CARRERA DE ESPECIALIZACIÓN EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

SCI-CAN: 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

Resumen I

1. Introducción general 1
- 1.1. Controller area network . . . 1
- 1.2. Servomotores . . . 1
- 1.3. Proyecto SN-17 . . . 2
- 1.4. Motivación . . . 3
- 1.5. Estado del arte . . . 4
- 1.6. Objetivos y alcance . . . 5

2. Introducción específica 9
- 2.1. Especificaciones SN-17 . . . 9
- 2.2. Características del protocolo CAN . . . 11
- 2.3. Entradas y salidas de controladores industriales . . . 14
- 2.4. Características de componentes electrónicos empleados . . . 15
  - 2.4.1. Pantallas LCD y conversores I2C . . . 15
  - 2.4.2. Matriz de botones . . . 16
  - 2.4.3. Conversores UART-USB . . . 16

3. Diseño e implementación 19
- 3.1. Arquitectura del sistema embebido . . . 19
- 3.2. Selección de componentes . . . 19
- 3.3. Comunicación CAN . . . 21
- 3.4. Desarrollo de software . . . 24
  - 3.4.1. Arquitectura de Software . . . 24
  - 3.4.2. Driver CAN . . . 25
  - 3.4.3. Interfaz HMI . . . 25
  - 3.4.4. Interfaz UART-USB . . . 26
- 3.5. Modificaciones firmware SN-17 . . . 27
- 3.6. Desarrollo de Hardware . . . 28
- 3.7. Desarrollo de gabinete . . . 30

4. Ensayos y resultados 33
- 4.1. Banco de pruebas . . . 33
- 4.2. Ensayo de mensajes CAN . . . 33
- 4.3. Ensayos eléctricos . . . 35
- 4.4. Ensayos de mensajes UART-USB . . . 37
- 4.5. Pruebas de funcionamiento en planta . . . 37
- 4.6. Comparación con estado del arte . . . 40

5. Conclusiones 41
- 5.1. Conclusiones generales . . . 41
- 5.2. Próximos pasos . . . 42

Índice de figuras

1.1. Esquema de red CAN[1] 2
1.2. Partes de un servomotor[2] 2
1.3. Plaqueta SN-17 3
1.4. Actuador lineal con SN-17 4
1.5. Servomotor AC con *driver*[3] 5
1.6. *Closed loop stepper* con *driver*[4] 6
1.7. Proyecto Mechaduino[5] 6

2.1. Dimensiones NEMA 17[6] 9
2.2. Modelo de capas OSI[7] 12
2.3. Esquema de red CAN con resistores de terminación[8] 12
2.4. Trama CAN estándar[9] 13
2.5. Circuito NPN[**Introduction_Industrial_Automation**] 14
2.6. Pantalla LCD 20x4[10] 15
2.7. Conexión de matriz de botones 4x3[**Arduino_Cookbook**] 16

3.1. Arquitectura del sistema 20
3.2. Teclado matricial Adafruit 4x4[**web_keypad**] 21
3.3. Tiempos de bit recomendados para CANopen 22
3.4. Estructura de mensajes CAN 22
3.5. Arquitectura de software 24
3.6. Ejemplo de opciones de menú - ARMAR IMAGEN 26
3.7. Esquemático de Interfaz CAN 29
3.8. Render del *PCB* 29
3.9. Ensamble 3D de gabinete 31

4.1. Banco de pruebas utilizado para verificaciones - ARAMR FIGURA 34
4.2. Niveles de señal CAN en osciloscopio 34
4.3. Medición de tiempo de bit 35
4.4. Instrucción calibrar motor 36
4.5. Instrucción manual mover motor 36
4.6. Medición de señal UART 38
4.7. Medición de señal USB 38
4.8. Esquema de conexionado en planta - ARAMR FIGURA 39
4.9. Monitoreo de SN-17 - ARAMR FIGURA INCLUIR ERROR 39
4.10. Sistema montado en línea de ensamble automático - ARAMR FIGURA 40

5.1. Línea de ensamble automática con SCI-CAN - ARAMR FIGURA . 42

Índice de tablas

1.1. Comparación de actuadores neumáticos y eléctricos 3
1.2. Estado del arte 7

2.1. Operaciones SN-17 11

3.1. Operaciones SN-17 23

4.1. Verificaciones eléctricas 37

Capítulo 1

Introducción general

En esta sección se da una introducción al protocolo CAN o *Controller Area Network*[**web_cia_can**] 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 [**can_basics_microchip**] orientado originalmente a la industria automotriz, que se popularizó y hoy es empleado en diversas industrias. En el año 1991 Bosch publica la especificación CAN 2.0 y, en el año 1993, es adoptado internacionalmente bajo la norma ISO 11898 [**web_ISO_CAN**].

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[**Understanding_CAN**]:

- 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 y CAN Low. Al comienzo de cada mensaje se emplea 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 5 nodos.

1.2. Servomotores

Un servomotor[**Industrial_Automation_Hands_On**] 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

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

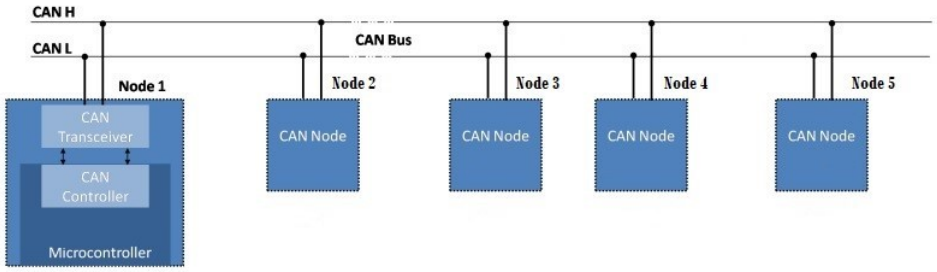

FIGURA 1.1. Esquema de red CAN[1]

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 si es o no un servomotor, sino que cuente con las cualidades de control descriptas. En la Figura 1.2 se pueden observar las distintas partes que conforman a un servomotor.

FIGURA 1.2. Partes de un servomotor[2]

1.3. Proyecto SN-17

El proyecto SN-17 (Servo NEMA 17) es un sistema desarrollado por la organización 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, generando un lazo cerrado y logrando 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, donde se lo conecta.

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

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

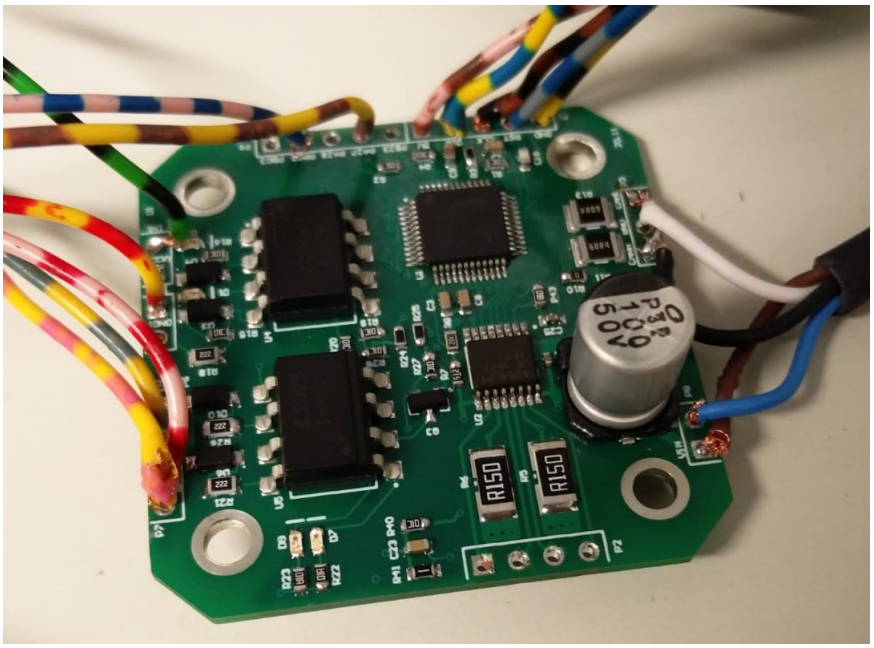

FIGURA 1.3. Plaqueta SN-17

Actualmente se está trabajando en implementar estos sistemas en la planta de la empresa Cambre ICyFSA[**web_cambre**], 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[**web_comparacion_actuadores**]. En la Tabla 1.1 se muestra una comparativa de ambos tipos de actuadores.

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

Los actuadores eléctricos suelen emplear un servomotor en su interior que, junto con un mecanismo, generan el movimiento deseado con una precisión superior a

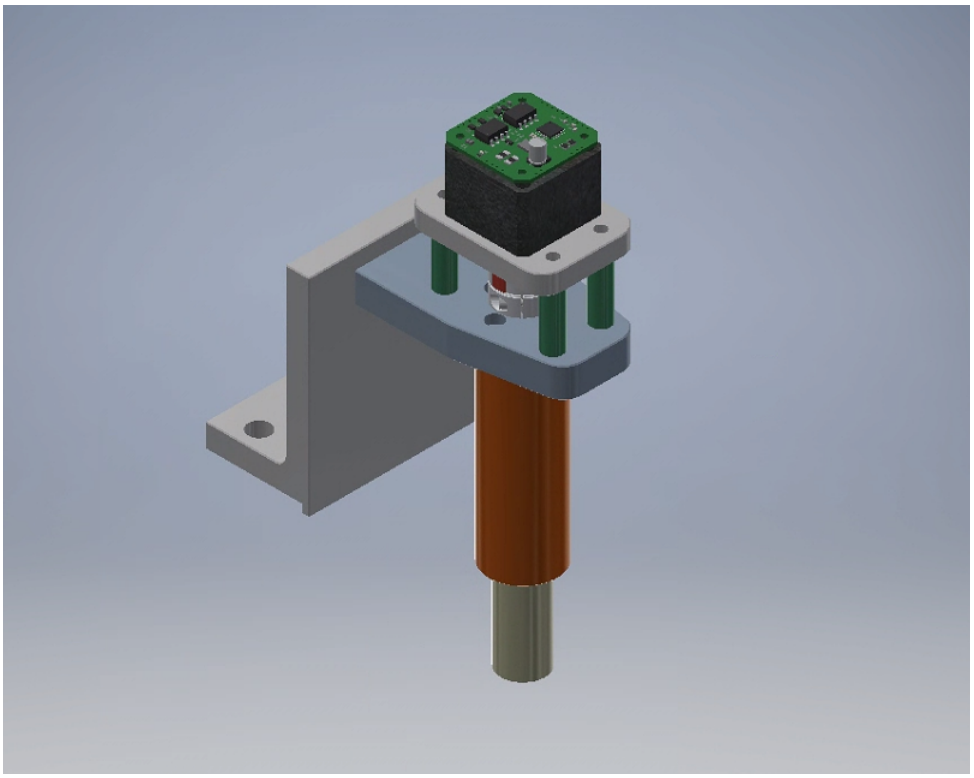

FIGURA 1.4. Actuador lineal con SN-17

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, con la utilización de los sistemas surgió la problemática de la dificultad para reprogramarlos en momentos de necesidad. La motivación de este trabajo es brindar una interfaz de programación simplificada para una red de motores que pueda ser utilizada por personal no especializado.

1.5. Estado del arte

Actualmente existe una gran oferta de servomotores y motores paso a paso. En general, lo que en el ámbito industrial se refiere a servomotores**[Industrial_Automation_Hands_On]** son motores del tipo DC *brushless* o AC de inducción (monofásicos y trifásicos). Estos tienen un valor significativamente mayor que los *steppers* y suelen requerir de *drivers* externos y, en caso de motores AC, variadores de frecuencia, convirtiéndolos en soluciones de alto costo. En los casos que se requiere alta potencia o muy alto rendimiento, estos son la mejor opción. Suelen requerir de técnicos muy capacitados para su programación y ofrecen interfaces de usuario avanzadas. 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*[3]

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 abierto. Requieren el uso de un *driver* externo para su operación y su programación suele hacerse con programas 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.

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 torques constantes. El problema principal de estas soluciones es que suelen tener características de *hobbista* y no son aplicables en ámbitos industriales. Además, requieren 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* [**web_mechaduino**].

En la Tabla 1.2 se resume la información explicada en esta sección.

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

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

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

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

FIGURA 1.6. *Closed loop stepper* con *driver*[4]

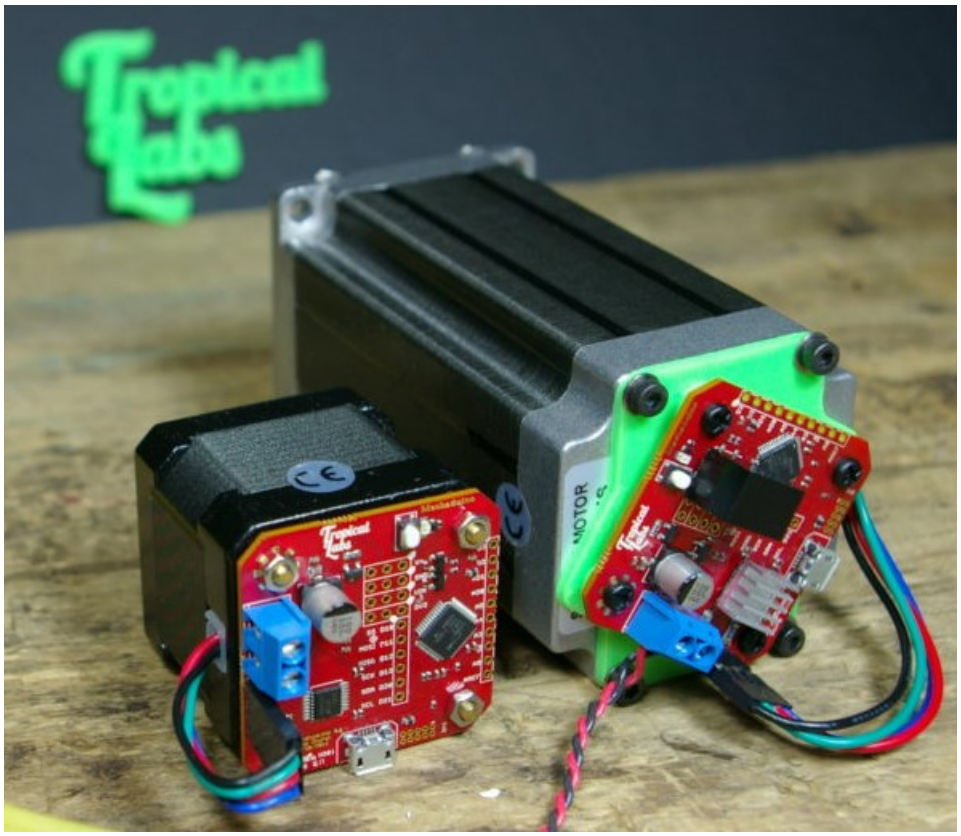

FIGURA 1.7. Proyecto Mechaduino[5]

TABLA 1.2. Resumen de características de servomotores

Motor	**Lazo de control**	***Driver***	**Programación**	**Industrial**	**Precio (USD)**
Servomotor	Cerrado	Externo	Compleja	Sí	>1.500
Stepper open loop	Abierto	Externo	Intermedia	Sí	300
Stepper closed loop	Cerrado s/torque	Externo	Intermedia	Sí	400
Mechaduino	Cerrado	Interno	Compleja	No	200
SN-17 + interfaz	Cerrado	Interno	Simple	Sí	300

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, se presentan algunas características de las entradas y salidas de los controladores industriales PLCs y se explican los distintos componentes seleccionados para el trabajo y su funcionamiento.

2.1. Especificaciones SN-17

El sistema SN-17 se desarrolló para controlar motores *stepper* pequeños que sigan los estándares NEMA (*National Electrical Manufacturers Association*)[**web_nema**]. Estos son estándares ampliamente utilizados para este tipo de motores, que indican las dimensiones que deben tener.

El sistema SN-17 consiste en una plaqueta electrónica que se monta en la parte trasera del motor. Las dimensiones de estas plaquetas permiten el montaje directo con motores del tipo NEMA-17. En la Figura 2.1 se presentan las dimensiones de los motores que cumplen con el estándar. Mediante una adaptación mecánica, también se puede emplear la plaqueta para controlar motores de menor o mayor tamaño, siempre y cuando sean del tipo *stepper* y no requieran corrientes mayores a 2 A.

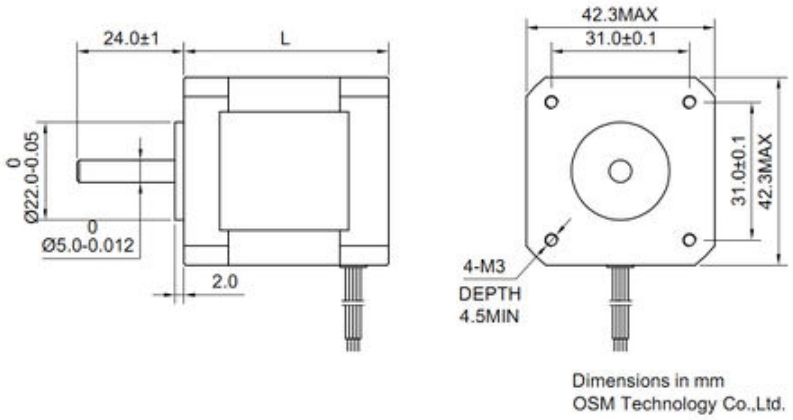

FIGURA 2.1. Dimensiones NEMA 17[1]

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, las entradas y salidas industriales.

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

- Regulación de 5 V para electrónica interna.
- Microcontrolador ATSAMC21[**web_ATSAMC21G18A**] con *core ARM Cortex-M0+* y periférico controlador de CAN.
- *Driver* de motor del tipo doble puente H[**web_A4954**].
- *Encoder* magnético absoluto[**web_AS5047D**] 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[**web_optoacopladores_LTV**], con alimentación separada, para interacción con controladores industriales.

El sistema funciona mediante un ciclo de control de lazo cerrado del tipo PID[**paper_PID_steppers**] 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 que luego es utilizada en operación.

En una capa superior al control mencionado corre 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 (*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 también configuraciones que se le pueden aplicar al motor. 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.

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

2.2. Características del protocolo CAN

Como se trató en el capítulo 1, el protocolo CAN está estandarizado bajo la norma internacional ISO 11898[**web_ISO_CAN**]. El estándar abarca solamente las capas física y de 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[**web_canopen**], del cual se tomaron ideas para este trabajo, pero no se implementa en si.

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, como puede visualizarse en la Figura **??**. El valor de esta resistencia depende de muchas variables, como 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 estás especificados por la norma, aunque existen recomendaciones[**Embedded_Networking_CAN**].

Las señales usadas 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 varios dispositivos estén conectados y puedan hablar al mismo tiempo y que se puedan detectar colisiones[**Embedded_Networking_CAN**]. En un estado recesivo, las líneas CAN-H y CAN-L están al mismo nivel de tensión. En un estado dominante, CAN-L baja mientras que CAN-H sube generando una diferencia entre ambos.

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,

[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/

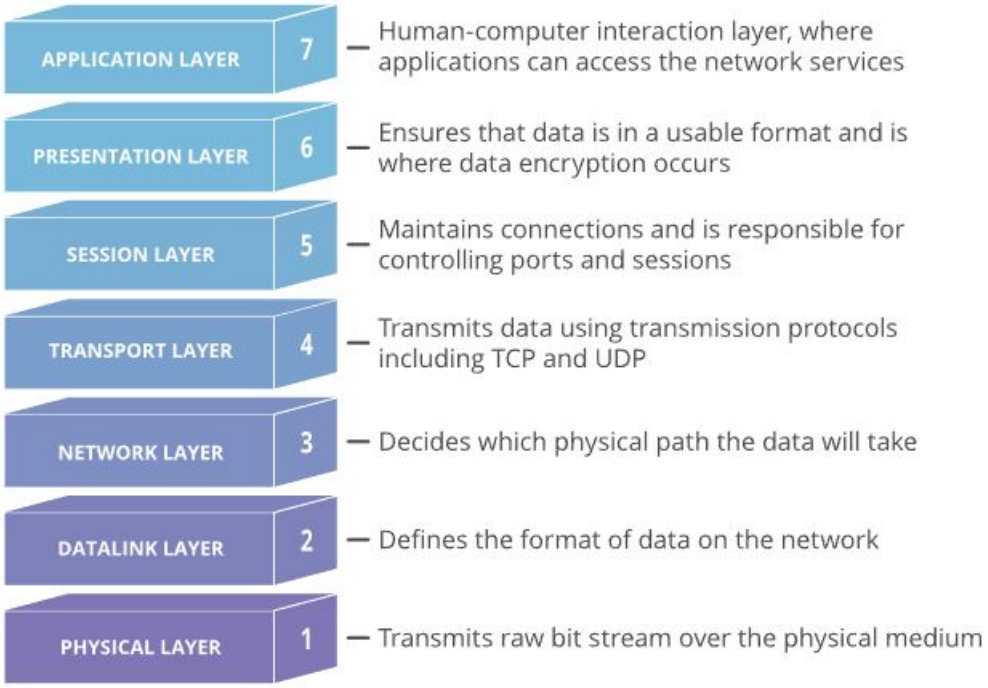

FIGURA 2.2. Modelo de capas OSI[2]

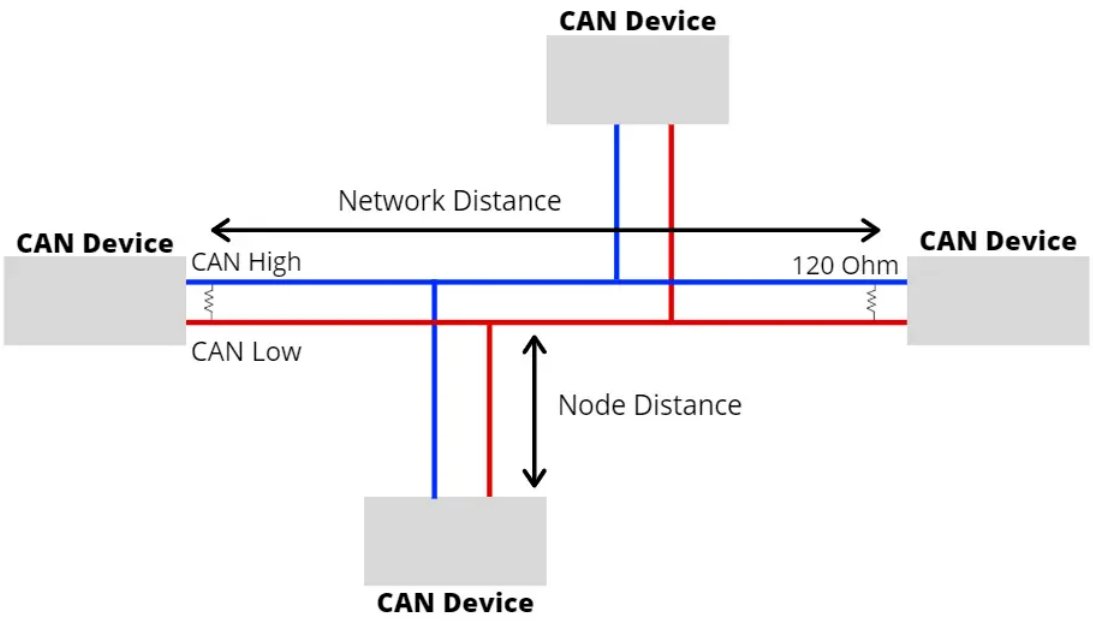

FIGURA 2.3. Esquema de red CAN con resistores de terminación[3]

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:

- Comienzo de trama - **SOF**
- Arbitraje
- Control
- Data a envíar
- Verificación
- Reconocimiento - *Acknowledge*
- Fin de trama - **EOF**

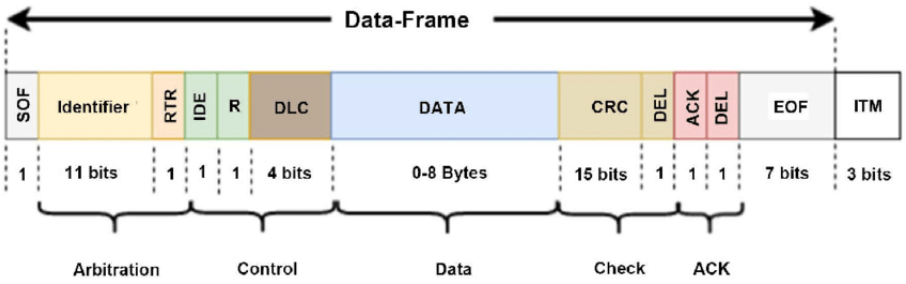

FIGURA 2.4. Trama CAN estándar[4]

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 2 o más nodos quieran 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. Esta funcionalidad se consigue gracias a la lógica AND del circuito del bus. Los nodos que no logran transmitir su mensaje debido a la prioridad inferior pueden reconocer esta situación y retransmitir el mensaje una vez que el bus esté 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 que tendrá el mensaje, pudiendo ser entre 0 a 8.

Luego se encuentran los campos de verificación y reconocimiento. La verificación está compuesta por el campo CRC, que permite comprobar que la información transmitida es igual a la información recibida, es decir, que no hayan ocurrido errores de transmisión. Luego de esto, viene el sector de reconocimiento, donde los receptores reconocen que el mensaje se ha recibido correctamente enviando un bit dominante. La trama termina con una sección de fin de trama.

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

En la actualidad, se comercializan una gran cantidad de controladores CAN. La gran mayoría ofrece todas las funcionalidades descriptas, que son lo requerido por el estándar. Según la aplicación, se pueden seleccionar controladores que tengan distinto número de filtros y máscaras para identificación de mensajes, tamaño de *buffer* de mensajes, velocidad de operación, entre muchas otras. También, muchos microcontroladores incluyen un periférico de CAN, que suele ser una solución conveniente para implementar redes de este protocolo.

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[**Introduction_Industrial_Automation**]. Normalmente, en el ámbito de la electrónica industrial de control se emplean 24 V de corriente contínua, aunque también es común encontrar 48 V DC o corrientes AC de línea y, en casos poco frecuentes, pueden aparecer otros valores. Los controladores industriales suelen estar preparados para trabajar con la mayoría de estas condiciones.

Uno de los circuitos de acondicionamiento de salidas de PLC se puede observar en la Figura 2.5. En este caso 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.

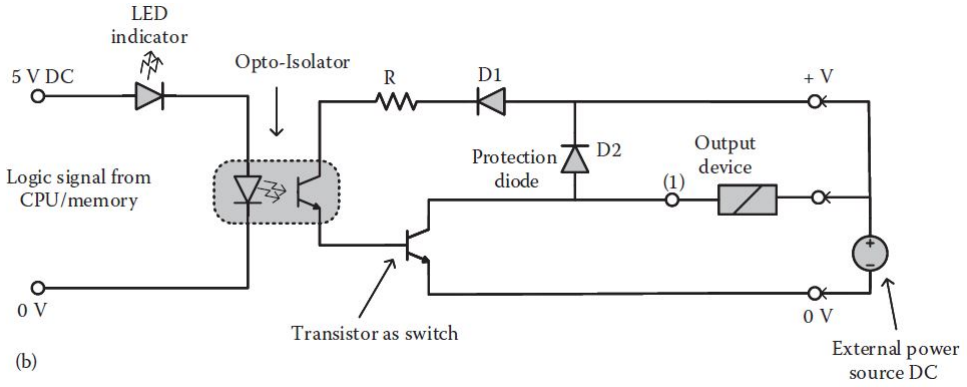

FIGURA 2.5. Circuito NPN[**Introduction_Industrial_Automation**]

Para especificar un módulo de entradas o de salidas se deben determinar los rangos de voltaje y corriente de operación, el consumo máximo de corriente, los valores de tensión lógicos, 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. Para los proyectos de electrónica, uno de los modelos más populares es el panel de texto basado en el Hitachi HD44780[**Arduino_Cookbook**]. En la Figura **??** se muestra un display de 4 líneas y 20 caracteres por línea. Estos dispositivos solo pueden representar caracteres, no pueden hacer dibujos ni gráficos, por lo tanto las interfaces que se obtienen son sencillas.

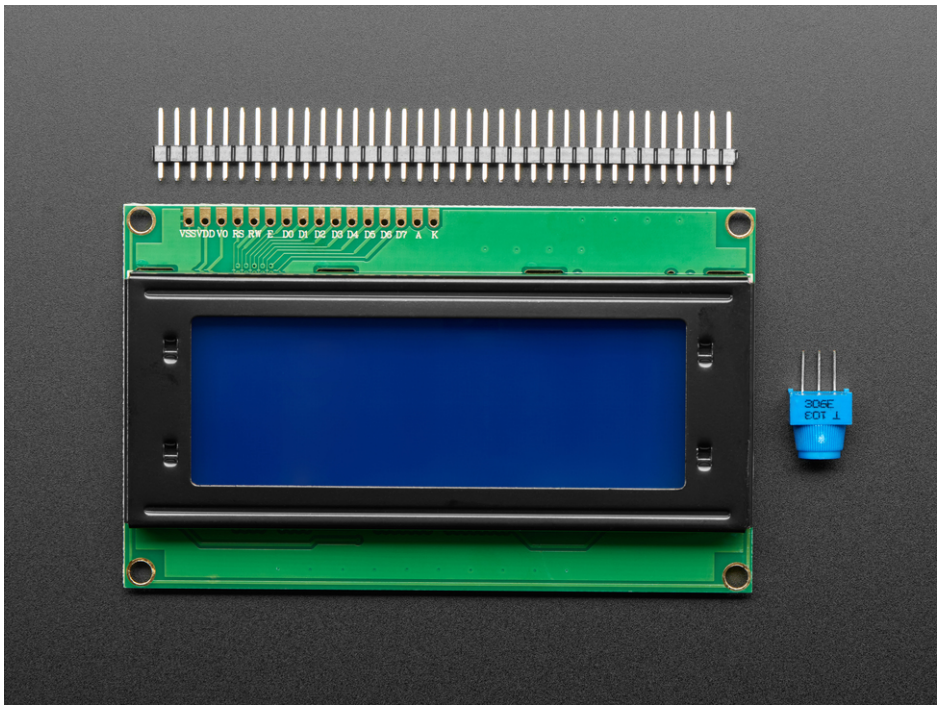

FIGURA 2.6. Pantalla LCD 20x4[5]

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 para modificar la configuración del dispositivo o mostrar texto, según se requiera.

La mayoría de microcontroladores que se emplean en la actualidad poseen un periférico de I2C y sus herramientas de desarrollo proporcionan drivers de este protocolo. I2C se caracteriza por ser del tipo serial y sincrónico. Utiliza 2 líneas de información, SDA y SCL, para transmitir la data y el reloj, respectivamente. En un bus I2C, los distintos dispositivos conectados tienen un código identificador que se emplea para controlar las comunicaciones.

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 puede ayudar a reducir significativamente los tiempos de

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

desarrollo y es el camino que se tomó en este trabajo. En particular, se utilizó la librería *LiquidCrystal_I2C* para Arduino[**web_repo_display_i2c**].

2.4.2. Matriz de botones

Las matrices de botones consisten en un conjunto de contactos normalmente abiertos que conectan una fila con una columna cuando se presionan. La metodología de uso consiste en conectar las filas a pines de entrada con resistores de pull-up en el microcontrolador y las columnas a pines de salida. La secuencia que se realiza es poner el nivel de tensión de una columna por vez en 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)[**Arduino_Cookbook**]. En la Figura 2.7 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.

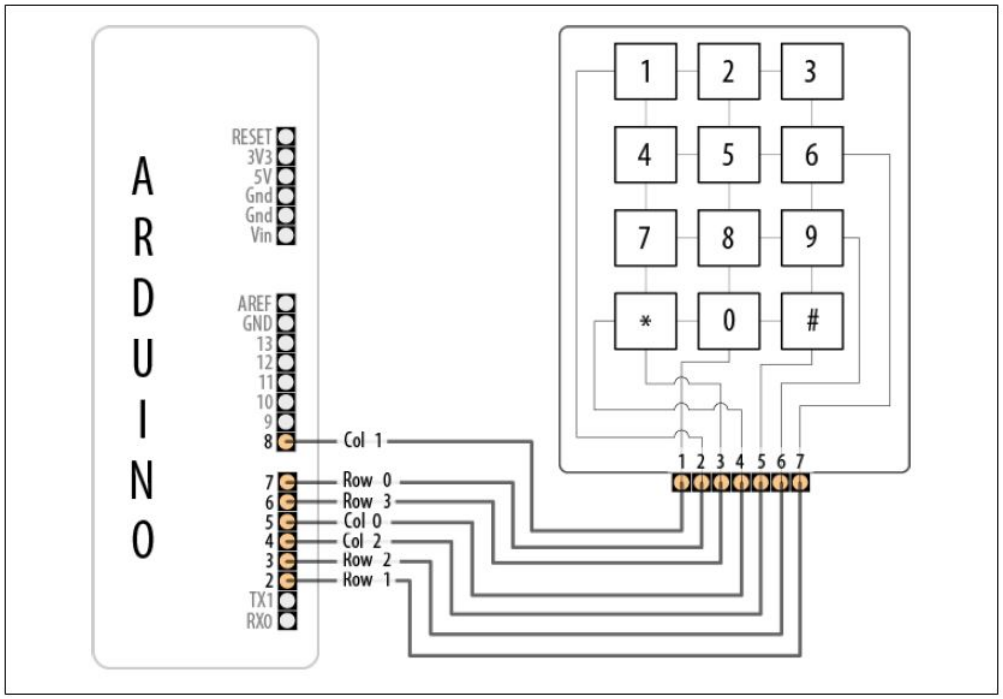

FIGURA 2.7. Conexión de matriz de botones 4x3[**Arduino_Cookbook**]

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 lo que se hizo en este trabajo. En particular, se utilizó la librería Keypad para Arduino[**web_repo_keypad**].

2.4.3. Conversores UART-USB

El *Universal Asynchronous Receive Transmit* o UART es un protocolo serial de comunicación muy utilizado en sistemas embebidos. Se caracteriza, principalmente, por su sencillez. Como su nombre indica es un protocolo asincrónico (sin línea de clock) que requiere solamente 1 línea para envíar data, o 2 para envíar y recibir.

La mayoría de los microcontroladores que se comercializan en la actualidad disponen de uno o más periféricos de 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 distintos sistemas operativos y son reconocidos por estos, por lo cual solo es necesario conectarlos a través del puerto USB. Desde la PC, cualquier programa de monitoreo de puertos seriales puede usarse 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, facilitando 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, y 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 azul, marcado como *controlador*, es lo que corresponde al desarrollo de este trabajo. En los recuadros externos están los dispositivos con los que este sistema debe interactuar: PLCs, servomotores SN-17 y PCs. Con los PLCs, la interacción es a través de señales discretas que necesitan estar correctamente acondicionadas, como se especifica en los requerimientos del trabajo. Con los servomotores, la comunicación es a través de un bus CAN, como se explicó en capítulos previos. La interacción con una PC se consigue meidante el protocolo USB, junto con una adaptación de señal de un conversor.

Las interacciones internas del sistema incluyen:

- Periférico controlador y *transceiver* CAN.
- Señales discretas eléctricamente aisladas de la alimentación del sistema.
- Display LCD.
- Teclado matricial.
- Puerto USB.

El display y el teclado son las interfaces directas que permiten la interacción de un usuario con el sistema. Como se explicó en el capítulo 2, utilizan tanto I2C como entradas y salidas discretas del controlador, respectivamente, para ser manipulados por el microcontrolador. También, en la sección 2.4.3, se detalla el funcionamiento de una interfaz UART-USB, utilizada para generar la interacción entre el controlador con una PC externa. Este medio puede ser utilizado como interfaz de usuario con el uso de un software de monitoreo serial en la PC.

3.2. Selección de componentes

Previo al desarrollo del hardware y del software, se determinaron los componentes sobre los cuales se construye el trabajo. En su selección, se eligió como proveedor principal a la empresa Digikey[**web_digikey**] y, para los que se consiguieran

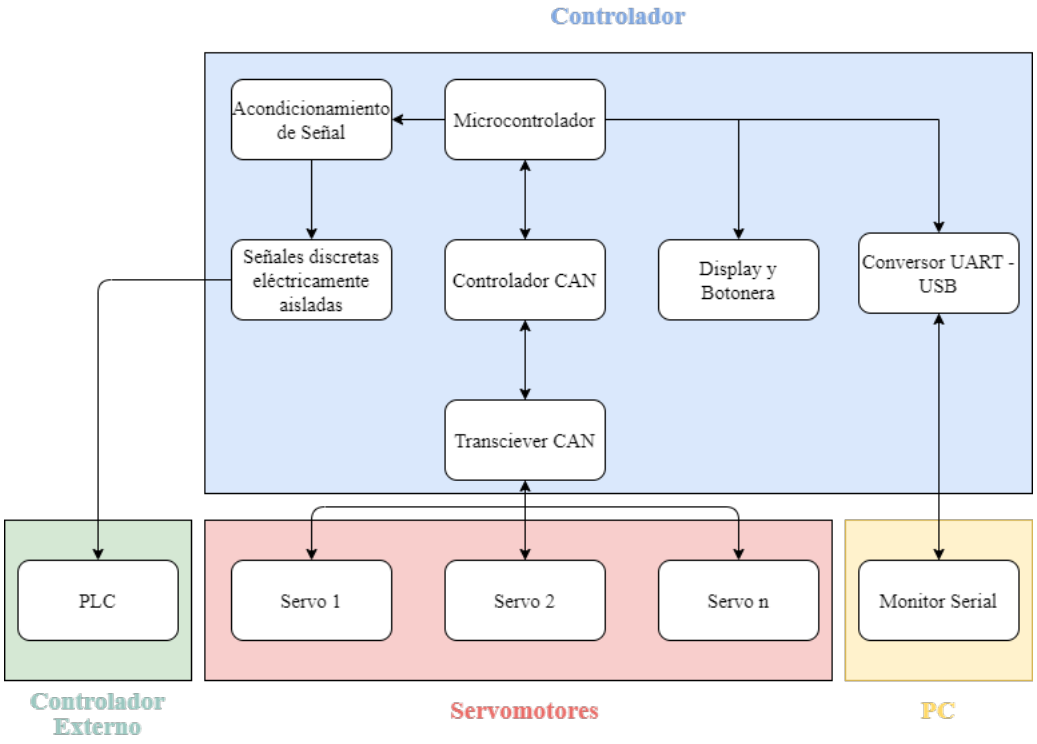

FIGURA 3.1. Arquitectura del sistema

dentro de la Argentina, a SyC electrónica[**web_syc**]. Ambas son empresas con las que Cambre ICyFSA trabaja regularmente.

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

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

La disponibilidad de los componentes terminó resultando uno de los problemas principales en el desarrollo del proyecto debido al gran desabastecimiento generado en el transcurso de la pandemia Covid-19. Esto fue una limitante a la hora de realizar la selección.

Algunos de los componentes se eligieron porque ya se había trabajado con ellos para el proyecto SN-17 y se contaba con stock de estos. Entre ellos se puede mencionar el microcontrolador ATSAMC21G18A[**web_ATSAMC21G18A**]. Este se destaca por ccontar con un periférico controlador CAN incorporado. También por razones de stock y de uso previo se seleccionó el *transceiver* CAN LTC2875IS8[**web_transciever_CAN**]

Para el regulador de tensión se seleccionó el LM2575[**web_LM2575**] con el empaquetado D2PAK. Este es un componente muy popular para este tipo de aplicaciones y presentaba disponibilidad de stock al momento de la elección. Se verificó que cumpliera los requerimientos de alimentación del sistema y se calculó que las temperaturas de operación fueran adecuadas.

Para la interfaz UART-USB, la falta de disponibilidad de opciones resultó especialmente limitante. Se buscó un modelo que el fabricante indicara que era reconocido por Windows y sus drivers instalados de forma automática, para facilitar el uso del dispositivo. Se seleccionó el integrado CY7C64225[**web_interfaz_USB_UART**].

Con respecto a los optoacopladores, se estudiaron aquellos que cumplieran con el requerimiento de aislación eléctrica de 1 kV, que fueran de 4 canales y que su *footprint* fuera el menor posible. Se eligió el LTV-846S[**web_optoacopladores_LTV**].

Una vez seleccionados todos los componentes principales del sistema, se estudiaron los datasheets y se seleccionaron los componentes periféricos indicados en estos. Con esta información se completó el listado de componentes del sistema y se prosiguió con su adquisición.

Para el display LCD se eligió uno similar al explicado en la sección 2.4.1 del cual se contaba con amplia disponibilidad en el mercado local. Este trae incorporado la interfaz I2C para su manipulación.

Para el teclado matricial se buscó un modelo que tuviera cierta robustez mecánica en su composición, y que tuviera un tamaño similar al display. Se eligió el que se muestra en la Figura 3.2, que es de 4 filas y 4 columnas, del fabricante Adafruit[**web_adafruit**].

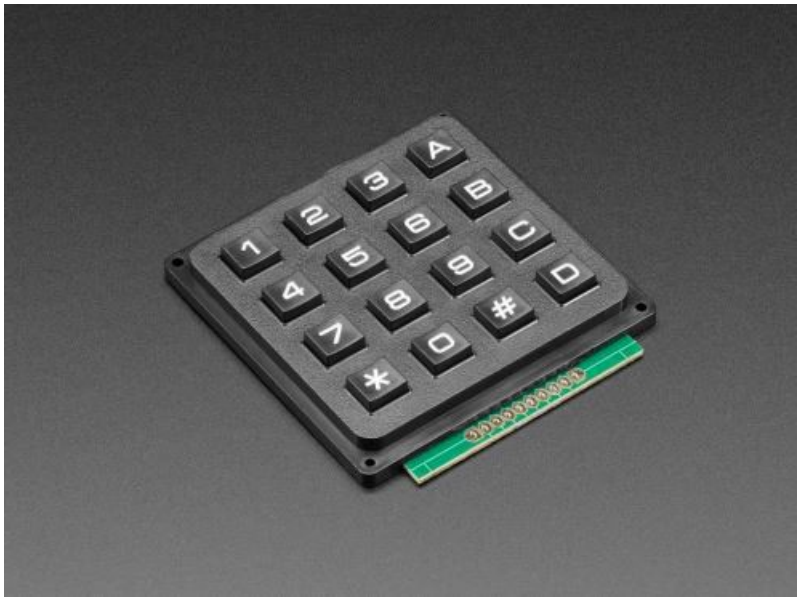

FIGURA 3.2. Teclado matricial Adafruit 4x4[**web_keypad**]

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 a la longitud del bus y al ruido eléctrico al que está expuesto. En la Figura 3.3 se presentan los tiempos de bit recomendados para implementaciones de CANopen[**Understanding_CAN**]. De este listado se seleccionó la fila para un bus de 250 m, que recomienda un *bitrate* de 250 kb/s y los tiempos de bit correspondientes. Esto se configura en el periférico de CAN del microcontrolador.

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

La estructura en la que se codificó 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.

En el campo de identificador se empleó el primer bit para indicar si el mensaje es un error. Dentro de la red, todos los mensajes que tengan este bit encendido no son filtrados y son los que tienen la mayor prioridad. Los siguientes 9 bits indican el identificador del dispositivo transmisor o receptor del mensaje, según corresponda. Cada dispositivo en la red tiene un número identificatorio único, el cual es fijo en el firmware. También, se especifican identificadores comunes para mensajes del tipo *broadcast*, dirigidos a todos los dispositivos de la red. El último bit se usa para indicar la dirección del mensaje, es decir, si es un mensaje envíado por el SCI-CAN, que actúa como dispositivo central de la red, o por un sistema SN-17.

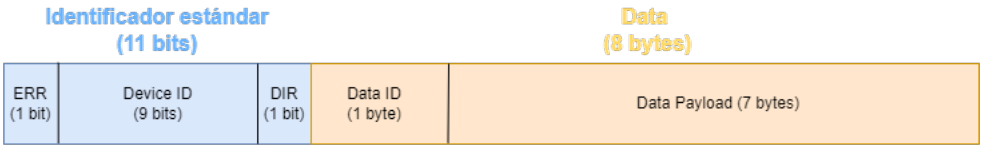

FIGURA 3.4. Estructura de mensajes CAN

Eel campo de data tiene una longitud total de 8 bytes. El primer byte se utiliza para identificar el tipo de información del mensaje, mediante un *data ID*. En la Tabla 3.1 se muestran todos los *data ID* implementados en el sistema y el carácter de esa información. Para cada uno de estos tipos de mensaje la información se codifica de una forma diferente. Distintas funciones debieron ser desarrolladas en el software para su implementación, como se explica en las secciones siguientes.

Los mensajes del tipo instrucción se refieren a las instrucciones de los programas que ejecutan los sistemas SN-17 en operación. Codifican un número según el tipo de dato que se utilice y el número de programa e instrucción al que corresponden. Para los mensajes de configuración, se agrega información complementaria en cada uno para indicar los parámetros configurables. Por ejemplo, en el caso de una constante PID, se indica en forma codificada cual es la constante en cuestión y el valor de esta. Tanto las señales de instrucción como de configuración pueden tener ambas direcciones de transmisión según si es una programación o una consulta.

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

Los mensajes de comando se caracterizan en que son envíados exclusivamente por el SCI-CAN para exigir la ejecución de alguna operación de los motores. Como en el caso de las configuraciones, algunos comandos requieren de información complementaria que se codifica en los bytes de *payload*, según sea necesario.

Los mensajes de supervisión son envíados exclusivamente por los motores cuando se encuentran en modo de operación y se usan para indicar el estado actual de funcionamiento de estos.

Finalmente, los mensajes del tipo general son envíados por la central como *broadcast* para todos los dispositivos de la red, los cuales responden usando el mismo identificador. Estos mensajes se usan para controlar el funcionamiento general de la red.

VER SI EN ESTA SECCION CONVIENE AGREGAR MAS GRAFICOS O TABLAS PARA DESCRIBIR EN MAYOR DETALLE LA CONFORMACION DE CADA UNO DE LOS MENSAJES. QUIZAS UNA TABLA INDICANDO CADA UNO DE LOS 8 BYTES SEGUN ID

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[**web_dev_board**]. Esto permitió hacer pruebas del sistema previo al diseño del circuito, lo que facilitó el proceso de implementación.

El software del sistema se diagramó siguiendo una estructura de capas. En la Figura 3.5 se muestra un esquema de la arquitectura de software 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.

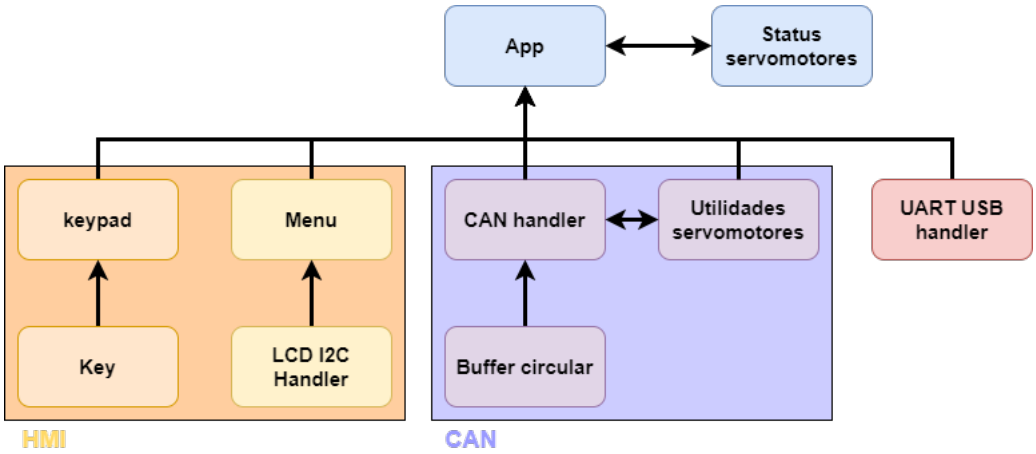

FIGURA 3.5. Arquitectura de software

En la capa superior se tiene a la aplicación, que interactúa, en un mismo nivel, con una sección del código que almacena el status 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 del software implementado.

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 que contiene el microcontrolador seleccionado. Se configuró mediante la herramienta de desarrollo provista por el fabricante[**web_microchip_studio**], a través de la cual se seleccionaron:

- Los tiempos de bit y la tasa de transmisión de 250 kb/s, según la Figura **??**.
- El uso del identificador estándar.
- El funcionamiento de los filtros.
- La operación en caso de colisiones.

El driver implementado opera por interrupción, con funciones de *callback* que son llamadas cuando se recibe o termina la transmisión de un mensaje. Para esto, se adaptó el código de ejemplo provisto por el fabricante.

Para la recepción de mensajes, se empleó un módulo de software de *buffer* circular, adaptado del trabajo final de especialización de Gabriel Gavinowich[**tpf_gabriel**]. Este permite el almacenaje de más de un mensaje y evita que esta información se pierda en caso de que el procesador se encuentre ocupado atendiendo otra tarea y llegue un nuevo mensaje en el bus. Los mensajes que se encuentran almacenados en el *buffer* son luego procesados fuera del estado de interrupción, dentro del ciclo del bucle principal.

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 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, esta se encuentra codificada en el primer byte de data del mensaje CAN. Esta es identificada y, utilizando una estructura *switch-case*, se selecciona la función decodificadora correspondiente.

VER SI SE PUEDE AGREGAR UNA IMAGEN PARA CLARIFICAR EL CONCEPTO DEL CODIGO

ESCRIBIR SERVICIOS A CAPAS SUPERIORES

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 comunicarse con la pantalla, y del teclado, un *middleware* que controle un sistema de menúes, la interacción de ambos drivers y que ofrezca funcionalidades a la capa superior de aplicación.

Con respecto al driver I2C para el display, para la configuración del protocolo se empleó la herramienta de desarrollo ofrecida por el fabricante del microcontrolador. Sobre eso, se construyeron una serie de servicios para mostrar información

en el display siguiendo ejemplos de códigos libres de implementaciones sobre Arduino[**web_arduino**] de este tipo de dispositivos[1].

Para el driver del teclado[2], se siguió una metodología similar, usando la herramienta de desarrollo para hacer la configuración de los pines de entradas y salidas del controlador y, sobre eso, se implementó una estructura de código adaptada de implementaciones libres de Arduino que ofrecen servicios para reconocer que teclas se oprimieron a capas superiores.

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 3.6 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.

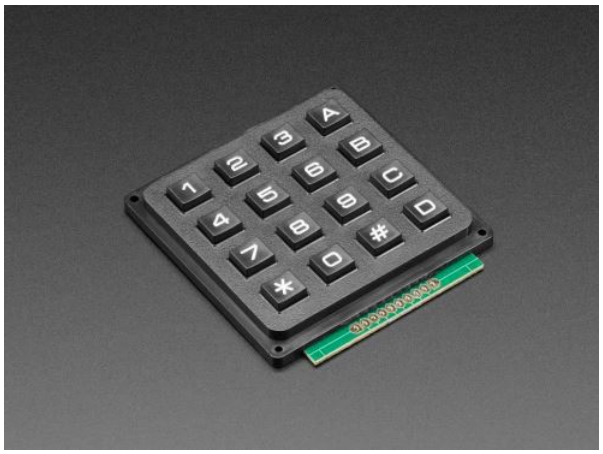

FIGURA 3.6. Ejemplo de opciones de menú - ARMAR IMAGEN

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 esta no puede ser modificada.

En el estado de programación, la configuración de los distintos motores es accesible para ser modificada y, además, 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.

[1] https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library

[2] https://github.com/Chris–A/Keypad

3.4.4. Interfaz UART-USB

La interfaz UART-USB involucró el desarrollo de un manejador de UART desde el microcontrolador. Para esto, se utilizó la herramienta de desarrollo provista por el fabricante, donde se realizó la configuración del periférico de comunicación. El software opera por interrupciones de hardware, que llaman a funciones de *callback* al completar una transmisión o recepción de mensaje.

Los mensajes UART son adaptados a señales USB mediante el circuito integrado CY7C64225[**web_interfaz_USB_UART**]. 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.

También, se puede utilizar una PC para enviar data al dispositivo, para que este lo transmita a un motor conectado. Esto se implementó utilizando la misma estructura de mensajes de CAN, los cuales se envían desde una conexión serial y el dispositivo los convierte a señales CAN directamente.

3.5. Modificaciones firmware SN-17

Para que el sistema SCI-CAN pueda comunicarse con las plaquetas SN-17, se tuvieron que realizar modificaciones al firmware de estas. 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 almacenado de las configuraciones en memoria no volátil.

Con respecto a la implementación de CAN, los módulos de software desarrollados para el SCI-CAN se utilizaron en los sistemas SN-17, 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.

Para la implementación del módulo de utilidades de servomotores se buscó también que ambos sistemas compartieran el mismo software. Para ello, se planteó el uso de sentencias condicionales que determinaran si un mensaje está dirigido a un sistema SCI-CAN o a un SN-17. A partir de esto, se determina la acción a realizar en la capa de aplicación según la información del mensaje. En el caso de un sistema SN-17, la capa de aplicación recibe de la capa CAN la solicitud de realizar determinada acción, la cual procesa según corresponda.

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 esta información. Esta estructura es accedida directamente por la capa de aplicación, quien se encarga de administrarla.

Esta misma estructura se aplicó en el dispositivo SCI-CAN, donde se utilizan varias instancias de esta para almacenar la información reportada por los sistemas

SN-17 conectados a la red. En el esquema de la Figura 3.5, el módulo Status servomotores es el que implementa la estructura descripta. La misma arquitectura es empleada en los sistemas SN-17, con la diferencia de que cuentan con una única instancia de la estructura de información.

VER DE AGREGAR ACA UNA IMAGEN EXPLICANDO LA INFORMACION DE LA ESTRUCTURA.

Para el almacenaje, se utilizó la misma estructura de información del sistema. El microcontrolador empleado permite la escritura de algunas secciones de memoria no volátil interna mientras el dispositivo se encuentra en operación. Sin embargo, estas secciones se comparten con el firmware del dispositivo, por lo que es necesario determinar correctamente las secciones disponibles. Para esto, se modificó el archivo *linker script*, donde se indica al compilador el tamaño se la memoria ROM. Esta modificación no permite que el firmware se almacene en cierta sección de la memoria. En esta sección es donde se almacena la información del sistema en tiempo de ejecución.

3.6. Desarrollo de Hardware

Para el desarrollo del hardware se utilizó el software de diseño Altium Designer[**web_altium**] y se eligió como fabricante a PCBWING[**web_pcbwing**] que es una empresa con la que regularmente trabaja Cambre ICyFSA. Se tomó, como punto de partida, las capacidades técnicas de este fabricante.

Se decidió hacer una placa de 4 capas, con dimensiones menores a 100 x 100 mm. Esto se debe a que el fabricante ofrece precios más económicos para estas especificaciones y se consideró que no imponen restricciones importantes el diseño requerido.

Previo al comienzo del desarrollo del circuito esquemático, se recopiló la información de todos los componentes seleccionados y se cargaron sus *pinouts, footprints* y modelos 3D.

Para los conexionados se seleccionaron los conectores tipo COMBICON con tornillo[**web_combicon**] para las conexiones externas, y los conectores tipo XH[**web_xh_connector**] para las conexiones internas del sistema.

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

Para cada uno de estos subcircuitos se armó un dibujo esquemático. En la Figura 3.7 puede verse 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 la hoja de datos correspondiente.

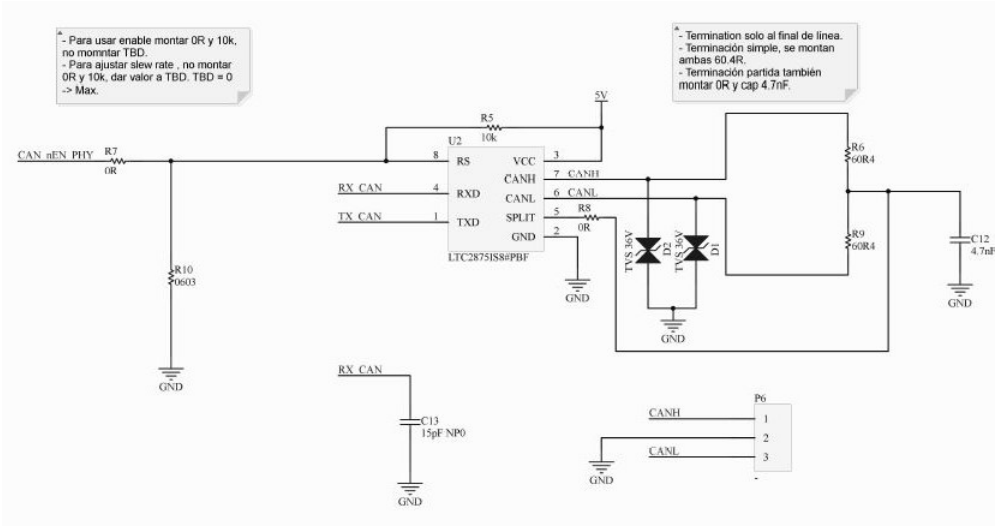

FIGURA 3.7. Esquemático de Interfaz CAN

Finalizados los subcircuitos esquemáticos, se continuó al 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 a una dimensión final de plaqueta de 100 x 80 mm.

Para el diseño del *PCB* se buscó que:

- Los subcircuitos quedaran separados.
- Los conectores quedaran todos 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 3.8 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

Para el desarrollo del gabinete se usó el software de diseño mecánico Autodesk Inventor[**web_inventor**]. Se determinó armar un ensamble que incluya todos los componentes y se lo pensó para poder luego ser impreso en 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[**web_grabcad**]. 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.

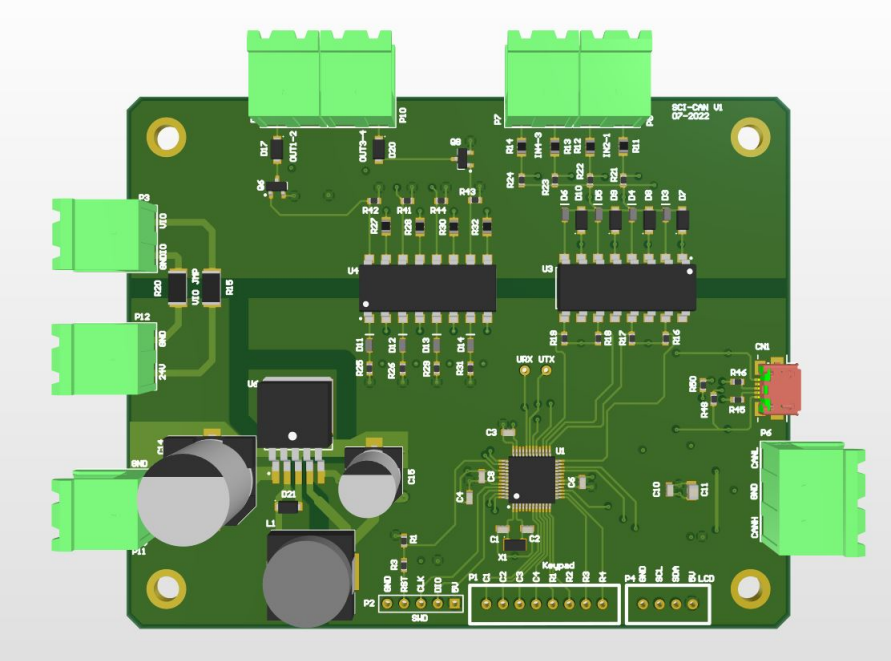

FIGURA 3.8. Render del *PCB*

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

- Facilitar el acceso a los conectores de la plaqueta SCI-CAN.
- Proteger los circuitos internos de la plaqueta SCI-CAN.
- 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 tuvieron que mantener las dimensiones máximas de las piezas a medidas menores que la capacidad de fabricación de la impresora 3D *Creality Ender-3*[**web_ender3**]. Esto corresponde a 220 x 220 x 250 mm.

En la Figura 3.9 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. Como se puede ver, 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.

Para la fabricación del conjunto, se utilizó el programa UltiMaker Cura[**web_cura3d**], 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.

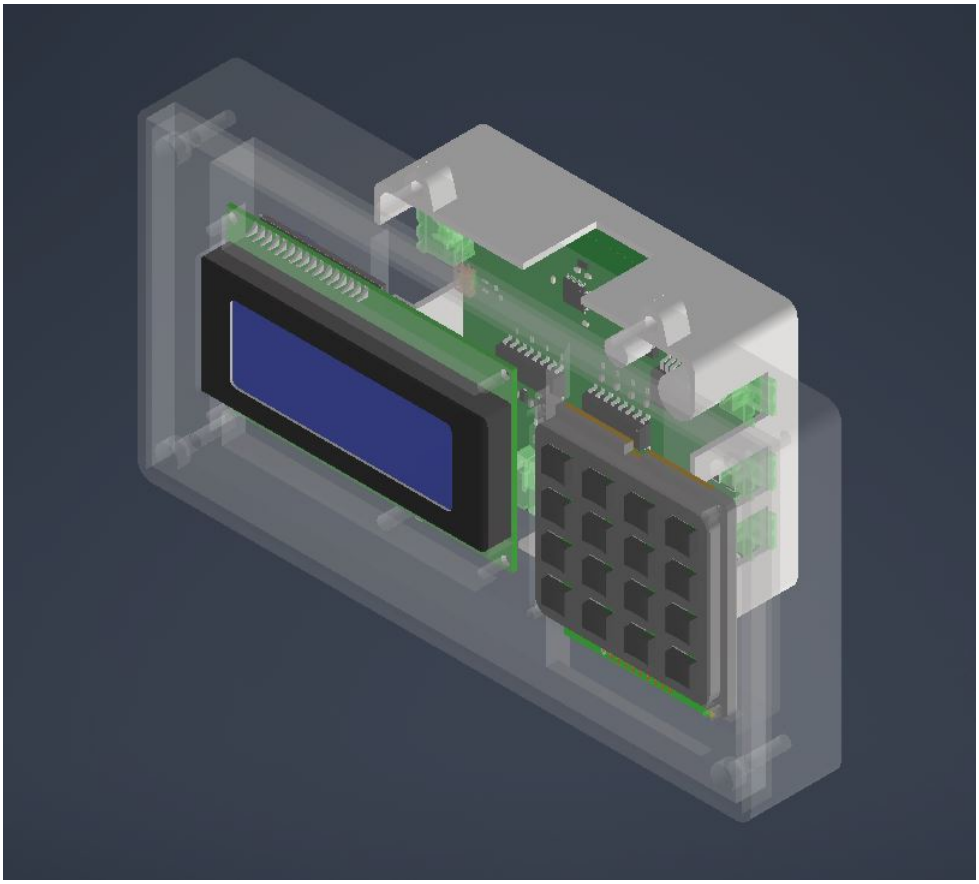

FIGURA 3.9. Ensamble 3D de gabinete

Capítulo 4

Ensayos y resultados

En este capítulo se detallan los ensayos y mediciones realizados sobre el sistema, para validar su funcionamiento y el cumplimiento de los requerimientos del trabajo. Se explicará cómo está compuesto el banco de pruebas utilizado y los instrumentos empleados, cómo se comprobó el envío de mensajes CAN a través de la red, qué ensayos eléctricos se realizaron sobre el sistema y cómo se verificó el envío de datos a través de UART y USB. Finalmente, se hablará de los ensayos que se realizaron en la planta de Cambre ICyFSA.

4.1. Banco de pruebas

Para validar el correcto funcionamiento del sistema, se ensambló un conjunto, se le cargó el firmware y se armó una red CAN con algunos motores paso a paso con plaquetas SN-17 conectadas. Dependiendo el ensayo a realizar, se conectaron 1 o más motores a la red, junto con un osciloscopio Hantek DSO2D10[1]. Los dispositivos se alimentan con una fuente DC regulable YIHUA 305D[2] trabajando a 24 V.

En la Figura 4.1 puede verse un esquema de la composición del banco de pruebas y los conexionados. En la PC se corre un programa de monitor serial llamado PuTTy[3] que se emplea para visualizar de forma simplificada el mensaje que el sistema intenta envíar a la red CAN.

4.2. Ensayo de mensajes CAN

En la Figura 4.2 se puede ver una de las tramas CAN tomadas desde el osciloscopio. En esta, se pueden ver las señales CAN-H y CAN-L en un estado recesivo a 2.5 V y separándose un poco más de 1 V, en ambos sentidos, en un estado dominante, este es el comportamiento esperado. Notar también la falta de ruido en la señal, lo que indica la correcta operación 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 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.

[1] http://hantek.com/products/detail/17182

[2] http://yihuasoldering.com/product-4-2-30v-dc-power-supply/160008/

[3] https://www.putty.org/

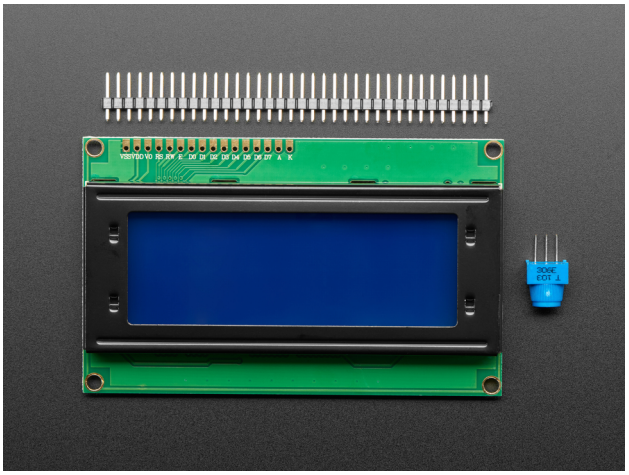

FIGURA 4.1. Banco de pruebas utilizado para verificaciones - ARAMR FIGURA

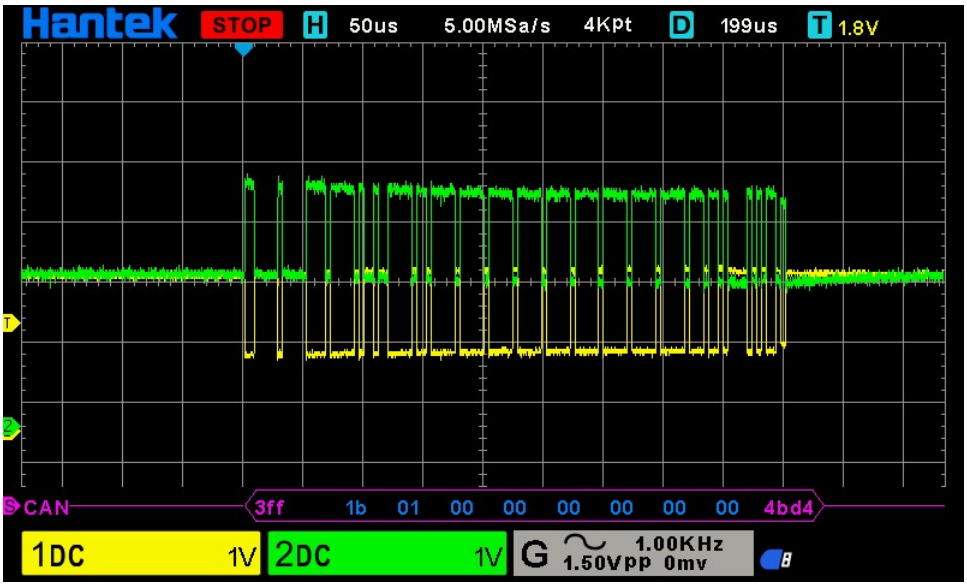

FIGURA 4.2. Niveles de señal CAN en osciloscopio

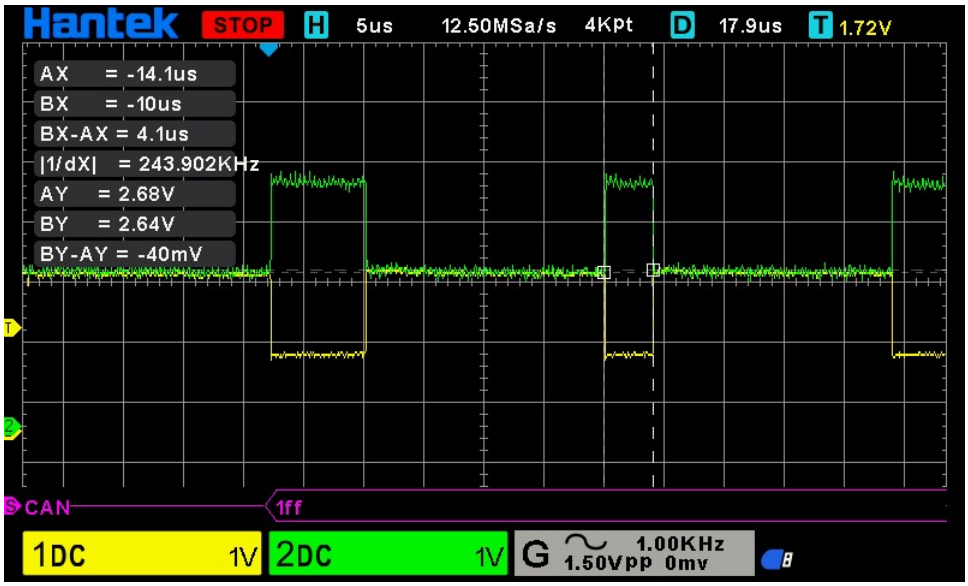

FIGURA 4.3. Medición de tiempo de bit

Para la verificación de los mensajes CAN se probaron cada uno de los mensajes posibles. Se revisó que el identificador y la data fueran correctas y que los sistemas conectados realizaran las acciones correspondientes. Siguiendo esta metodología, se ensayaron cada uno de los mensajes especificados, según la Tabla 3.1.

En la Figura 4.4 se puede ver una imagen tomada del osciloscopio para la instrucción de calibrar motor (código: 0x0c). En la parte iferior puede verse el decodificador CAN que el osciloscopio trae incorporado, marcando correctamente la instrucción. También, se puede ver el identificador del mensaje (código: 0x0b) compuesto por el identificador del motor (0x05) y el bit final indicando 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.

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 este ángulo es un número que requiere más de un byte para su representación, aparece descompuesto en el mensaje. Si se hace la conversión del número 168, de hexadecimal a decimal, se comprueba que es efectivamente 360. Al recibir el mensaje, el motor correctamente gira lo estipulado.

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.

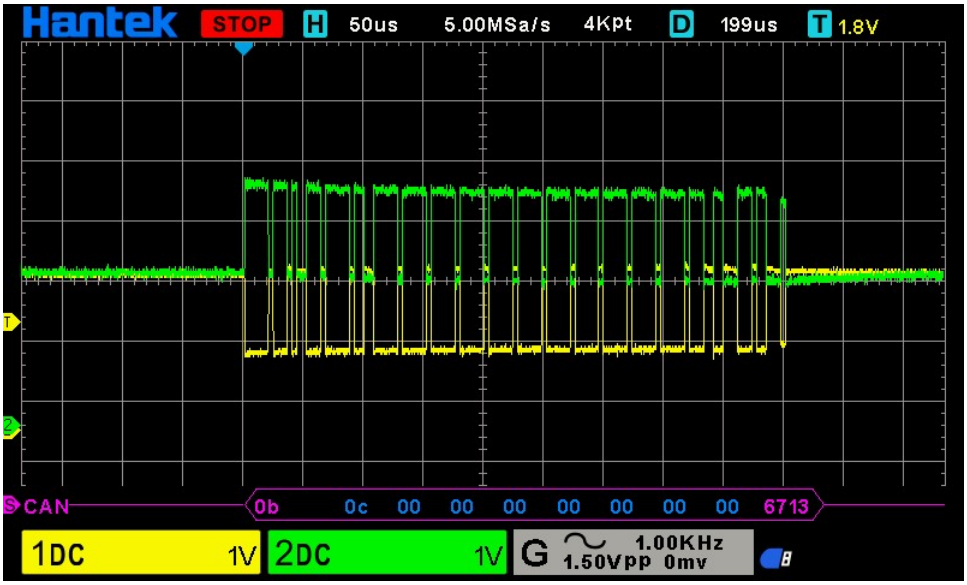

FIGURA 4.4. Instrucción calibrar motor

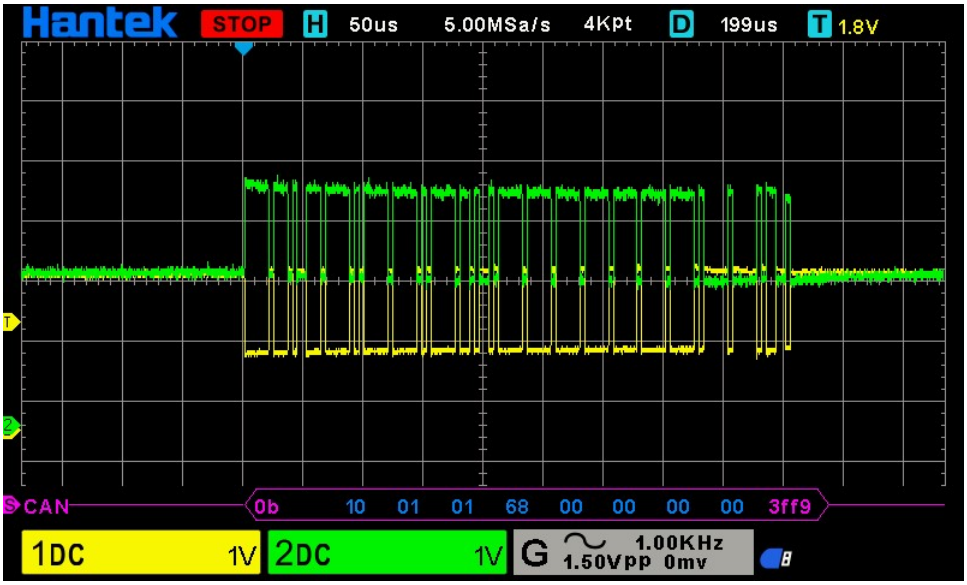

FIGURA 4.5. Instrucción manual mover motor

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

Con respecto a las salidas PNP del sistema, se ensayaron con una conexión a un PLC Omron NX102[4]. Las 4 salidas se conectaron a un módulo de entradas DC NX-ID5442 para este PLC que opera a 24 V. Se comprobó que, al activar las salidas, el PLC las recibiera de forma correcta.

Para 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 recibiera la señal desde el sistema.

VER DE AGREGAR UNA IMAGEN CON EL DISPOSITIVO CONECTADO AL PLC

4.4. Ensayos de mensajes UART-USB

Para los mensajes de UART y USB se conectó el osciloscopio a la línea UART del sistema y a la conexión USB desde el sistema a una PC y se realizaron las mediciones.

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.

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 un estado de desarrollo, la cual cuenta con varios sistemas SN-17 implementados 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.

[4] https://www.ia.omron.com/data_pdf/cat/nx1_p130-e1_dita_4_4_csm1063211.pdf?id=3705

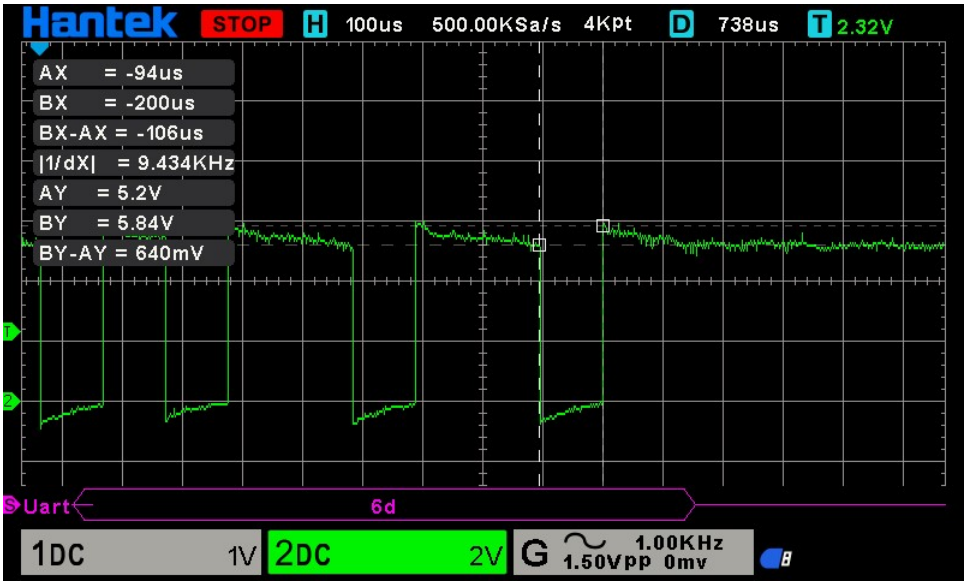

FIGURA 4.6. Medición de señal UART

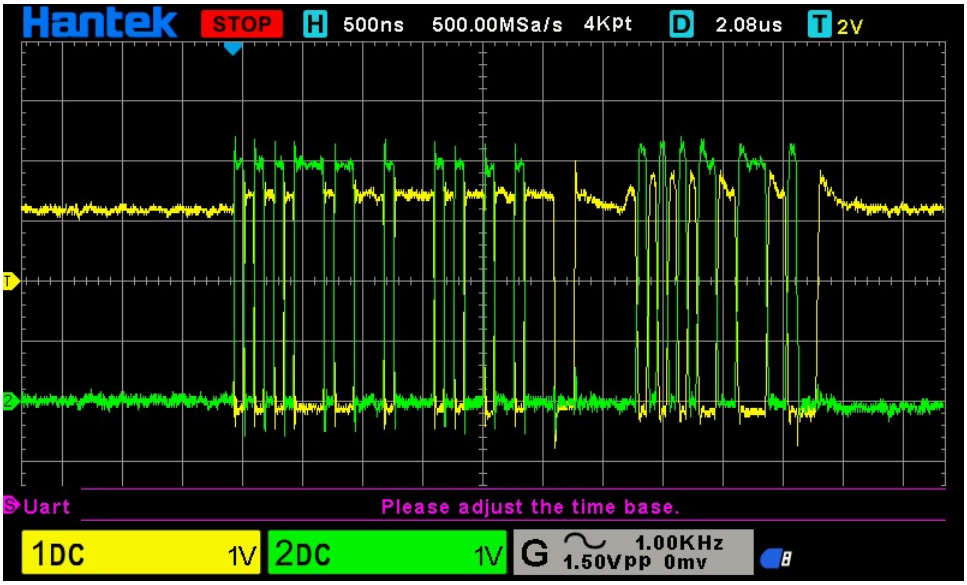

FIGURA 4.7. Medición de señal USB

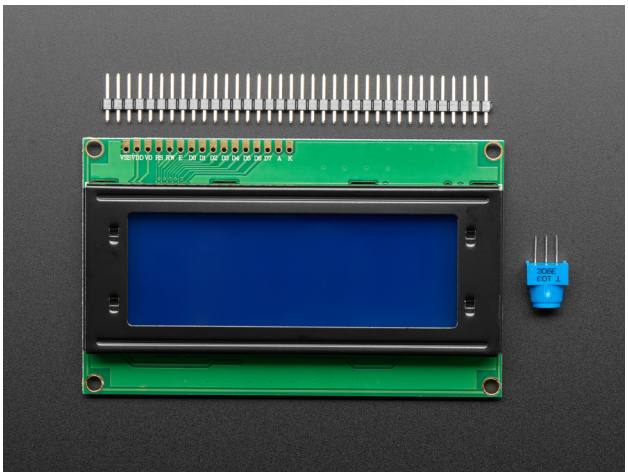

FIGURA 4.8. Esquema de conexionado en planta - ARAMR FIGURA

Con la disposición explicada, se realizó la programación de los sistemas SN-17 de la línea y se hicieron pruebas de monitoreo del sistema junto con los ensayos realizados sobre la línea. Se comprobó que el sistema mostrara 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 sistema 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.

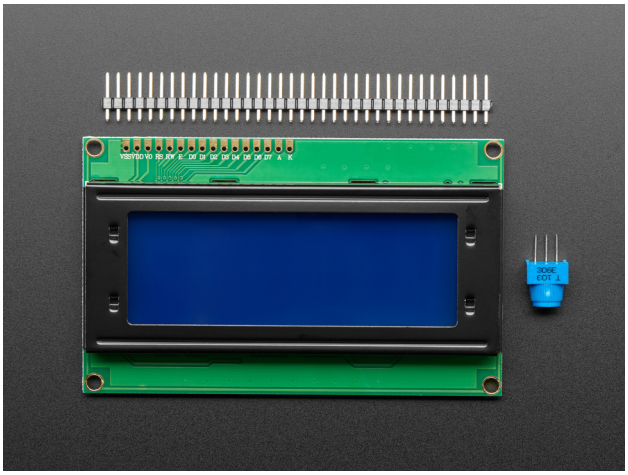

FIGURA 4.9. Monitoreo de SN-17 - ARAMR FIGURA INCLUIR ERROR

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 este 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 sistema montado en la línea de ensamblaje automático en Cambre ICyFSA. Como se mencionó, la máquina se encuentra en desarrollo al momento de la escritura de este informe, por lo que la ubicación y

montaje del sistema aún no están determinados y las conexiones con los diferentes dispositivos no es la definitiva.

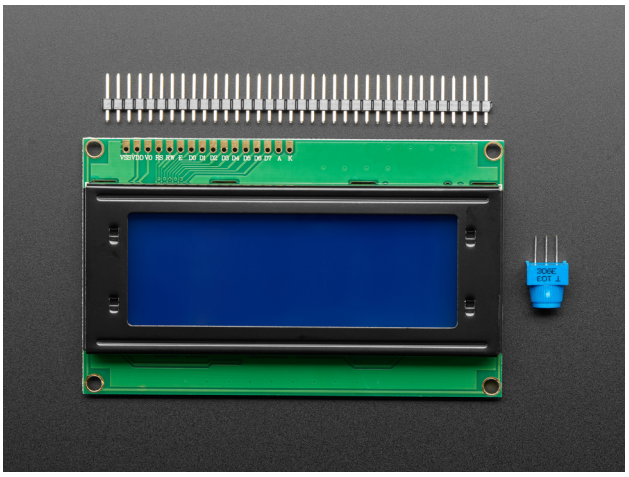

FIGURA 4.10. Sistema montado en línea de ensamble automático - ARAMR FIGURA

4.6. Comparación con estado del arte

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. Conclusiones generales

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 buscar realizar las compras de forma prioritaria fue efectivo, permitiendo 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.

El sistema obtenido pudo implementarse en una línea de ensamble automático en la planta industrial de Cambre ICyFSA. Si bien esta máquina, al momento de la escritura de este informe, está en desarrollo, se pudo probar el sistema en un ámbito industrial y su correcto funcionamiento. En la Figura 5.1 se muestra una imagen de la línea de ensamblaje en cuestión y se puede visualizar el sistema desarrollado junto con los actuadores que cuentan con los sistemas SN-17.

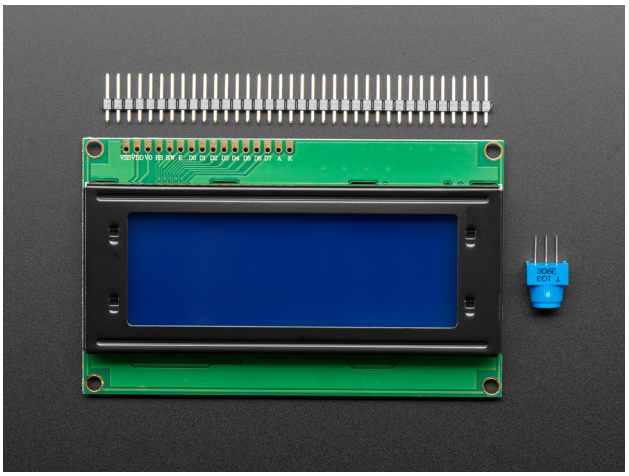

FIGURA 5.1. Línea de ensamble automática con SCI-CAN - ARAMR FIGURA

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.
- Implementar otros protocolos de comunicación industriales, permitiendo más diversidad de interacciones.