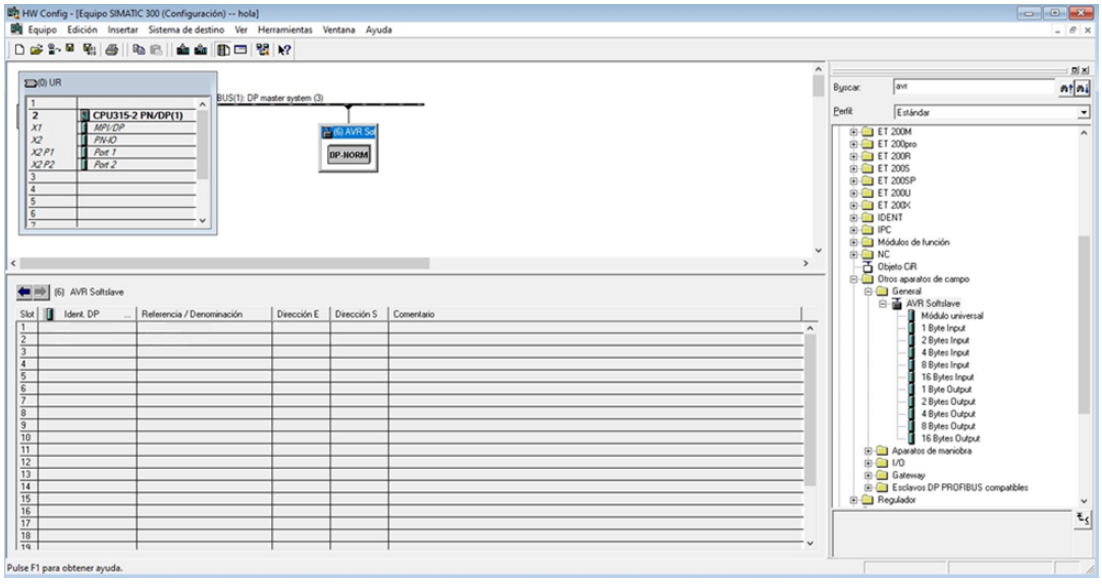
Tomar Foto de la conexión



Después de estos pasos puede encender el PLC de manera normal.

Parte 3: Configuración de comunicación PROFIBUS DP:

Para enviar datos des del PLC hacia el circuito de acople, debe de entrar a la pantalla de “Hardware” para poder seleccionar el módulo de “AVR Software”. Este posee la cantidad de datos que se desean mandar se debe seleccionar 1 byte de entrada y 1 byte de salida.



¿Qué dirección de entradas utilizó para él envió de datos?

La dirección de localidad de memoria para el envió de datos es la AB0

Parte 4. Envió de datos con el programa:

En la pantalla de OB1 se busca la función de “Move” y con esta se ejecutara un programa que use enciende y apague el servo controlador (con el botón verde), cambio de velocidad alta, velocidad baja y alto (con los la perilla del tablero) de y el cambio de dirección del motor (por medio del botón rojo) y un alto de emergencia (con el botón de emergencia del tablero).

Bits de comunicación	Puertos internos del Arduino	Conexiones al puerto CN1
No.0	A4	Run
No.1	3	Cambio de frecuencia para CW
No.2	5	Cambio de dirección de giro
No.3	A5	ECRST & VSEL2
No.4	7	GSEL & VZER0 & TLSEL
No.5	A2	GSEL & VZER1
No.6	A1	NOT
No.7	A0	POT

Muestre el funcionamiento de sus programas al catedrático o auxiliar.

ENTREGABLE:

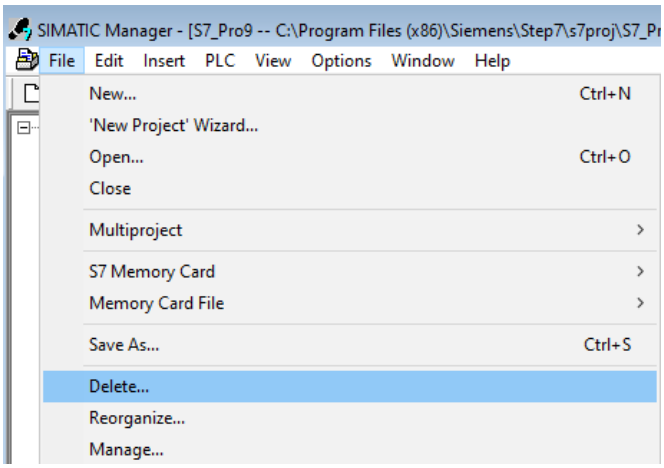
1. Responda:

   - ✓ ¿Qué circuito utilizó para darle poder al servo controlador (dibuje)?
   - ✓ Preguntas hechas en la guía.

2. Captura de pantalla del código programado en el OB1.

3. Captura de pantalla de la tabla de símbolos utilizada.

Al finalizar y ya tener su archivo zip guardado en la nube, USB o en Canvas, **elimine por completo de la PC** el proyecto realizado, aparte del zip, borrar en Simatic donde dice Delete (a menos que sea su laptop personal puede ignorar este paso).



Recuerde realizar la limpieza de su área de trabajo antes de retirarse y cambiar la **IP a dinámica** (si está la cambió a estática).