CARRERA DE ESPECIALIZACIÓN EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Controlador CAN de servomotores

Autor:
Ing. Alejandro Virgillo

Director:
Esp. Ing. Gabriel Gavinowich (FIUBA)

Jurados:
Nombre del jurado 1 (pertenencia)
Nombre del jurado 2 (pertenencia)
Nombre del jurado 3 (pertenencia)

Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires, entre Octubre de 2021 y Abril de 2023.

Resumen

El presente trabajo aborda el proceso de diseño y fabricación de un sistema de control centralizado para una serie de servomotores utilizando el protocolo CAN. El desarrollo se hace junto a la organización A3 Engineering para implementar en las instalaciones industriales de Cambre ICyFSA.

El documento detalla los conocimientos utilizados sobre los protocolos de comuncación implementados, el desarrollo del software embebido en capas, el diseño y fabricación del hardware y su gabinete, y las consideraciones de manufactura que se tomaron.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción general

En esta sección se da una introducción al protocolo CAN o *Controller Area Network*[1] y al concepto de un servomotor. Se describe el proyecto SN-17, la motivación y alcance del trabajo y se resume el estado del arte de esta tecnología.

1.1. Controller area network

CAN es un protocolo de comunicación desarrollado por la empresa Bosch [2] orientado originalmente a la industria automotriz. En el año 1991 Bosch publica la especificación CAN 2.0 y, en el año 1993, es adoptado internacionalmente bajo la norma ISO 11898 [3]. Desde sus inicios el protocolo obtuvo gran popularidad y en la actualidad es utilizado en diversas industrias.

CAN se caracteriza por su robustez y bajos requerimientos de cableado. En su concepción fue pensado para proveer comunicaciones determinísticas en sistemas complejos, caracterizado por las siguientes cualidades[4]:

- Prioridad de mensajes y latencia máxima asegurada.
- Comunicaciones a varios dispositivos al mismo tiempo.
- Bus multi-maestro.
- Detección de errores en nodos y mensajes.
- Protocolo asincrónico sin línea de clock.

Para realizar la transferencia de información CAN requiere únicamente 2 líneas de transmisión denominados CAN High (CAN H) y CAN Low (CAN L). Los mensajes CAN contienen un identificador que indica la prioridad del mensaje, así como a quien está dirigido. Usando esta información cada nodo de la red determina que acción debe realizar. En la figura 1.1 se presenta un esquema básico de una red CAN compuesta por 4 nodos.

1.2. Servomotores

Un servomotor[5] es un tipo de motor eléctrico que tiene la capacidad de controlar la posición angular de su eje, así como la velocidad de rotación y el torque de salida. Esto se consigue con la utilización de un sistema de lazo cerrado de control que se retroalimenta con un sensor de posición angular llamado *encoder*. En general, el tipo de bobinas del motor eléctrico no determina la condición de

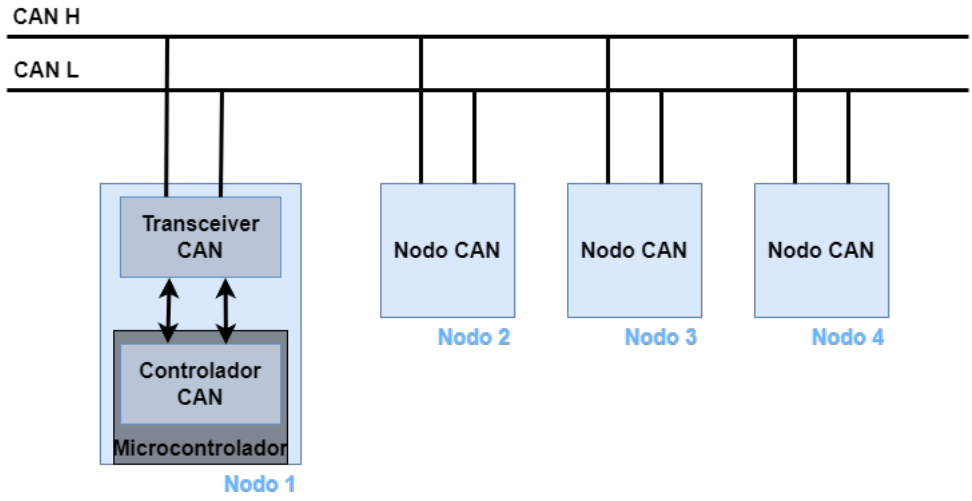

FIGURA 1.1. Esquema de red CAN.

servomotor, sino que este cuente con las cualidades de control descriptas. En la figura 1.2 se pueden observar las distintas partes que conforman un servomotor.

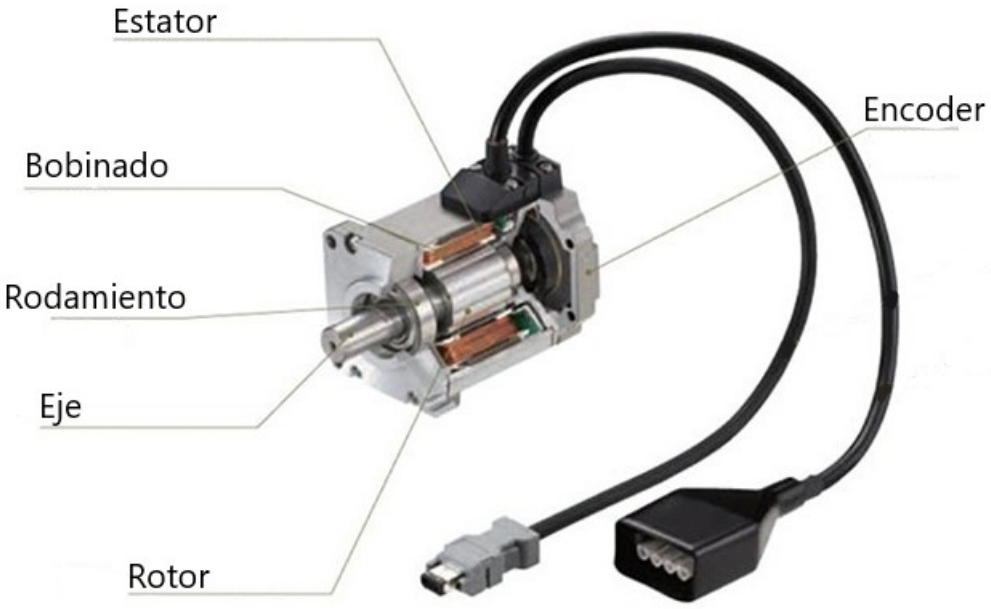

FIGURA 1.2. Partes de un servomotor[1].

1.3. Proyecto SN-17

El proyecto SN-17 (Servo NEMA 17) es un sistema desarrollado como emprendimiento personal (A3 Engineering) que adapta motores eléctricos del tipo paso a paso o *steppers* para funcionar como servomotores. El sistema SN-17 agrega un *encoder* y un *driver* al motor, el cual genera un lazo cerrado y logra las funcionalidades de un servomotor. En la figura 1.3 se puede observar una placa de control SN-17. Esta se coloca en la parte trasera del motor paso a paso.

[1] https://www.logicbus.com.mx/blog/que-es-un-servo-motor/

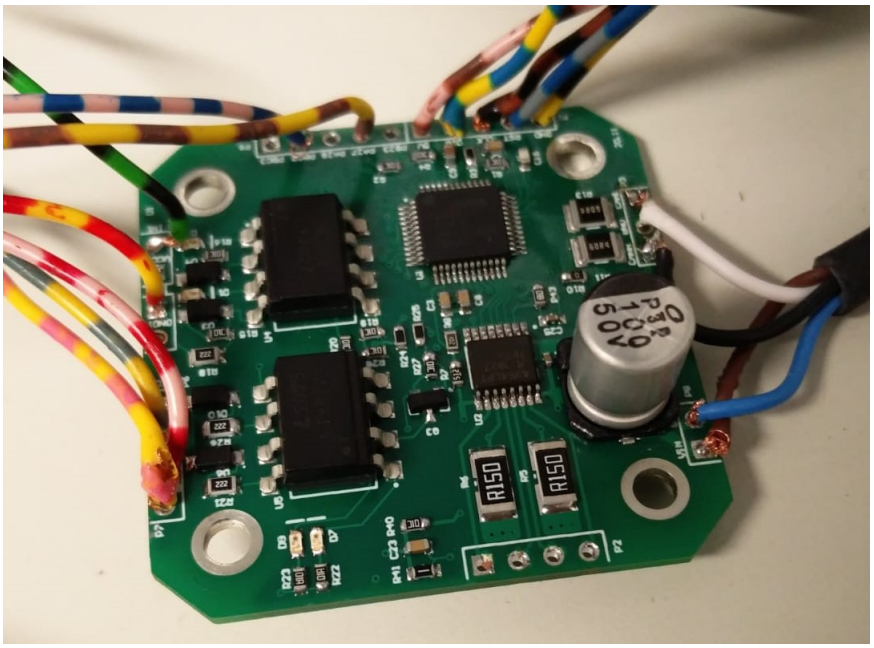

FIGURA 1.3. Plaqueta SN-17.

Además de generar las funcionalidades mencionadas, el sistema SN-17 tiene señales discretas industriales que le permite comunicarse con controladores industriales (*Programmable Logic Controllers* o PLCs).

Actualmente se está trabajando en implementar estos sistemas en la planta de la empresa Cambre ICyFSA[6], donde se están construyendo distintos dispositivos para aplicaciones industriales con estos motores. En la figura 1.4 se puede observar un dibujo CAD de un actuador lineal con un sistema SN-17 implementado. Este funciona como un prensador en un proceso industrial.

1.4. Motivación

En la actualidad el ámbito de actuadores industriales está principalmente dominado por aquellos de carácter neumático BUSCAR REFERENCIA. Sin embargo, en los últimos años el uso de actuadores eléctricos ha empezado a ser más significativo debido a las mayores aptitudes de control que poseen y a sus menores costos operativos[7]. En la tabla 1.1 se muestra una comparativa de ambos tipos de actuadores.

Los actuadores eléctricos suelen emplear un servomotor en su interior que, junto con un mecanismo, generan el movimiento deseado con una precisión superior a los actuadores neumáticos. Aún así, este tipo de actuadores tienen como problema su elevado costo de adquisición y su dificultad de implementación, que limitan su uso. En este contexto la organización A3 Engineering desarrolló el sistema SN-17 en un intento de disminuir los costos de los servomotores de aplicaciones pequeñas y, en un futuro, de los actuadores eléctricos.

La empresa Cambre ICyFSA instaló una serie de sistemas SN-17 en su planta productiva con resultados iniciales exitosos. Sin embargo, se limitó la implementación de más sistemas debido a su dificultad para reprogramarlos en momentos

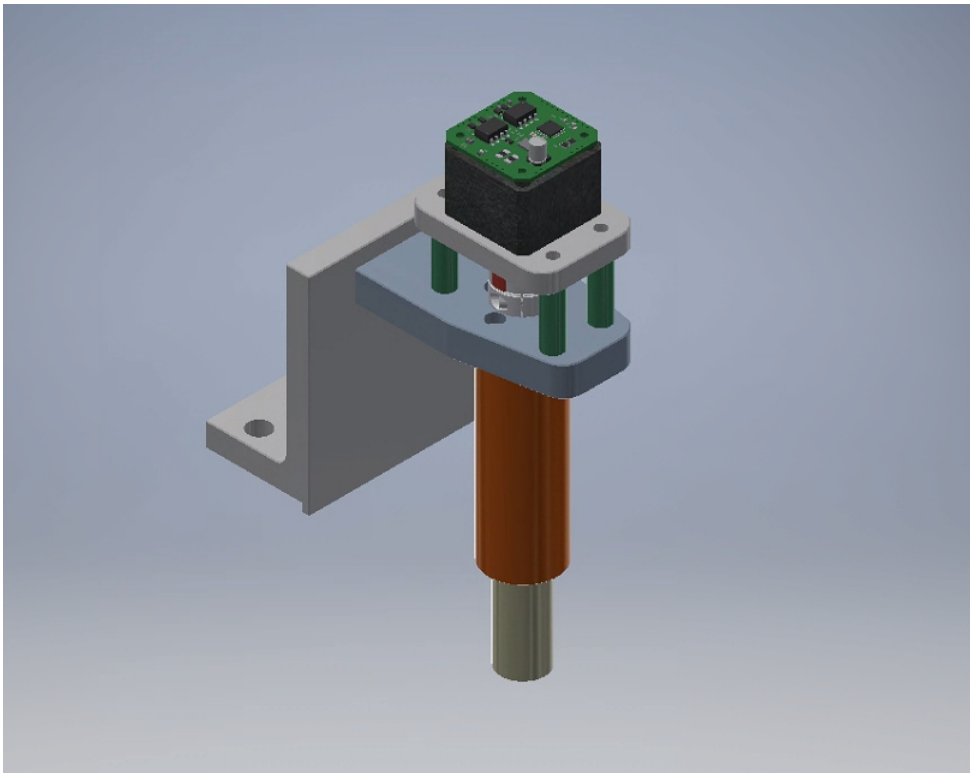

FIGURA 1.4. Actuador lineal con SN-17.

TABLA 1.1. Comparación de actuadores neumáticos y eléctricos.

	Actuadores neumáticos	**Actuadores eléctricos**
Diseño	Simple	Complejo
Implementación	Simple	Complejo
Fuerza	Según presión de aire	Según reducción mecánica
Presición	Baja	Alta
Repetitibilidad	Baja	Alta
Eficiencia	Baja	Alta
Control de movimiento	Bajo	Alto
Mantenimiento	Alto	Bajo

de necesidad. Esto se debe a que requieren de un cambio de *firmware* para modificar su configuración y esto solo puede ser realizado por personal calificado. La motivación de este trabajo es brindar una interfaz para programación y monitoreo de una red de motores que pueda ser utilizada por personal no especializado.

1.5. Estado del arte

Actualmente existe una extensa oferta de servomotores y motores paso a paso en el mercado. En general, lo que en el ámbito industrial se refiere a servomotores[5] son motores del tipo DC *brushless* o AC de inducción (monofásicos y trifásicos). Estos tienen un precio significativamente mayor que los *steppers* y suelen requerir de *drivers* externos y, en caso de motores AC, variadores de frecuencia. En los casos en que se requiere alta potencia o alto rendimiento, estos servos son la opción indicada. Requieren de técnicos capacitados para su programación y ofrecen interfaces de usuario avanzadas. En general, representan soluciones de alto costo. Un motor AC de la empresa Panasonic junto con su *driver* se presenta en la figura 1.5.

FIGURA 1.5. Servomotor AC con *driver*[2].

Por otro lado, en lo referido a motores *steppers*, la oferta también es variada. Estos pueden adquirirse a precios muy accesibles y en distintos tamaños con lazo de control abierto. Requieren el uso de un *driver* externo para su operación y su programación suele hacerse con software de terceros. Según la aplicación pueden requerir de conocimientos técnicos elevados. También existen variantes que agregan un *encoder* de posición angular, llamados *closed loop steppers*. Estos solventan uno de los mayores problemas de estos motores relacionado con la pérdida de posición en caso de una perturbación, pero no permiten entregar torque constante. También requieren de *drivers* externos y, según el fabricante, las interfaces de programación pueden ser complejas. Un ejemplo de un *closed loop stepper* se presenta en la figura 1.6, junto con su *driver* correspondiente.

[2] https://na.industrial.panasonic.com/products/industrial-automation/motion-control/lineup/ac-servo-motors

[3] https://www.autonics.com/product/category/2000023

FIGURA 1.6. *Closed loop stepper* con *driver*[3].

Finalmente, existen placas de control que, además de funcionar como un *closed loop stepper*, tienen un *driver* incorporado que permite controlar los motores de forma diferente, logrando entregar torque constante. El problema principal de estas soluciones es que suelen tener características de *hobbista* y no son aptas para ámbitos industriales. También requieren de elevado conocimiento técnico para su programación debido a la falta de interfaces de programación. En la figura 1.7 se puede ver un ejemplo de una de estas placas llamada *Mechaduino* [8].

En la Tabla 1.2 se resume la información de los distintos tipos de motores y sus controles explicada en esta sección.

TABLA 1.2. Resumen de características de servomotores.

Motor	**Lazo de control**	***Driver***	**Programación**	**Precio (USD)**
Servomotor	Cerrado	Externo	Compleja	>1.500
Stepper open loop	Abierto	Externo	Intermedia	300
Stepper closed loop	Cerrado s/torque	Externo	Intermedia	400
Mechaduino	Cerrado	Interno	Compleja	200
SN-17 + interfaz	Cerrado	Interno	Simple	300

[4]https://tropical-labs.com/mechaduino/

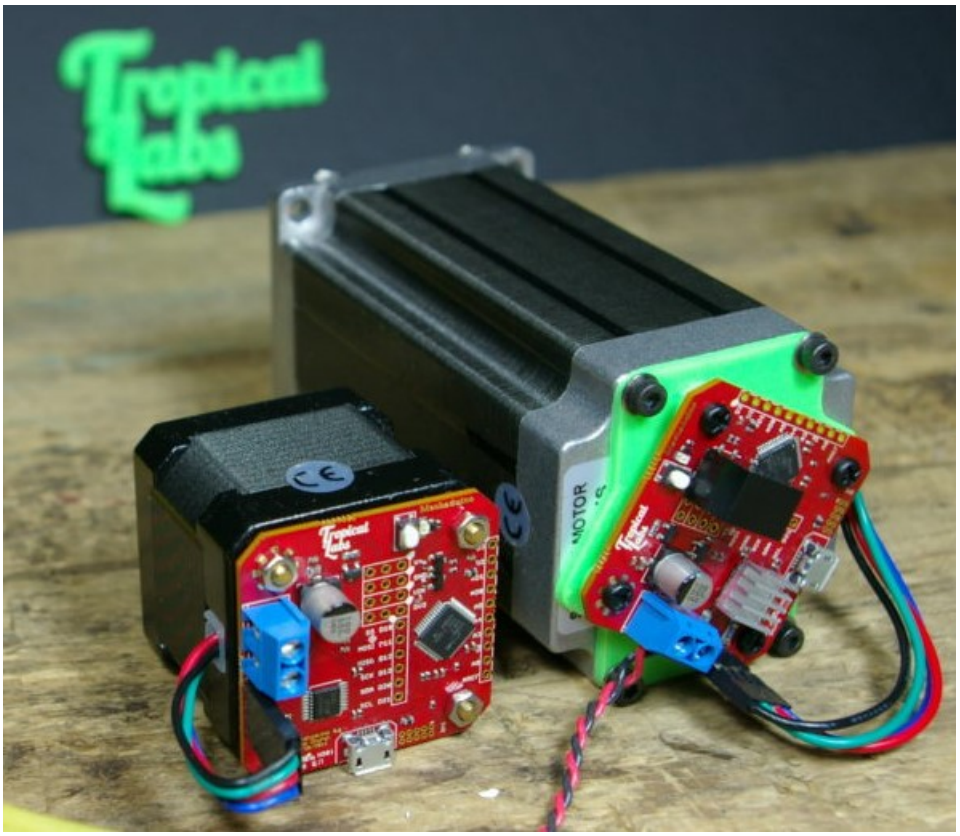

FIGURA 1.7. Proyecto Mechaduino[4].

1.6. Objetivos y alcance

El objetivo de este trabajo es proporcionar una interfaz de usuario para el sistema SN-17 que permita que un operario no especializado pueda modificar los programas de los distintos motores conectados a través de una red CAN. La interfaz también debe poder emplearse para monitorear el estado de los motores conectados cuando están operativos.

El proyecto incluye:

- Una interfaz de usuario que permita configurar y supervisar los servomotores conectados.
- La construcción e implementación de la estructura de mensajes CAN.
- La configuración de la red CAN.
- El desarrollo y fabricación de una plaqueta con el sistema embebido.
- El desarrollo del firmware para este sistema y la adaptación del firmware del sistema SN-17.

Capítulo 2

Introducción específica

Este capítulo da una introducción detallada del sistema SN-17 y el protocolo CAN. También presenta algunas características de las entradas y salidas de los controladores industriales PLCs y explica los distintos componentes seleccionados para este trabajo junto con su funcionamiento.

2.1. Especificaciones SN-17

El sistema SN-17 se desarrolló para controlar motores *stepper* pequeños que siguen los estándares NEMA (*National Electrical Manufacturers Association*)[9]. Estos son estándares ampliamente utilizados para esta clase de motores que indican, entre otras características, las dimensiones mecánicas que debe tener el motor.

El sistema SN-17 consiste en una plaqueta electrónica que se monta en la parte trasera de un motor *stepper*. Las dimensiones de estas plaquetas permiten el montaje directo con aquellos del tipo NEMA-17 cuyas dimensiones se muestran en la figura 2.1. Mediante una adaptación mecánica, también es posible emplear la plaqueta para controlar motores de menor o mayor tamaño, mientras estos sean del tipo *stepper* y no requieran corrientes mayores a 2 A.

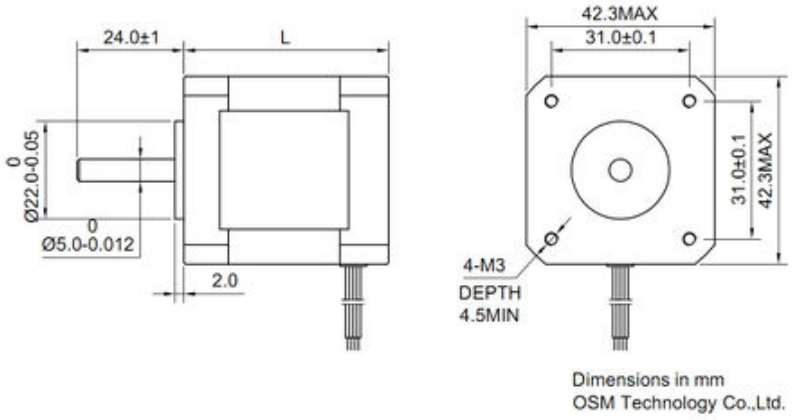

FIGURA 2.1. Dimensiones NEMA 17[1].

Las características electrónicas del sistema SN-17 son:

- Alimentación de 12 a 48 V DC para la operación del motor, de la placa y, opcionalmente, de las entradas y salidas industriales.

[1] https://reprap.org/wiki/NEMA_17_Stepper_motor

- Regulación de 5 V para electrónica interna.
- Microcontrolador ATSAMC21[10] con *core ARM Cortex-M0+* y periférico controlador de CAN.
- *Driver* de motor del tipo doble puente H[11].
- *Encoder* magnético absoluto[12] para sensar la posición angular del motor.
- *Transceiver* CAN para adaptar las señales de comunicación del bus.
- Subcircuito de entradas y salidas discretas PNP optoacopladas[13], con alimentación separada, para interacción con controladores industriales.

El sistema funciona mediante un ciclo de control de lazo cerrado del tipo PID[14] al cual se le indica un valor deseado (*set point*) de posición, velocidad o torque. Esta información se compara con los valores obtenidos del encoder y se determina como deben ser alimentadas las bobinas del motor para alcanzar el *set point*. Para que el sistema opere correctamente el *encoder* debe ser calibrado junto con el motor mediante una rutina que genera una tabla de referencia para ser utilizada en operación.

Sobre este control se ejecuta una aplicación en la que se cargan programas que el motor debe realizar. Estos programas estan compuestos de instrucciones configurables que permiten establecer los modos de control deseados (posición, velocidad o torque), los *set points* y errores admisibles para estos movimientos (*thresholds*), una limitación del torque para esa instrucción, los tiempos que deben cumplirse para considerar correcta la instrucción (*hold time*) y los tiempos para que se considere que el sistema entró en estado de error (*timeout*). También cuenta con instrucciones para el control del flujo de programa, interacción con las entradas y salidas y comunicaciones a través del puerto CAN.

Existen configuraciones globales que se le pueden aplicar al sistema. Estas son:

- Constantes PID del lazo de control.
- Tipo de rutina de *homing* del motor.
- Funcionamiento de entradas y salidas.
- Guardado de posiciones de eje en memoria.

El sistema también puede recibir comandos de forma manual:

- Calibración de encoder con motor.
- Activación o apagado de motor.
- Cerado de motor.
- Activación de salidas.

En la Tabla 2.1 se resume la información sobre las distintas funciones que la aplicación del sistema SN-17 puede realizar.

2.2. Características del protocolo CAN

Como se trató en el capítulo 1, el protocolo CAN está estandarizado bajo la norma internacional ISO 11898[3]. El estándar abarca solamente las capas física y de

TABLA 2.1. Funciones de SN-17.

Señal	Tipo	Descripción
Tipo de instrucción	Instrucción	Define la instrucción
Límite de torque	Instrucción	Torque máximo de instrucción
Modo de control	Instrucción	Posición, velocidad, torque
Set point	Instrucción	Valor de lazo de control
Threshold	Instrucción	Valor de error admisible
Hold time	Instrucción	Tiempo de cumplimiento
Timeout	Instrucción	Tiempo de no cumplimiento
Guardar posición	Configuración	Guarda posición actual
Constantes PID	Configuración	Constantes lazo de control
Entradas y salidas	Configuración	Funcionamiento de las IO
Homing	Configuración	Establece rutina de cerado
Calibración	Comando	Inicia la rutina de calibración
Activar motor	Comando	Enciende/apaga el motor
Go Home	Comando	Inicia la rutina de cerado
Activar salidas	Comando	Enciende/apaga salidas

enlace de datos, es decir las 2 capas más bajas en un modelo OSI (*Open System Interconection*), como el que se presenta en la figura 2.2. Para las capas superiores existen otros estándares como CANOpen[15], el cual se utilizó como referencia para el armado de la red de este trabajo.

Otras características de la red se relacionan con el armado del circuito de los nodos, el cual suele estar especificado en las hojas de datos de los *transceivers*, así como la necesidad de colocar una resistencia de terminación en los extremos de la red. Estas características se presentan en la figura 2.3. El valor de las resistencias de terminación depende de diferentes variables, entre ellas el largo de la red y la velocidad de transmisión y se emplean guías para seleccionarlas. Los medios físicos de transmisión, así como los conectores, no están especificados por la norma pero existen recomendaciones para su diseño[16].

Las señales usadas por el protocolo son del tipo diferencial y se clasifican en dominantes y recesivas. Una señal dominante en el bus tiene prioridad por sobre una señal recesiva, lo que permite que los nodos sepan cuando pueden enviar mensajes sin que ocurran colisiones en el bus[16].

El estándar subdivide el tiempo de un bit en distintos segmentos: Sincronización, propagación, fase 1 y fase 2. Estos se describen en una unidad de tiempo menor referida como *time quanta*. Para un buen funcionamiento de la red, todos estos parámetros deben ser correctamente configurados en los distintos nodos CAN.

En lo que respecta a las tramas CAN, existen de 2 tipos: estándar y extendida. En este trabajo el enfoque está en las tramas estándar que se caracterizan por tener un identificador de 11 bits, donde se codifica información del receptor del mensaje. En la figura 2.4 se puede ver un esquema detallando la trama CAN estándar. La trama se separa en:

[2] https://www.cloudflare.com/es-es/learning/ddos/glossary/open-systems-interconnection-model-osi/

[3] https://www.seeedstudio.com/blog/2019/11/27/introduction-to-can-bus-and-how-to-use-it-with-arduino/

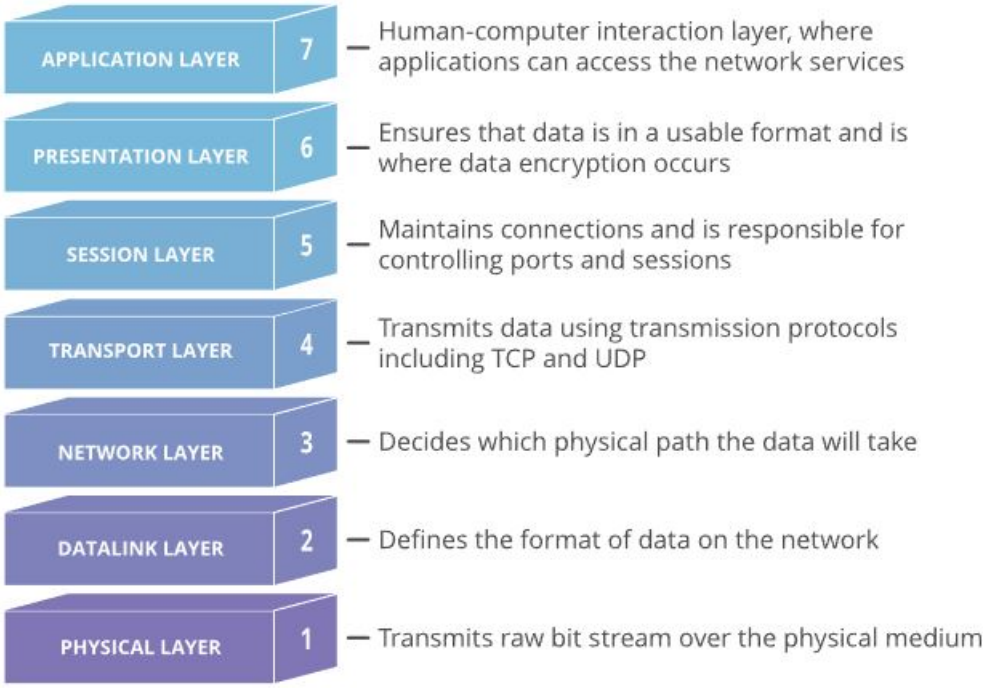

FIGURA 2.2. Modelo de capas OSI[2].

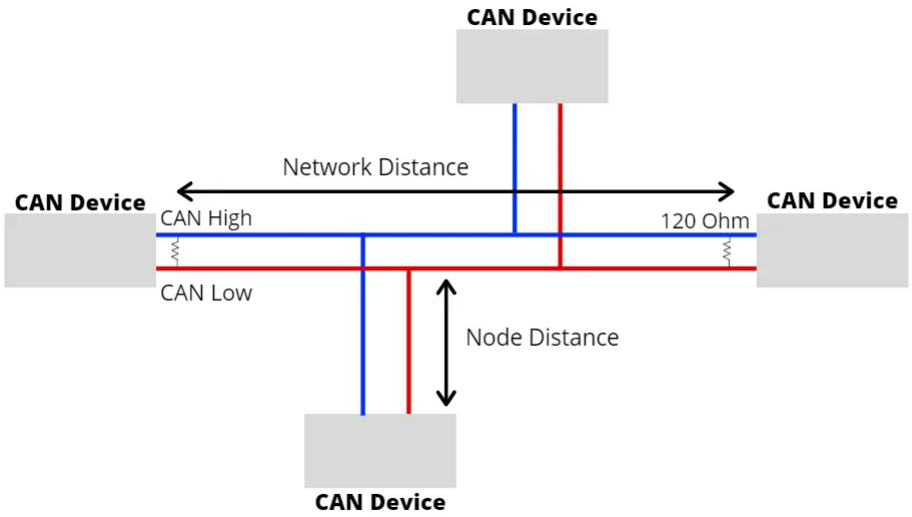

FIGURA 2.3. Esquema de red CAN con resistores de terminación[3].

- *Start of frame*.
- Arbitraje.
- Control.
- Datos a envíar.
- Verificación.
- Reconocimiento - *Acknowledge*.
- *End of frame*.

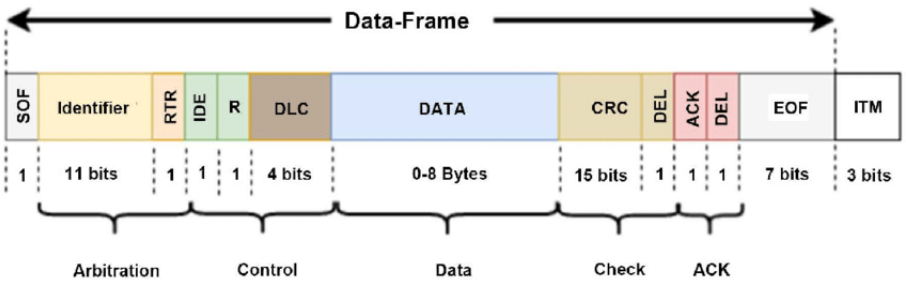

FIGURA 2.4. Trama CAN estándar[4].

La sección de arbitraje está compuesta por el identificador y el bit RTR (*Remote Transmission Request*). El identificador es procesado por cada nodo de la red y, mediante el uso conjunto de máscaras y filtros, determinan si el mensaje corresponde a ese nodo o no. También, en caso de que dos o más nodos deseen transmitir un mensaje al mismo tiempo, el que lo haga con el identificador más bajo será el que tendrá prioridad y tomará el control de la red. Los nodos que no logran transmitir su mensaje reconocen la situación y retransmiten el mensaje una vez que sensan el bus desocupado.

El campo de control del frame tiene al bit IDE, que identifica el tipo de trama entre estándar y extendida, y el DLC, donde se indica la cantidad de bytes del mensaje a transmitir entre 0 y 8.

Luego se encuentran los campos de verificación y reconocimiento que se usan para determinar la correcta transmisión y recepción del mensaje. Estas secciones son manejadas de forma automática por los *transceivers*, así como el fin de trama.

2.3. Entradas y salidas de controladores industriales

Un PLC (*Programmable Logic Controller*) es un tipo de controlador utilizado para manejar procesos industriales. Se caracteriza por su robustez y su estilo de programación similar a la lógica de relé.

En general, los PLC requieren interactuar con un gran número de sensores y actuadores presentes en un proceso industrial, por lo que cuentan con distintos tipos de módulos de entradas y salidas digitales. Estos módulos adaptan el nível de tensión de las señales que reciben y transmiten y protegen los circuitos internos del PLC[17]. Los valores de tensión más utilizados en el ámbito industrial son

[4] https://www.researchgate.net/publication/328607559_Classification_Approach_for_Intrusion_Detection_in_Vehicle_Systems

24 V y 48 V (en menor medida) de corriente contínua aunque en algunos casos pueden encontrarse otras tensiones.

Uno de los circuitos típicos de acondicionamiento de salidas de un PLC se puede observar en la figura 2.5 [17]. Este ejemplo es del tipo NPN, que hace referencia al transistor empleado y al tipo de conexión externa que requiere para su línea común. Es importante notar el uso de un optoacoplador para separar eléctricamente los circuitos y la cantidad de elementos de protección que se agregan para dar robustez al sistema.

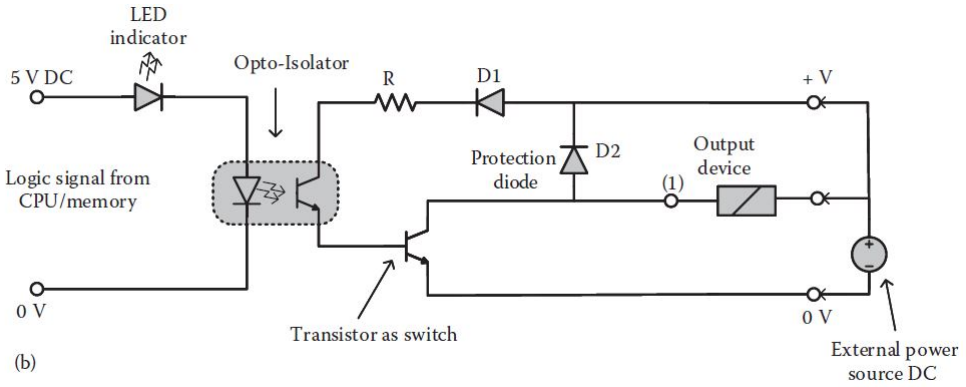

FIGURA 2.5. Circuito NPN.

Para especificar un módulo de entradas o de salidas se deben determinar los rangos de tensión y corriente de operación, el consumo máximo de corriente, el tipo de conexión (NPN, PNP o relé), y la velocidad y frecuencia de conmutación.

2.4. Características de componentes electrónicos empleados

2.4.1. Pantallas LCD y conversores I2C

Las pantallas LCD ofrecen una forma conveniente y económica para generar una interfaz de usuario. Dentro de las opciones comercializadas, uno de los modelos más populares es el panel de texto basado en el Hitachi HD44780[18]. En la figura 2.6 se muestra un display de 4 líneas y 20 caracteres por línea.

Existen modelos de LCD que incluyen una interfaz I2C (*Inter-Integrated Circuit*), un protocolo de comunicación simple muy utilizado en sistemas embebidos. Esta interfaz facilita la interacción con el display, que normalmente requiere una serie de conexiones discretas para controlarlo. De esta manera se pueden envíar comandos a través del protocolo de comunicación para modificar la configuración del dispositivo o mostrar texto en pantalla.

La mayoría de los microcontroladores que se emplean en la actualidad poseen un periférico de I2C y sus herramientas de desarrollo proporcionan drivers de este protocolo. I2C es serial y sincrónico y requiere 2 líneas bidireccionales para su operación, los datos y el reloj. En un bus I2C los distintos dispositivos conectados tienen un código identificador que se emplea para controlar las comunicaciones.

[5] https://www.adafruit.com/product/198

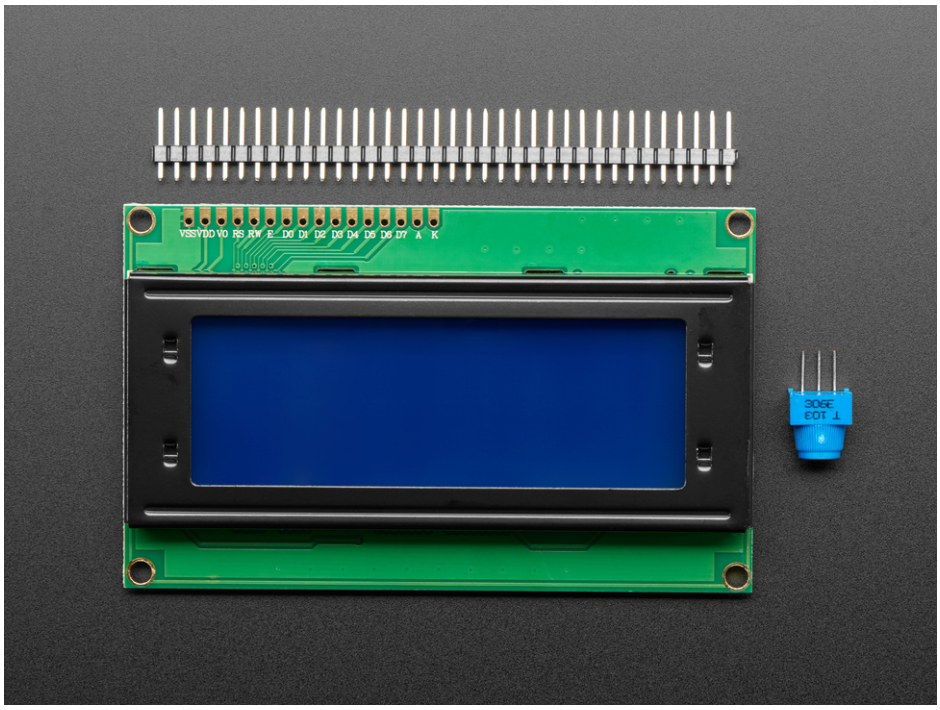

FIGURA 2.6. Pantalla LCD 20x4[5].

A la hora de implementar una solución con este tipo de pantallas puede tomarse de referencia alguna de las librerías de código abierto que se encuentran disponibles en la web. Esto ayuda a reducir significativamente los tiempos de desarrollo y es el camino que se tomó en este trabajo. En particular, se utilizó la librería *LiquidCrystal_I2C* para Arduino[19].

2.4.2. Matriz de botones

Las matrices de botones son un conjunto de contactos normalmente abiertos que conectan una fila con una columna cuando se presionan. Para su conexión con un microcontrolador se conectan las filas a pines de entrada con resistores de pull-up y las columnas a pines de salida. La secuencia de funcionamiento consiste en colocar el nivel de tensión de una columna por vez en un nivel bajo (mientras las otras se mantienen en un nivel alto). En ese momento, se leen las entradas de las filas, si alguna está en un nivel bajo es indicación de que el botón de esa fila y columna fue presionado (normalmente estarían en un nivel alto por las resistencias de pull-up)[18]. En la figura 2.7 [18] se puede visualizar un esquema de conexionado de una matriz de botones 4x3 a un microcontrolador. También se muestran los conexionados internos de la matriz, de las filas y las columnas.

Como en el caso de las pantallas LCD, existen muchas librerías de código abierto que implementan soluciones de matrices de botones. Se considera conveniente consultarlas si se desea utilizar uno de estos dispositivos y es el camino elegido en este trabajo. En particular, se utilizó la librería Keypad para Arduino[20].

2.4.3. Conversores UART-USB

La *Universal Asynchronous Receive Transmit* o UART es un protocolo serial de comunicación muy utilizado en sistemas embebidos. Se caracteriza principalmente

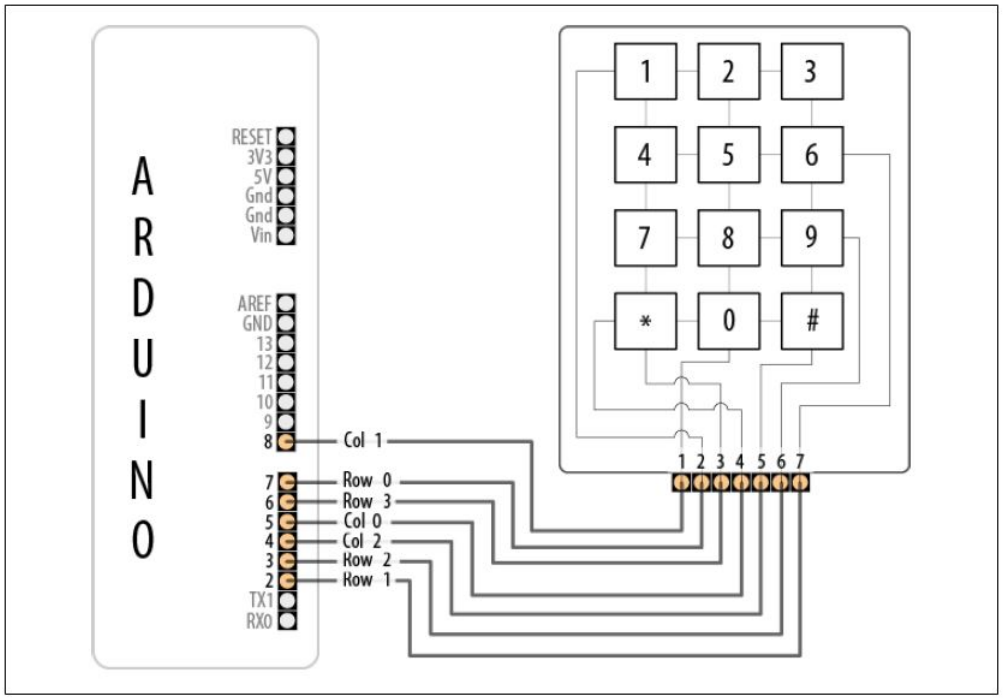

FIGURA 2.7. Conexión de matriz de botones 4x3.

por su sencillez. Como su nombre indica es un protocolo asincrónico que requiere una única línea para el envío de datos o dos si se desea envíar y recibir.

La mayoría de los microcontroladores que se comercializan en la actualidad disponen de uno o más periféricos UART. Es común que el propio fabricante provea los *drivers* de implementación para reducir los tiempos de desarrollo.

Otro protocolo ampliamente usado es el USB que suele ser el preferido para interactuar con una PC. Existen en el mercado una gran cantidad de conversores que transforman un mensaje codificado para el protocolo UART a uno USB, permitiendo la interacción entre una PC y un sistema embebido. En general, estos conversores cuentan con *drivers* para los sistemas operativos de PC más populares, por lo que solo es necesario conectarlos a través del puerto USB. Desde la PC puede utilizarse un programa de monitoreo de puertos seriales para envíar y recibir mensajes con el sistema embebido.

Este tipo de implementaciones suele ser muy útil para realizar operaciones de búsqueda de errores o para armar interfaces gráficas con una PC, lo que facilita la operatoria de un usuario final. En este trabajo se utiliza uno de estos módulos para esta finalidad.

Capítulo 3

Diseño e implementación

Este capítulo presenta la arquitectura del sistema embebido, la estructura de los componentes de software principales, las consideraciones en la selección de componentes electrónicos, el diseño del PCB y el gabinete del sistema.

3.1. Arquitectura del sistema embebido

En la figura 3.1 se muestra un diagrama de bloques con la arquitectura del sistema SCI-CAN y sus interacciones. El recuadro marcado como controlador corresponde a lo desarrollado en este trabajo. En los recuadros externos están los dispositivos con los que este sistema debe interactuar: PLCs, servomotores SN-17 y PC.

La interacción con los PLC es a través de señales discretas que necesitan estar correctamente acondicionadas como se especificó en los requerimientos del trabajo (COMO SE REFERENCIA EL PLAN DE TRABAJO? ES UN APENDICE?). Con los servomotores la comunicación es a través de un bus CAN y la interacción con una PC se consigue utilizando un conversor USB - UART.

Las interacciones internas del sistema incluyen:

- Periférico controlador y *transceiver* CAN.
- Señales discretas eléctricamente aisladas de la alimentación del sistema.
- Display LCD con protocolo I2C.
- Teclado matricial mediante IOs.
- Puerto USB empleando protocolo UART y un conversor USB - UART.

El usuario puede interactuar con el sistema de las siguientes formas:

- Display: Visualización de la información de los servomotores.
- Teclado: Navegación a través de los menúes del display.
- PC: Monitor serial para envío de configuraciones.

3.2. Selección de componentes

Previo al desarrollo del hardware y del software se determinaron los componentes a utilizar en el diseño.

En general, se buscó que los componentes tengan las siguientes características:

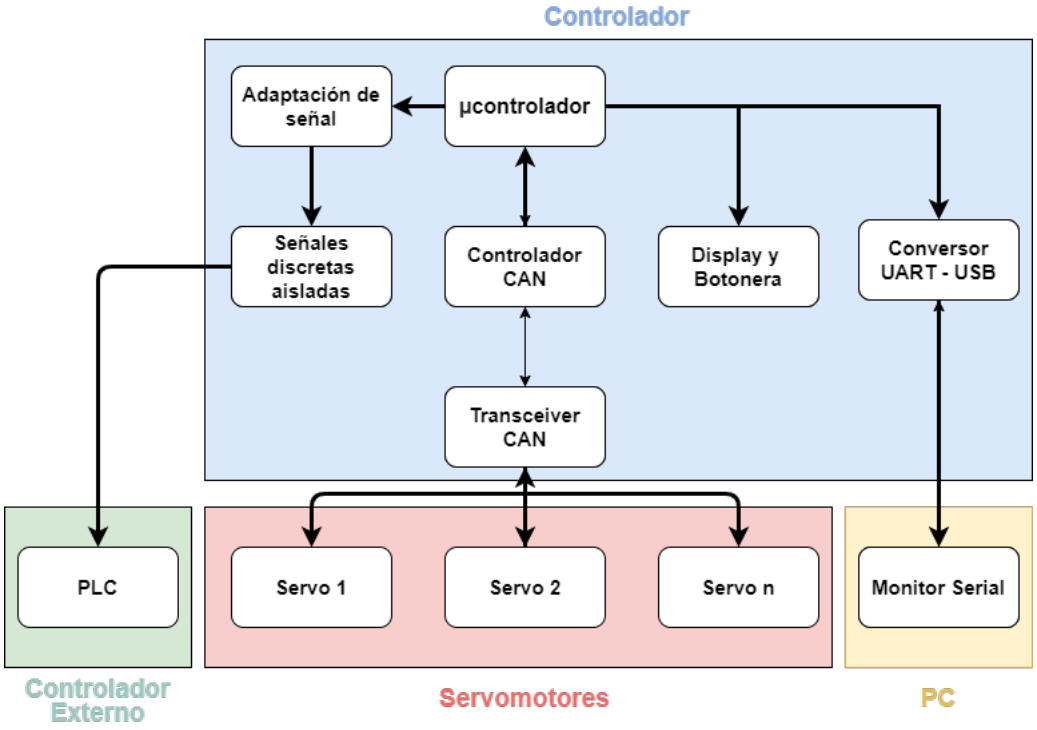

FIGURA 3.1. Arquitectura del sistema.

- Disponibilidad en el mercado.
- Facilidad para ensamble manual.
- Conocimiento previo sobre su uso.
- Bajo costo.

La disponibilidad de los componentes resultó una problemática principal en el desarrollo del proyecto debido al desabastecimiento de electrónica presente en el transcurso de la pandemia Covid-19.

Los componentes electrónicos seleccionados fueron:

- Microcontrolador ATSAMC21G18A[10]: Es el mismo modelo empleado en los sistemas SN-17, posee un controlador CAN integrado.
- Transceiver CAN LTC2875IS8[21]: También es utilizado en el sistema SN-17.
- Regulador de tensión LM2575[22]: Cumple con los requerimientos de alimentación y de temperatura del sistema.
- Interfaz USB - UART CY7C64225[23]: Posee drivers USB para interactuar con Windows.
- Optoacopladores LTV-846S[13]: Cumplen con el requerimiento de aislación eléctrica de 1 KV, cuentan con 4 canales y su *footprint* era el menor entre las opciones disponibles.

Una vez seleccionados todos los componentes principales del sistema se estudiaron sus datasheets y se seleccionaron los componentes periféricos indicados en

estos. Con esta información se completó el listado de componentes del PCB del sistema.

Los componentes de interfaz elegidos fueron:

- Display LCD de veinte caracteres y cuatro líneas con controlador HD44780 e interfaz de programación I2C (explicado en la sección 2.4.1).
- Teclado matricial de cuatro filas por cuatro columnas del fabricante Adafruit[24] (figura 3.2). Este es mecánicamente robusto para poder ser utilizado en un ambiente industrial.

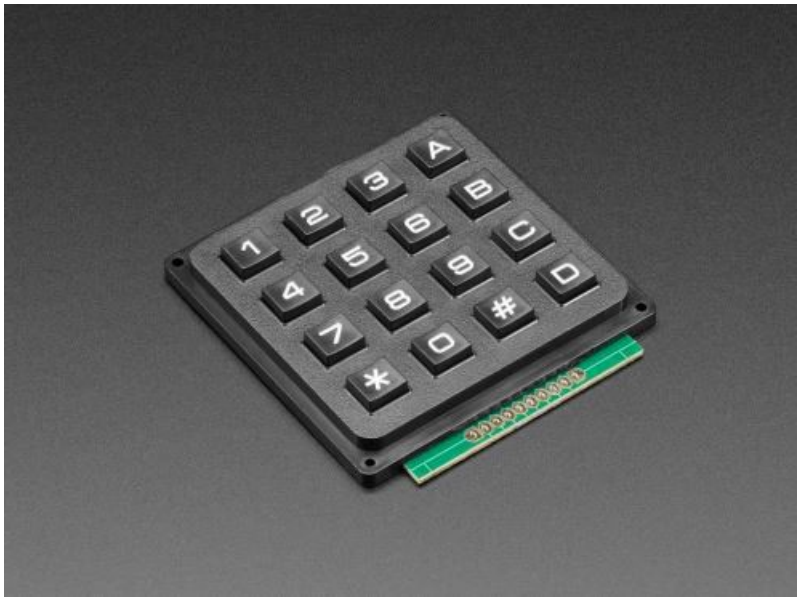

FIGURA 3.2. Teclado matricial Adafruit 4x4.

3.3. Comunicación CAN

La implementación de una red CAN requiere consideraciones de diseño relacionadas con el *bitrate* y los tiempos de bit con los cuales trabaja el protocolo para la lectura y envío de información. Como se explicó en la sección 2.2, estas variables están relacionadas con la longitud del bus y con el ruido eléctrico al que está expuesto. En la figura 3.3 se presentan los tiempos de bit recomendados para implementaciones de CANopen[4]. Del listado se seleccionó la fila para un bus de 250 m, que recomienda un *bitrate* de 250 kb/s. Estos parámetros se configuraron en el periférico CAN del microcontrolador.

La estructura empleada para la codificación de la información dentro de los mensajes CAN se muestra en la figura 3.4. Se utilizó tanto el campo de identificador estándar de 11 bits como los bytes de data para indicar el tipo de mensaje y la manera en que debe ser procesado.

Dentro del campo identificador se codificó la siguiente información:

- Bit 1: Identificador de error
- Bits 2 a 10: *Device ID*. Detalla el identificador único y fijo del dispositivo en la red.

Table 9.5 Acceptable bit rates, bus lengths, and bit timings as specified by CANopen

Bit rate	Bus length	Bit time	Time quantum	Num of quanta	Bit sample time
1Mb/s	25 m	1 it µs	125 ns	8	6
800 kb/s	50 m	1.25 it µs	125 ns	10	8
500 kb/s	100 m	2 it µs	125 ns	16	14
250 kb/s	250 m	4 it µs	250 ns	16	14
125 kb/s	500 m	8 it µs	500 ns	16	14
50 kb/s	1,000 m	20 it µs	1.25 it µs	16	14
20 kb/s	2,500 m	50 it µs	3.125 it µs	16	14
10 kb/s	5,000 m	100 it µs	6.25 it µs	16	14

FIGURA 3.3. Tiempos de bit recomendados para CANopen.

- Bit 11: Dirección. Define si un mensaje es envíado por un dispositivo controlador de red (SCI-CAN) o un dispositivo periférico (SN-17).

Se estableció el primer bit del mensaje como identificación de errores para que estos tengan la mayor prioridad en el proceso de arbitraje de la red CAN. Además, los mensajes que tienen este bit encendido no son filtrados por ningún dispositivo de la red.

En los bits que corresponden al *device ID* también se especifican identificadores comunes para mensajes del tipo *broadcast*. Estos permiten la transmisión de información a todos los nodos de la red.

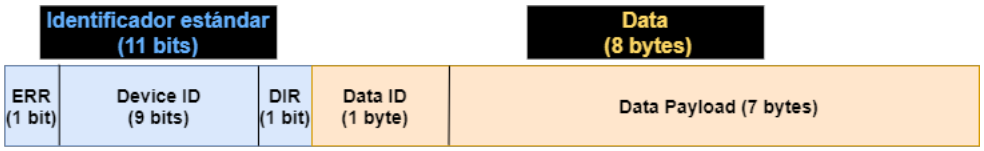

FIGURA 3.4. Estructura de mensajes CAN

El campo de data tiene una longitud total de 8 bytes. Este contiene la siguiente información:

- Byte 1: *data ID*. Identifica el tipo de información del mensaje según la tabla 3.1
- Bytes 2 a 8: *Data payload*. Contiene la información codificada según el *data ID*.

La tabla 3.1 presenta todos los *data ID* implementados en el sistema. Para cada uno de estos tipos de mensaje la información se codifica de una forma diferente.

Los tipos de mensajes implementados se dividen en:

- Instrucción: Identifican los comandos en la secuencia de un programa en los sistemas SN-17 durante su operación. Son de lectura y escritura.
- Configuración: Establecen los parámetros operativos de los sistemas SN-17. Son de lectura y escritura.
- Comando: Indican a un sistema SN-17 una acción manual a realizar. Son de solo escritura.

TABLA 3.1. Funciones de SN-17

Data ID	Tipo	Descripción
Tipo de instrucción	Instrucción	Define la instrucción
Límite de torque	Instrucción	Torque máximo de instrucción
Modo de control	Instrucción	Posición, velocidad, torque
Set point	Instrucción	Valor de lazo de control
Threshold	Instrucción	Valor de error admisible
Hold time	Instrucción	Tiempo de cumplimiento
Timeout	Instrucción	Tiempo de no cumplimiento
Guardar posición	Configuración	Guarda posición actual
Calibrar posición	Configuración	Guarda posición manual
Constantes PID	Configuración	Constantes lazo de control
Entradas y salidas	Configuración	Funcionamiento de las IO
Homing	Configuración	Establece rutina de cerado
Calibración	Comando	Inicia la rutina de calibración
Activar motor	Comando	Enciende/apaga el motor
Go Home	Comando	Inicia la rutina de cerado
Mover motor	Comando	Rota un ángulo específico
Rotar motor	Comando	Gira a velocidad específica
Torquear motor	Comando	Gira a torque específico
Límite torque manual	Comando	Limita el torque de comandos
Activar salidas	Comando	Enciende/apaga salidas
Consultar motor	Comando	Consulta información del motor
Programa manual	Comando	Corre un programa en modo maual
Instrucción actual	Supervisión	Instrucción ejecutada del motor
Encoder	Supervisión	Posición de encoder del motor
Error	Supervisión	Tipo de error del motor
Cambio de modo	General	Programación o operación
Device ID	General	Consulta de IDs de red

- Supervisión: Indican al SCI-CAN el estado de un SN-17. Son de solo lectura.
- General: Son mensajes del tipo *broadcast* para el control de la red. Son de lectura y escritura.

Los mensajes del tipo instrucción codifican un número según el *data ID* que se utilice e identifican su posición en la secuencia del programa del sistema SN-17.

Los mensajes de configuración agregan información complementaria para indicar los parámetros modificables. Por ejemplo, en el caso de una constante PID, se indica en forma codificada cual es la constante y su valor.

3.4. Desarrollo de software

3.4.1. Arquitectura de Software

El desarrollo del software se realizó sobre una placa de evaluación del microcontrolador seleccionado[25]. Esto permitió hacer pruebas del sistema previo al diseño del circuito y facilitó el proceso de implementación.

El software del sistema se diagramó siguiendo una estructura en capas. En la figura 3.5 se muestra un esquema de la arquitectura implementada. Como se puede observar, las capas intermedias interactúan con la capa superior de aplicación mediante el uso de servicios públicos que cada uno de los manejadores ofrece. Las capas inferiores (Key, LCD I2C handler, buffer circular) no son visibles por la capa de aplicación, pero ofrecen funcionalidades para las capas intermedias.

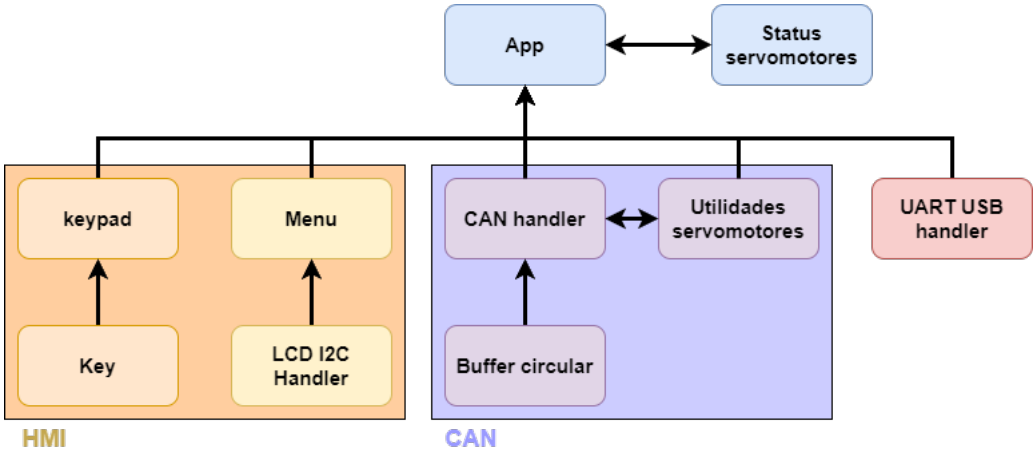

FIGURA 3.5. Arquitectura de software

La capa superior contiene a la aplicación que opera, en un mismo nivel, con una sección del código que almacena el estado de los distintos servomotores. En esta capa de software se coordina el funcionamiento e interacción de las capas intermedias.

El recuadro marcado como HMI (*Human Machine Interface*) se relaciona con la sección del programa encargada de proveer la interfaz de usuario. Maneja las acciones del teclado matricial y la información que se visualiza en el display.

La sección CAN se relaciona con las configuraciones del periférico controlador del protocolo, las funciones para codificar los mensajes, el envío y recepción de mensajes y los servicios ofrecidos por los servomotores.

El manejador de UART-USB es el encargado de las interacciones con una PC. Realiza la codificación de información de los mensajes y procesa los datos recibidos por el puerto USB.

En las subsecciones siguientes se explica el desarrollo de estos componentes principales de software.

3.4.2. Driver CAN

Para la implementación del módulo CAN handler de la figura 3.5 se utilizó el periférico de este protocolo de comunicación en el microcontrolador seleccionado. Se realizó su configuración mediante la herramienta de desarrollo *Microchip Studio* provista por el fabricante[26], a través de la cual se seleccionaron:

- Los tiempos de bit y la tasa de transmisión de 250 kb/s, según la figura 3.3.
- El uso del identificador estándar.
- El funcionamiento de los filtros.

- La opción de reenvío de mensaje en caso de colisiones.

El driver implementado opera por interrupción, con funciones de *callback* llamadas cuando se recibe un mensaje o termina una transmisión. El software de este módulo resultó de una adaptación del código de ejemplo de operación del periférico CAN provisto por el fabricante.

En la figura 3.6 se observa el flujo de programa implementado para el manejo de la recepción de los mensajes CAN.

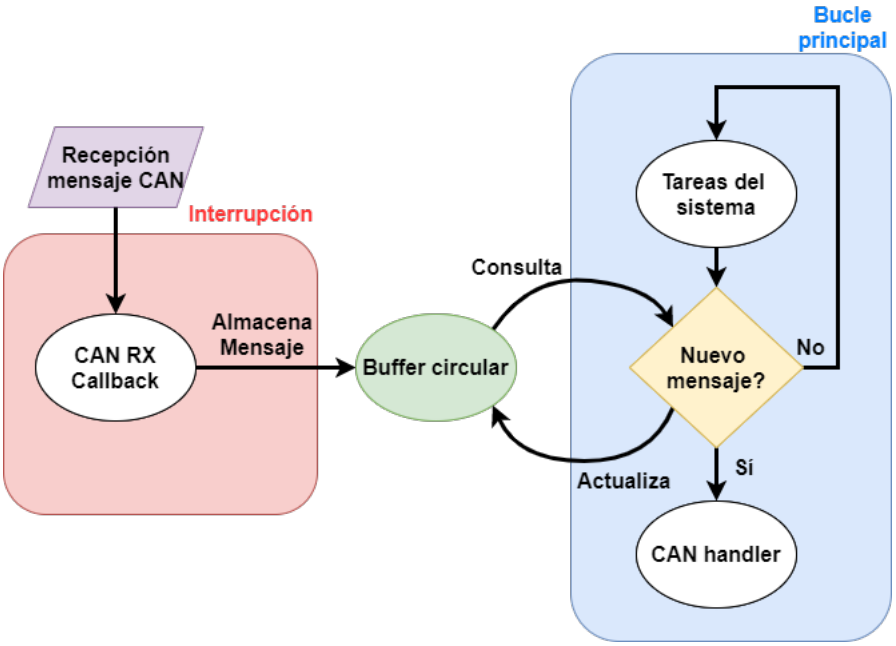

FIGURA 3.6. Flujo de recepción de mensajes CAN.

En recepción de mensajes se empleó un módulo de software de *buffer* circular[27]. La operación de recepción de mensajes CAN se hace en un estado de interrupción. Para evitar el procesamiento del mensaje en este estado se utiliza el algorítmo de *buffer* circular, que almacena el mensaje en tiempo de interrupción sin procesarlo y señaliza esta condición. Estos mensajes son luego tratados dentro del bucle principal del programa. Este funcionamiento permite una operatoria ordenada y evita la pérdida de información en caso de que nuevos mensajes lleguen al bus sin que los anteriores hayan sido procesados.

Para la interpretación de los mensajes se construyó el módulo de software marcado como utilidades servomotores en la figura 3.5. Dentro de este se implementaron distintas funciones locales que decodifican el identificador y la data según el tipo de mensaje. Con el identificador CAN se determina si el mensaje recibido indica un error (primer bit) y si está dirigido a un SN-17 o al SCI-CAN (último bit de dirección). Con respecto a la información del tipo, como se explicó en la sección 3.3, se encuentra codificada en el primer byte de data del mensaje CAN. Esta información es discriminada por el algorítmo y, utilizando una estructura *switch-case*, se selecciona la función decodificadora correspondiente.

3.4.3. Interfaz HMI

La interfaz HMI involucra la interacción del software del sistema con el teclado y con el display LCD. Su implementación implicó el desarrollo de los drivers de I2C

para trabajar con la pantalla y manejadores para el teclado. También, se desarrolló un *middleware* que controle un sistema de menúes, la interacción de ambos drivers y que ofrezca funcionalidades a la capa superior de aplicación.

La capa inferior del display se compone del driver I2C obtenido del fabricante del microcontrolador y una serie de servicios para mostrar información que son llamados por las capas superiores. Los servicios fueron modelados siguiendo ejemplos de código libre de implementaciones para Arduino[19].

El driver del teclado sigue una arquitectura similar [20]. Este usó como ejemplo implementaciones libres de Arduino[20]. Esta capa ofrece servicios a las superiores para reconocer las teclas que fueron oprimidas.

Sobre estos manejadores se desarrolló un software que utiliza a ambos para generar el sistema de menúes y su interacción con un usuario. Mediante el uso del teclado se puede navegar a través de una lista de opciones que se muestran en el display y se permite visualizar y configurar los parámetros de los motores conectados.

En la figura 3.7 se muestra un ejemplo de cómo se presentan las opciones en el display. La flecha presente a la izquierda de la imagen es la que le permite al usuario ubicarse a medida que se navega por los menúes con la utilización del teclado.

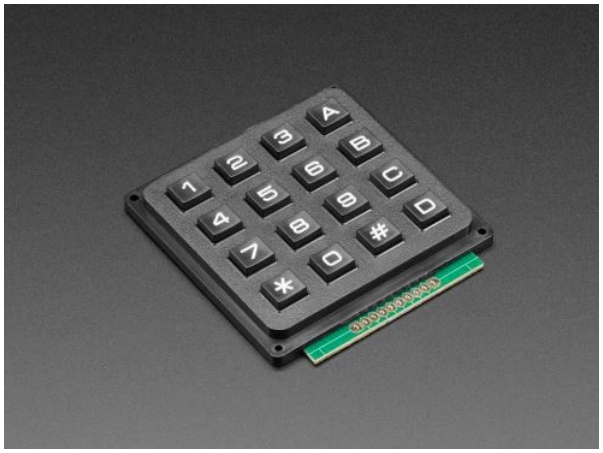

FIGURA 3.7. Ejemplo de opciones de menú - ARMAR IMAGEN

Se planteó que el menú cuente con 2 pantallas diferentes según el estado en el que esté: monitoreo o programación. En el estado de monitoreo, se muestra una primera pantalla donde puede observarse el programa e instrucción que cada motor conectado está ejecutando en ese momento y puede navegarse para consultar la configuración, pero no puede ser modificada.

En el estado de programación, la configuración de los distintos motores es accesible para ser modificada y se muestran los distintos comandos manuales que pueden ejecutarse para cada motor. Al pasar del estado de monitoreo al de programación, la central envía un mensaje de *broadcast* usando un ID prioritario a la red al que cada motor responde con su ID particular.

El software de menúes le indica a la capa superior de aplicación la acción solicitada por el usuario y es esta capa la que interactúa con los otros drivers, como se explicó en la sección 3.4.

3.4.4. Interfaz UART-USB

La interfaz UART-USB involucró el desarrollo de un manejador de UART desde el microcontrolador. La configuración de la comunicación se realizó con herramientas provistas por el fabricante. El software opera por interrupciones de hardware que llaman a funciones de *callback* al completar una transmisión o recepción de un mensaje.

Los mensajes UART son adaptados a señales USB mediante el circuito integrado CY7C64225[23]. Esto permite la conexión del dispositivo con un monitor serial en una PC.

Sobre este manejador UART se construyó una herramienta de reporte que se ejecuta en la capa de aplicación que envía las acciones seleccionadas por el menú a través de USB. Esto permite confirmar que la información seleccionada es la correcta y facilita la detección de errores.

La interfaz de PC permite también configurar las acciones de los motores. La configuración una vez que llega al SCI-CAN es transformada en comandos CAN y enviada al nodo correspondiente.

3.5. Modificaciones firmware SN-17

Para que el sistema SCI-CAN pueda comunicarse con las plaquetas SN-17 ya instaladas se debió trabajar y actualizar su firmware. Estos cambios tuvieron 3 objetivos principales:

- Incorporar los drivers de CAN y la estructura de mensajes planteada.
- Generar que las configuraciones y programas sean variables modificables.
- Lograr el almacenamiento de las configuraciones en memoria no volátil.

La implementación de CAN utilizó los mismos módulos de software desarrollados para el SCI-CAN sin modificaciones. Esto fue posible debido a que ambos sistemas utilizan el mismo microcontrolador. Siguiendo la estructura de CAN de la figura 3.5, los módulos de CAN handler y buffer circular se mantuvieron idénticos para ambos sistemas.

En la implementación del módulo de utilidades de servomotores se buscó también que ambos sistemas compartan el mismo software. Para ello, se planteó el uso de sentencias condicionales que determinen si un mensaje está dirigido a un sistema SCI-CAN o a un SN-17. A partir de esto se decide la acción a realizar en la capa de aplicación según la información del mensaje.

Para lograr que las configuraciones y los programas del sistema SN-17 sean modificables sin necesidad de cambiar el firmware se utilizó una estructura para almacenar la información. Esta es accedida directamente por la capa de aplicación, quien se encarga de administrarla. La información presente en la estructura se resume en la tabla 3.2.

Esta misma estructura se replicó en el dispositivo SCI-CAN, donde se utilizan varias instancias, una por cada nodo conectado, donde se guarda el estado reportado por los sistemas SN-17. En el esquema de la figura 3.5 el módulo Status servomotores es el que implementa la estructura descripta.

TABLA 3.2. Estructura de estado de servomotores.

Nombre	**Descripción**
CAN ID	Identificador CAN
Modo de trabajo	Operación o configuración manual
Estado manual	Estructura auxiliar para operación manual
Control PID	Información de variables de conttrol PID
Cerado	Configuraciones de cerado de motor
Posiciones memoria	Memoria de posiciones guardadas
Encoder	Última posición de encoder leída
Posición absoluta	Última posición de eje calculada
Instrucción	Instrucción actual en operación
Programa	Programa actual en operación
Salidas	Configuración de salidas discretas
Entradas	Configuración de entradas discretas
Error	Registro del último error

3.6. Desarrollo de Hardware

El desarrollo del hardware se realizó con software Altium Designer[28] y las placas fueron fabricadas por un proveedor habitual de Cambre ICyFSA[29]. Para el diseño del PCB se tuvieron en cuenta las capacidades técnicas publicadas por el fabricante[30].

Se establecieron las siguientes decisiones de diseño:

- Placa de cuatro capas con dimensiones menores a 100 x 100 mm.
- Conectores COMBICON con tornillo[31] para conexiones externas.
- Conectores XH[32] para conexiones internas.

El primer paso del diseño fue determinar los subcircuitos del sistema. Estos son:

- Microcontrolador
- Regulador de tensión
- Interfaz CAN
- Entradas discretas
- Salidas discretas
- Interfaz UART-USB

En cada uno de los subcircuitos se desarrolló un dibujo esquemático. En la figura 3.8 se presenta el correspondiente a la interfaz CAN donde se muestra el *transceiver* elegido y sus conexiones. Para cada uno de los componentes principales del sistema se siguieron las recomendaciones indicadas en su hoja de datos.

Finalizados los subcircuitos esquemáticos, se continuó con el desarrollo del PCB. Primero, se determinaron las dimensiones de la plaqueta, manteniendo las restricciones impuestas y asegurando una cómoda posición de todos los componentes para permitir un ensamble manual. Se eligió una dimensión final de plaqueta de 100 x 80 mm.

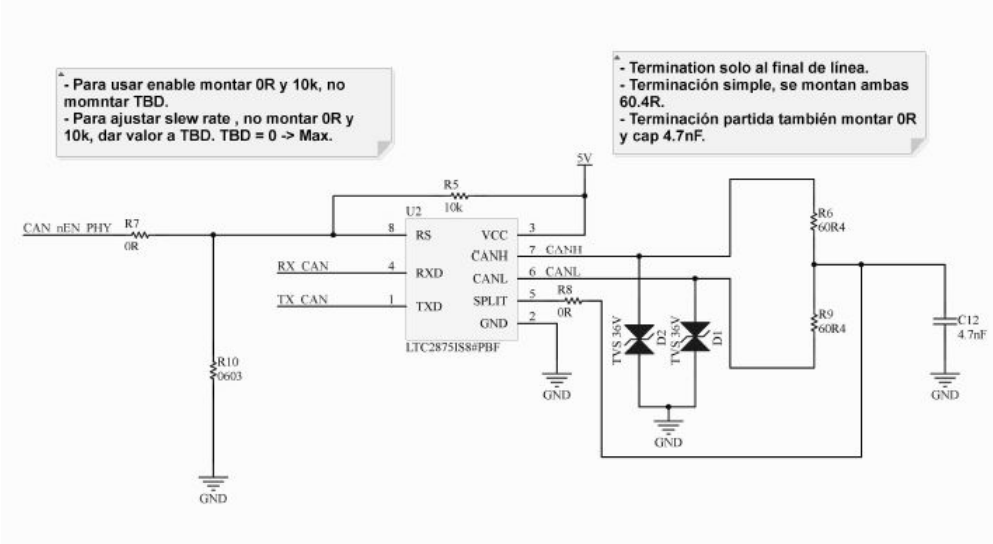

FIGURA 3.8. Esquemático de Interfaz CAN.

En el diseño del *PCB* se buscó que:

- Los subcircuitos quedaran separados.
- Los conectores se colocaran en los extermos de la plaqueta.
- Se respetara la aislación eléctrica de las entradas y salidas con el circuito de control.
- Se minimizara la longitud de las líneas de CAN y se maximizara su simetría.

La figura 3.9 muestra el render 3D del *PCB* obtenido. En la parte superior se encuentran los circuitos de entradas y salidas industriales eléctricamente aislados por los optoacopladores y separados del resto del circuito. En la esquina inferior izquierda se encuentra el regulador de tensión de 5 V. En el centro el microcontrolador y a la derecha el circuito de CAN y de USB.

3.7. Desarrollo de gabinete

El desarrollo del gabinete utilizó el software de diseño mecánico Autodesk Inventor[33]. Se decidió armar un ensamble que incluya todos los componentes y que sea posible su impresión 3D.

Para los componentes adquiridos, como la pantalla LCD y la matriz de botones se tomaron los modelos 3D de uso libre de la plataforma GrabCAD[34]. Estos se verificaron para asegurarse que sus medidas correspondieran con el dispositivo físico adquirido. El modelo de la plaqueta electrónica SCI-CAN desarrollada se exportó desde el software Altium.

Una vez obtenidos los modelos de estos componentes, se realizó el desarrollo de las distintas piezas que conforman el gabinete. Como consideraciones de diseño se planteó:

- Facilitar el acceso a los conectores de la plaqueta SCI-CAN.
- Proteger los circuitos internos de la plaqueta SCI-CAN.

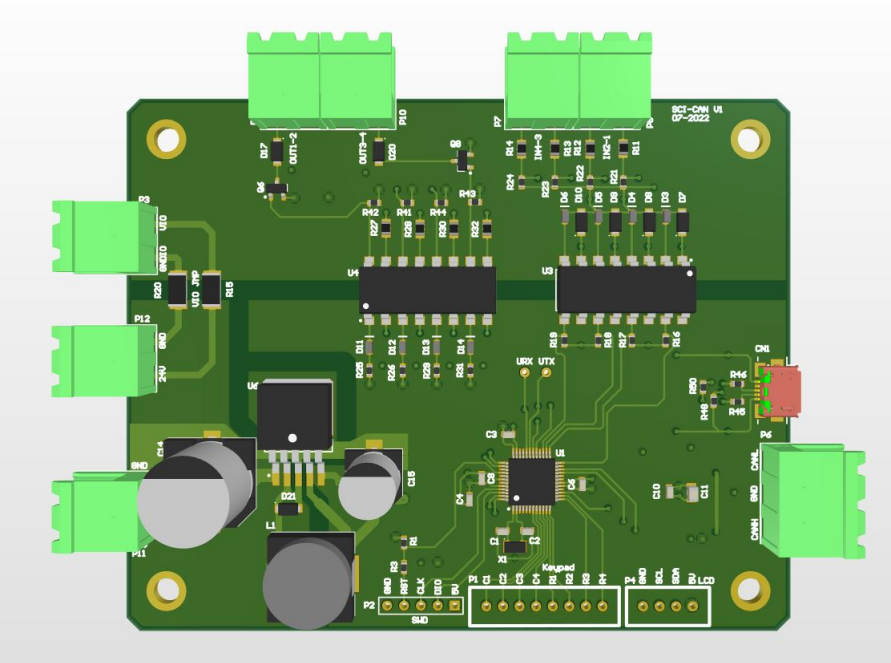

FIGURA 3.9. Render del *PCB*.

- Presentar el teclado y el display de forma contigua.
- Esconder dentro del gabinete las conexiones entre el display y el teclado con la plaqueta SCI-CAN.
- Minimizar la cantidad de partes y de ajustes necesarios.
- Limitar los ajustes a roscas métricas.

Como restricción adicional de diseño se debió mantener las dimensiones máximas de las piezas a medidas menores que la capacidad de fabricación de la impresora 3D *Creality Ender-3*[35]. Esto corresponde a 220 x 220 x 250 mm.

En la figura 3.10 se presenta una imagen del ensamble desarrollado tomada de Inventor. Es importante notar que las piezas superiores, donde están la pantalla y el teclado, se muestran transparentes para facilitar la visualización. El ensamble consta de 3 componentes: una tapa delantera donde apoyan el display y el teclado, una caja intermedia que encierra a estos, y una caja trasera donde se coloca la plaqueta de control.

En la fabricación del conjunto se empleó el programa UltiMaker Cura[36] donde se generaron los archivos de fabricación para impresión 3D. Se decidió utilizar como material PLA que es uno de los plásticos más económicos y simples de manipular y que cuenta con las características mecánicas apropiadas para la aplicación.

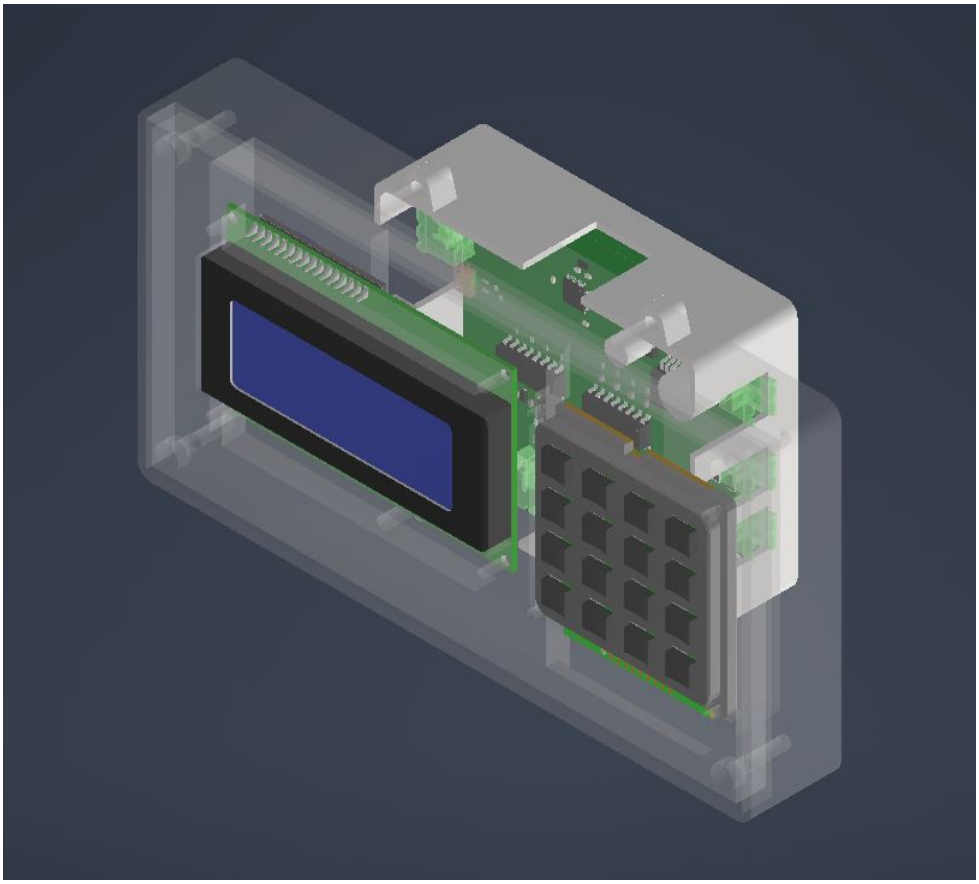

FIGURA 3.10. Ensamble 3D de gabinete.

Capítulo 4

Ensayos y resultados

En este capítulo se detallan los ensayos y mediciones realizados tanto en banco de prueba como en planta para validar los requerimientos planteados.

4.1. Banco de pruebas

En la figura 4.1 se puede observar el banco de pruebas utilizado. El osciloscopio es el modelo Hantek DSO2D10[37] y la fuente regulable es una YIHUA 305D[38]. En la PC se ejecuta un programa de monitor serial llamado PuTTy[39] que se emplea para visualizar de forma simplificada el mensaje que el sistema envía a la red CAN.

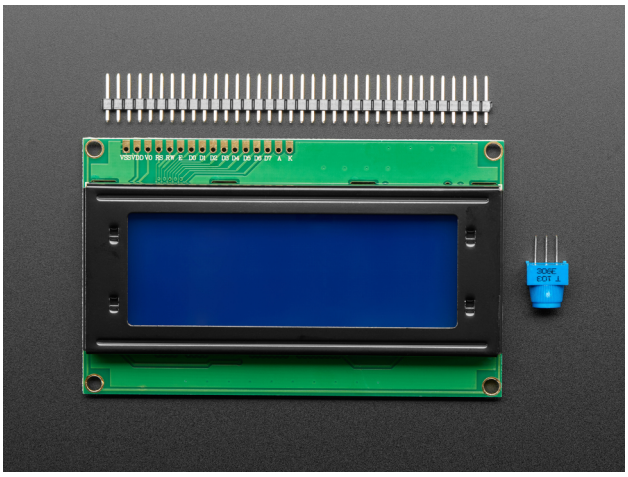

FIGURA 4.1. Banco de pruebas utilizado para verificaciones - ARAMR FIGURA

4.2. Ensayo de mensajes CAN

En la figura 4.2 se puede ver una de trama CAN medida con el osciloscopio. Se identifican las señales CAN-H y CAN-L con valores de tensión acordes a lo esperado. Es importante notar la ausencia de ruido en la señal, que indica la correcta slección de los resistores de terminación.

En la figura 4.3 se presenta la medición de tiempo tomada desde el osciloscopio para uno de los bits de un mensaje CAN. Se puede ver en la esquina superior

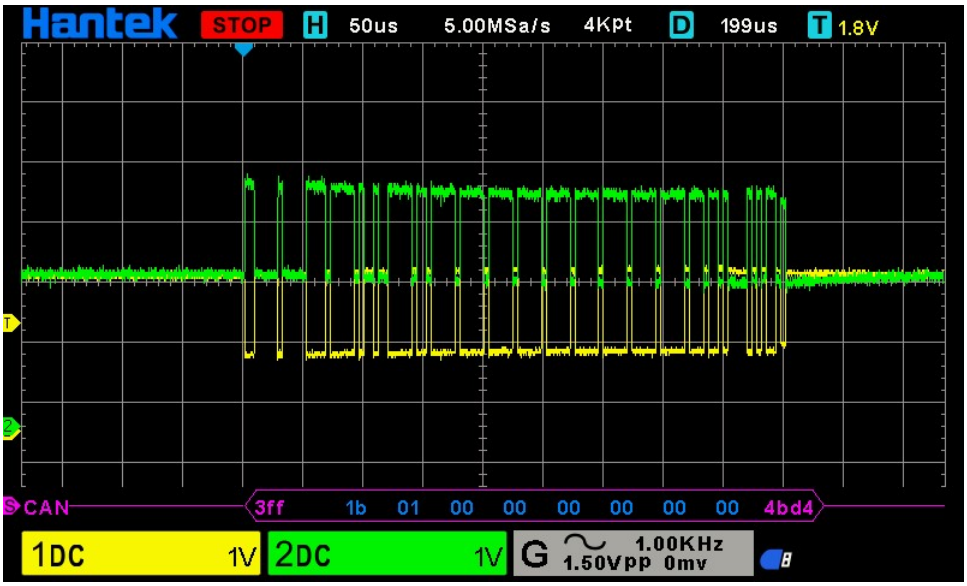

FIGURA 4.2. Niveles de señal CAN en osciloscopio

izquierda el valor de tiempo obtenido de 4.1 μs, que es lo esperado según lo explicado en la sección 3.3. Este tiempo implica una velocidad de transmisión de 250 kb/s.

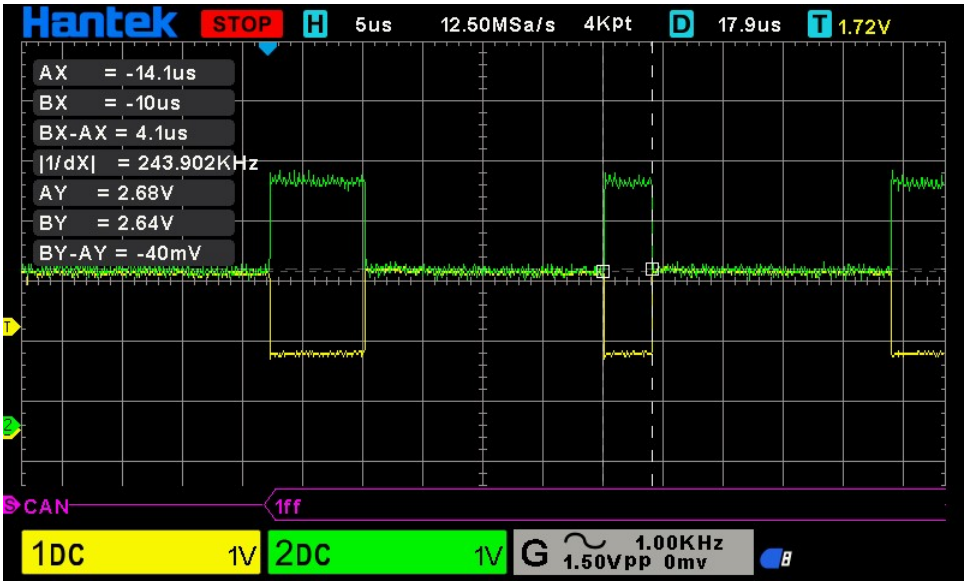

FIGURA 4.3. Medición de tiempo de bit

Los mensajes de la tabla 3.1 fueron probados uno a uno, tanto el identificador como el payload correspondiente. Se verificó que los sistemas SN-17 conectados realicen las acciones correctas.

Como ejemplo, la figura 4.4 presenta una imagen tomada del osciloscopio para la instrucción de calibrar motor (código: 0x0c). En la parte inferior puede observarse el decodificador CAN que el osciloscopio trae incorporado que marca correctamente la instrucción. También se percibe el identificador del mensaje (código: 0x0b) compuesto por el identificador del motor (0x05) y el bit final que indica que es un mensaje desde el dispositivo controlador. El resto de los bytes de data

se marcan en 0 ya que no es necesario envíar más información para este tipo de mensaje. Los últimos bytes pertenecen al CRC.

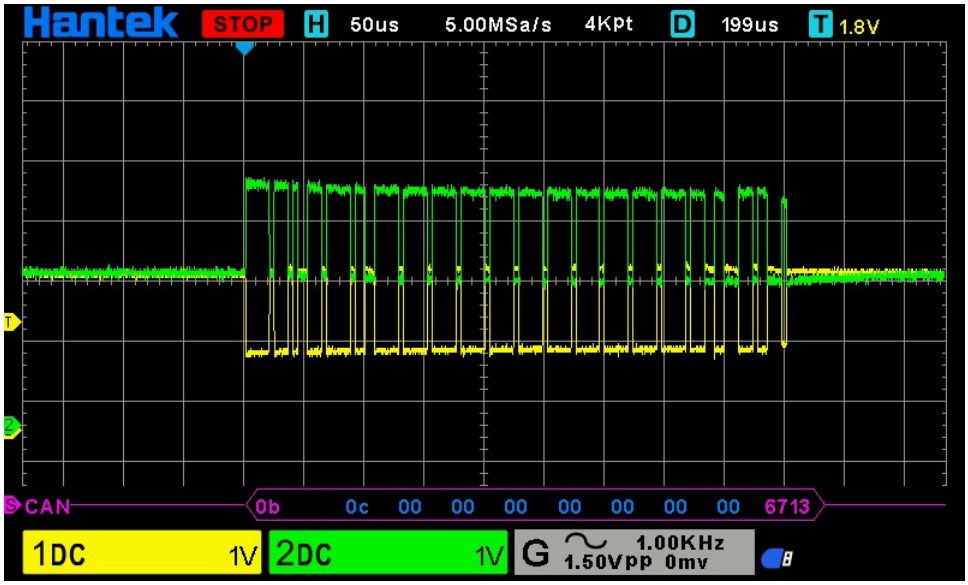

FIGURA 4.4. Instrucción calibrar motor

Otro ejemplo de mensaje puede visualizarse en la figura 4.5 en la que se envía al motor un comando manual de movimiento angular (código: 0x10), en dirección en contra de las agujas del reloj (código 0x01), un ángulo de 360 grados (códigos: 0x01 y 0x68). En este caso, como el ángulo es un número que requiere más de un byte para su representación aparece descompuesto en el mensaje. Al recibir el mensaje, el motor correctamente gira lo estipulado.

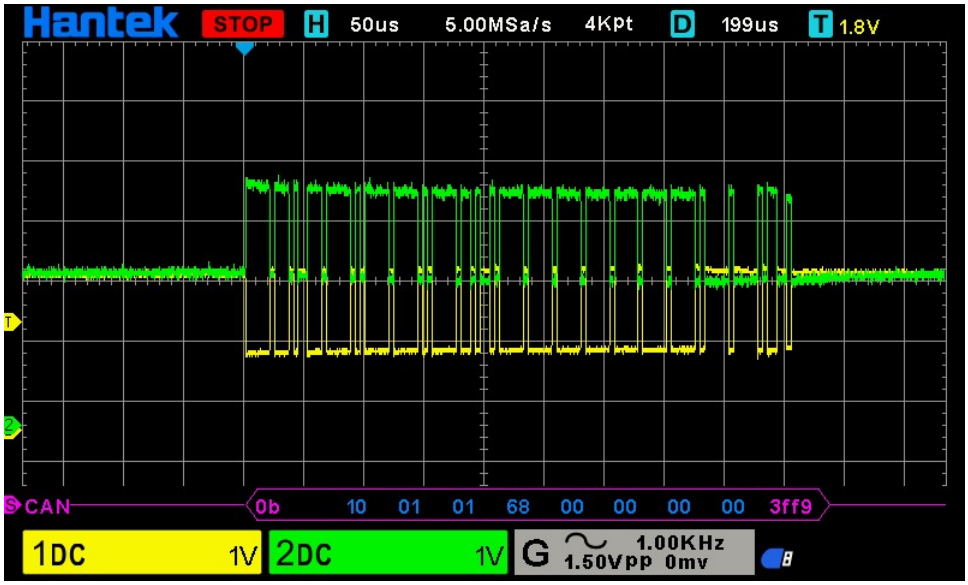

FIGURA 4.5. Instrucción manual mover motor

4.3. Ensayos eléctricos

Los ensayos eléctricos se realizaron sobre la plaqueta de control conectada directamente a la fuente de alimentación regulable configurada a 24 V y sin ningún

otro dispositivo conectado. Se hicieron mediciones utilizando un multímetro para verificar que los niveles de tensión del sistema fueran los apropiados. Las verificaciones realizadas se presentan en la Tabla 4.1.

TABLA 4.1. Verificaciones eléctricas

Ensayo	Descripción
Regulador	5 V a la salida del regulador
Referencias fuente	24 V en bornes de referencia de fuente
Referencias regulador	5 V en bornes de referencia de controlador
IOs controlador	5 V en bornes de IOs de controlador
IOs puenteadas	24 V en IOs aisladas puenteadas a tensión de entrada
IOs diferidas	IOs aisladas a 12 V con alimentación de sistema a 24 V

Las salidas PNP del sistema se ensayaron con una conexión a un PLC Omron NX102[40]. Las cuatro se conectaron a un módulo de entradas DC NX-ID5442. Se comprobó que al activar las salidas el PLC las recibe de forma correcta.

Con las entradas del sistema se realizó un procedimiento similar, conectando cada una a un módulo de salidas PNP NX-OD5256. Se activaron las salidas desde el PLC y se verificó que se recibe la señal desde el sistema.

VER DE AGREGAR UNA IMAGEN CON EL DISPOSITIVO CONECTADO AL PLC

4.4. Ensayos de mensajes UART-USB

En la medición de la comunicación con la PC se utilizó el mismo banco de medición conectando la punta del osciloscopio al bus de datos UART-USB.

En la figura 4.6 se muestra una imagen del osciloscopio con las mediciones realizadas sobre la señal de UART. Se verifica el tiempo de bit correcto de 106 μs, correspondiente a un *baudrate* de 9600 y el nível de tensión de 5 V. Se comprobó que los mensajes enviados y recibidos fueran correctos y que el sistema actuara acorde a lo esperado.

Por otro lado, se verificó que los niveles de señal USB, obtenida luego del integrado CY7C64225 fueran correctos. En la figura 4.7 se presenta la medición obtenida, donde se ve la señal diferencial correspondiente al protocolo USB en modo *full-speed*. Se puede comprobar el nivel de tensión de 3 V y el tiempo de bit de 83 ns. Esto corresponde con la especificación del dispositivo.

4.5. Pruebas de funcionamiento en planta

El dispositivo fue ensayado en la planta de Cambre ICyFSA donde se conectó en una línea de ensamble automática en desarrollo. Esta línea cuenta con varios sistemas SN-17 montados para realizar distintas actuaciones mecánicas. También, la línea cuenta con un PLC Omron NX-102 que controla el proceso. La figura 4.8 muestra el esquema de conexionado empleado para realizar las pruebas del sistema en la línea de ensamblaje.

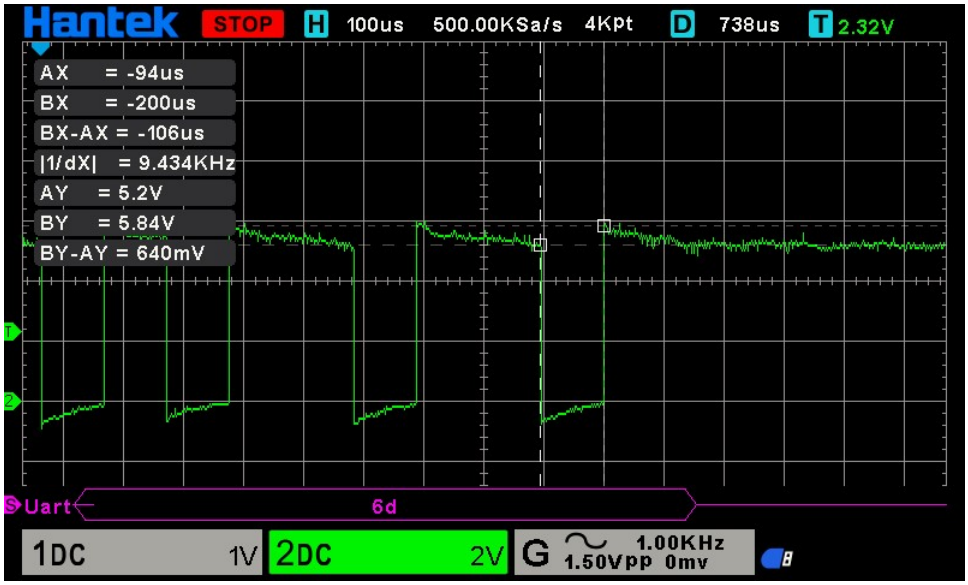

FIGURA 4.6. Medición de señal UART

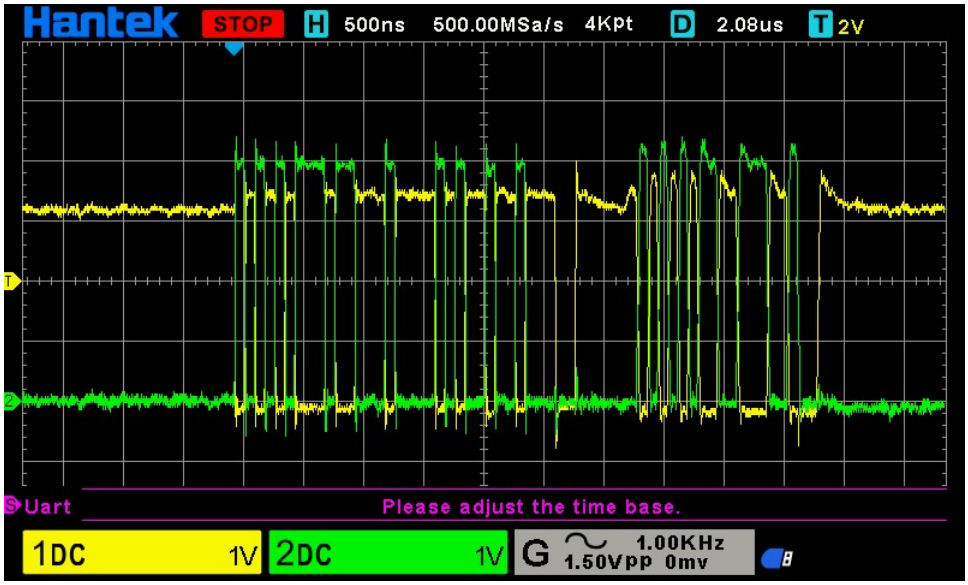

FIGURA 4.7. Medición de señal USB

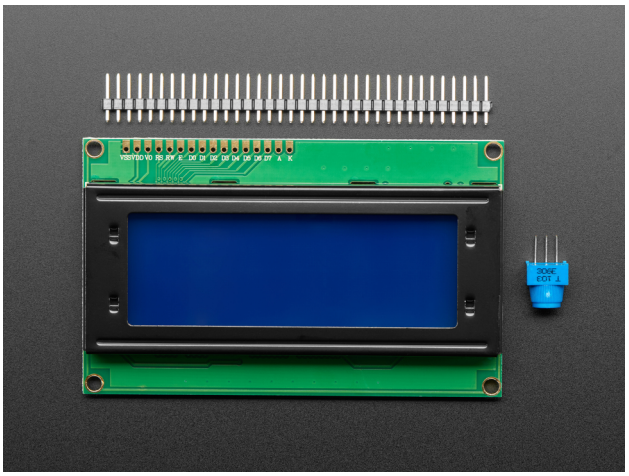

FIGURA 4.8. Esquema de conexionado en planta - ARAMR FIGURA

Con la disposición explicada se realizó la programación de los sistemas SN-17 de la línea y se monitoreó su operación. Se comprobó que el sistema muestre correctamente el estado de funcionamiento en operación de los SN-17 y el número de programa e instrucción en ejecución de cada uno de ellos. En la figura 4.9 se observa la información de monitoreo que ofrece el SCI-CAN en el display de los motores conectados en operación. El primer número indica el número de programa en ejecución por el motor, el segundo el número de instrucción de ese programa.

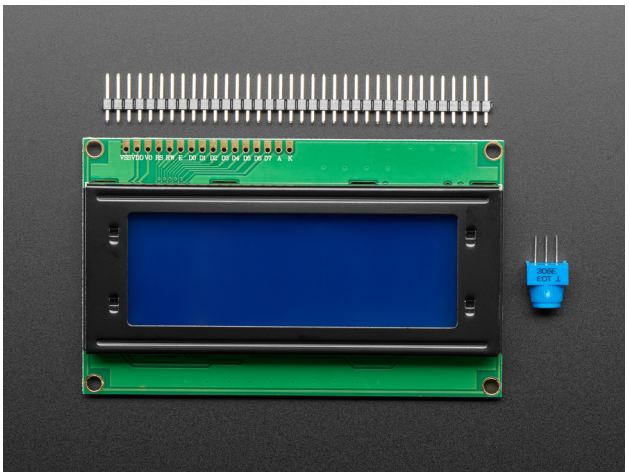

FIGURA 4.9. Monitoreo de SN-17 - ARAMR FIGURA INCLUIR ERROR

También, se verificó la comunicación a través de señales discretas con el PLC. Como se puede observar en el display del sistema de la figura 4.9, en caso de que un sistema SN-17 se encuentre en error se visualiza junto al número de instrucción. En estos casos, el sistema envía una señal al PLC para indicarle el estado de error.

En la figura 4.10 se muestra el dispositivo montado en la línea de ensamblaje automático en Cambre ICyFSA.

4.6. Comparación con estado del arte

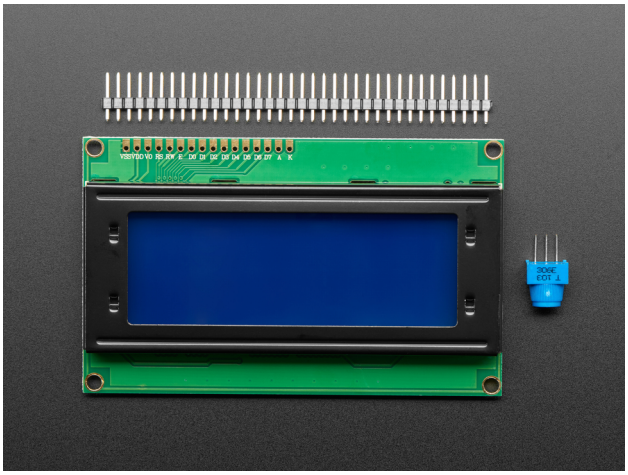

FIGURA 4.10. Sistema montado en línea de ensamble automático - ARAMR FIGURA

Capítulo 5

Conclusiones

En este capítulo se hace un repaso de las actividades realizadas y se resume la conformidad de los requerimientos del trabajo con respecto a la planificación inicial. También, se hace mención a ciertos puntos a mejorar y a los próximos pasos del proyecto.

5.1. Resultados obtenidos

En general, el proyecto cumplió con todos los requerimientos planteados en la planificación. Se agregaron puntos adicionales, relacionados con la comunicación USB para facilitar el uso del dispositivo, que fueron implementados. Se considera que el objetivo del trabajo se logró de forma satisfactoria: el sistema obtenido facilitó la interacción con los sistemas SN-17 y permite monitorear el estado de los motores en operación.

Se destaca la manifestación de uno de los riesgos especificados en la planificación: el desabastecimiento de componentes electrónicos en el mercado mundial y la dificultad para traerlos a la Argentina generó retrasos en el desarrollo del proyecto e implicó la selección de componentes alternativos. El plan de mitigación de realizar las compras de forma prioritaria fue efectivo y permitió concluir el proyecto a tiempo.

Para el correcto desarrollo del trabajo, se utilizaron los conocimientos adquiridos a lo largo del posgrado, en particular:

- Programación de microprocesadores: funcionamiento general de un microcontrolador, programación de periféricos (comunicación y timers) y máquinas de estado.
- Ingeniería de software: construcción de la arquitectura del software en capas, utilización de repositorios y definición de los servicios requeridos de cada uno de los drivers implementados.
- Protocolos de comunicación en sistemas embebidos: Utilización de diversos protocolos dentro del sistema, entre ellos, CAN, UART, SPI e I2C.
- Arquitectura de microprocesadores: Utilización de optimizaciones de compilador, cambios en linker script para almacenaje en memoria no volátil de información.
- Testing de software en sistemas embebidos: uso de herramientas de pruebas automáticas para encontrar errores en el software desarrollado.

- Diseño de circuitos impresos: recomendaciones para el desarrollo físico de la plaqueta. Utilización de reglas para desarrollo de PCB. Esquemáticos eléctricos separados por subcircuitos.
- Diseño para manufacturabilidad: selección de componentes para facilitar el ensamble, técnicas de soldadura recomendadas y consideraciones para ensamble automático.

Es importante notar que el sistema pudo ser montado y probado en una línea de montaje real en desarrollo en la planta industrial de Cambre ICyFSA. Los resultados de los ensayos realizados en esta fueron satisfactorios

5.2. Próximos pasos

Con el dispositivo implementado en las líneas productivas de Cambre ICyFSA, se le hará un seguimiento exhaustivo para corregir posibles errores y realizar posibles mejoras que puedan surgir con el uso del dispositivo.

Se plantea como trabajo futuro:

- Modificar el firmware para que trabaje con un sistema operativo, en lugar de *bare metal*.
- Generar una interfaz gráfica para PC utilizando la conexión USB implementada.
- Implementar mensajes CAN entre diferentes dispositivos SN-17, sin la interacción del SCI-CAN para generar movimientos coordinados.
- Generar identificación dinámica de dispositivos, sin necesidad de utilizar identificadores de CAN fijos.
- Implementar otros protocolos de comunicación industriales, permitiendo más diversidad de interacciones.

Bibliografía

[1] CiA. *CAN in Automation*. Visitado el 18/02/2023. 2023. URL: https://www.can-cia.org/.

[2] Keith Pazul. «Controller Area Network (CAN) Basics». En: (2018).

[3] ISO. *Road vehicles — Controller area network (CAN) — Part 2: High-speed medium access unit*. Visitado el 20/01/2023. 2023. URL: https://www.iso.org/standard/33423.html#:~:text=ISO%2011898%2D2%3A2003%20specifies,protocol%20that%20supports%20distributed%20real%2D.

[4] Marco Di Natale et al. *Understanding and Using the Controller Area Network Communication Protocol*. Springer, 2012.

[5] Frank Lamb. *Industrial Automation: Hands-On*. McGraw Hill Professional, 2013.

[6] Cambre. *Cambre*. Visitado el 19/02/2023. 2023. URL: https://cambre.com.ar/.

[7] Oriental motor. *Pneumatic Actuators vs Electric Actuators: Which is Better?* Visitado el 19/02/2023. 2023. URL: https://blog.orientalmotor.com/pneumatic-actuators-vs-electric-actuators-which-is-better.

[8] Tropical Labs. *Mechaduino*. Visitado el 22/01/2023. 2023. URL: https://tropical-labs.com/mechaduino/.

[9] NEMA. *National Electrical Manufacturers Association*. Visitado el 19/02/2023. 2023. URL: https://www.nema.org/.

[10] Microchip. *ATSAMC21G18A*. Visitado el 22/01/2023. 2023. URL: https://www.microchip.com/en-us/product/ATSAMC21G18A.

[11] Allegro microsystems. *A4954: Dual Full-Bridge DMOS PWM Motor Driver*. Visitado el 22/01/2023. 2023. URL: https://www.allegromicro.com/en/products/motor-drivers/brush-dc-motor-drivers/a4954.

[12] AMS. *AS5047D High Speed Position Sensor*. Visitado el 22/01/2023. 2023. URL: https://ams.com/en/as5047d.

[13] LITEON. *Photocoupler Product Data Sheet*. Visitado el 22/01/2023. 2023. URL: https://optoelectronics.liteon.com/upload/download/ds-70-96-0016/ltv-8x7%20series%20201610%20.pdf.

[14] Stefano Ricci y Valentino Meacci. «Simple Torque Control Method for Hybrid Stepper Motors Implemented in FPGA». En: (2018).

[15] CiA. *CANopen – The standardized embedded network*. Visitado el 19/02/2023. 2023. URL: https://www.can-cia.org/canopen/.

[16] Olaf Pfeiffer, Andrew Ayre y Christian Keydel. *Embedded Networking with CAN and CANopen*. Copperhill Media, 2008.

[17] Stamatios Manesis y George Nikolakopoulos. *Introduction to Industrial Automation*. CRC Press, 2018.

[18] Michael Margolis. *Arduino Cookbook*. O'Reilly Media Inc, 2011.

[19] johnrickman. *LiquidCrystal I2C*. Visitado el 19/02/2023. 2020. URL: https://github.com/johnrickman/LiquidCrystal_I2C.
[20] Chris–A. *Keypad*. Visitado el 19/02/2023. 2017. URL: https://github.com/johnrickman/LiquidCrystal_I2C.
[21] Analog Devices. *LTC2875 - High Speed CAN FD Transceiver*. Visitado el 22/01/2023. 2023. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/LTC2875.pdf.
[22] Texas Instruments. *LM2575 1-A Simple Step-Down Switching Voltage Regulator*. Visitado el 20/02/2023. 2023. URL: https://www.ti.com/lit/ds/symlink/lm2575.pdf?ts=1676886811918&ref_url=https%253A%252F%252Fwww.google.com%252F.
[23] Cypress. *CY7C64225 - USB-to-UART Bridge Controller*. Visitado el 04/02/2023. 2023. URL: https://ar.mouser.com/datasheet/2/196/CYPR_S_A0003298042_1-3003715.pdf.
[24] Adafruit. *Adafruit*. Visitado el 20/02/2023. 2023. URL: https://www.adafruit.com/.
[25] Microchip. *SAM C21 XPLAINED PRO EVALUATION KIT*. Visitado el 20/02/2023. 2023. URL: https://www.adafruit.com/.
[26] Microchip. *Microchip Studio for AVR and SAM Devices*. Visitado el 20/02/2023. 2023. URL: https://www.microchip.com/en-us/tools-resources/develop/microchip-studio.
[27] Gabriel Gavinowich. «Implementación de biblioteca CAN para sAPI». En: (2020).
[28] Altium. *Altium*. Visitado el 20/02/2023. 2023. URL: https://www.altium.com/.
[29] PCBWING. *PCBWING*. Visitado el 20/02/2023. 2023. URL: https://www.pcbwing.com/.
[30] PCBWING. *PCBWING capabilities*. Visitado el 02/04/2023. 2023. URL: https://www.pcbwing.com/Capability.php.
[31] Phoenix Contact. *COMBICON*. Visitado el 20/02/2023. 2023. URL: https://combicon50.phoenixcontact.com/en/combicon-50.html.
[32] JST. *XH Connector*. Visitado el 20/02/2023. 2023. URL: https://www.jst.com/products/crimp-style-connectors-wire-to-board-type/xh-connector/.
[33] Autodesk. *Inventor*. Visitado el 20/02/2023. 2023. URL: https://www.autodesk.com/products/inventor/overview.
[34] GRABCAD. *GRABCAD*. Visitado el 20/02/2023. 2023. URL: https://grabcad.com/.
[35] Creality. *Ender-3*. Visitado el 20/02/2023. 2023. URL: https://www.creality.com/products/ender-3-3d-printer.
[36] UltiMaker. *Cura 3D*. Visitado el 20/02/2023. 2023. URL: https://ultimaker.com/software/ultimaker-cura.
[37] Hantek Electronics. *DSO2000 Series*. Visitado el 02/04/2023. 2023. URL: http://hantek.com/products/detail/17182.
[38] YIHUA. *Fuente de alimentación DC YIHUA-3010D*. Visitado el 02/04/2023. 2023. URL: http://yihuasoldering.com/product-4-2-30v-dc-power-supply/160008/.
[39] PuTTy. *PuTTy*. Visitado el 02/04/2023. 2023. URL: https://www.putty.org/.

[40] Omron. *Machine Automation Controller NX1*. Visitado el 02/04/2023. 2023. URL: https://www.ia.omron.com/data_pdf/cat/nx1_p130-e1_dita_4_4_csm1063211.pdf?id=3705.