

Grado Universitario en Ingeniería en Electrónica Industrial
y Automática
2022-2023

Trabajo Fin de Grado

Librería de comunicación CANopen en
MATLAB para un cuello robótico blando

Cristian Jara Corros

Tutor

Luis Fernando Nagua Cuenca

Leganés, Madrid

Julio · 2023



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**

RESUMEN

Este Trabajo Fin de Grado se centra en el desarrollo e implementación de una librería de comunicaciones en el entorno software de Matlab, específicamente diseñada para el proyecto SOFIA de la Universidad Carlos III de Madrid. Basándose en el protocolo de comunicación CANopen, utilizado por el robot humanoide TEO, esta librería permite una comunicación eficaz entre los drivers de control del cuello robótico blando y el ordenador.

A lo largo de los meses de desarrollo del trabajo se propusieron diferentes soluciones para abordar el diseño de la librería hasta dar con el más adecuado para este proyecto. Como resultado, se diseñó un software basado en la programación orientada a objetos a fin de dar una mayor versatilidad, solidez y escalabilidad a los programas que se crean a partir de esta. Además, una aplicación que se ha implementado en la librería creada es la cinemática inversa de un cuello robótico blando, este aspecto es clave para poder comandar a una posición correcta a cada uno de los motores de la articulación.

Las pruebas realizadas sobre el cuello y sobre el brazo robótico blando han validado la librería de comunicaciones, demostrando que puede ser utilizada en distintas articulaciones que serán integradas en el robot TEO.

Palabras clave: SOFIA, HUMASOFT, TEO, OSI, CAN, CANopen, MATLAB, POO.

ABSTRACT

This Final Bachelor Thesis focuses on the development and implementation of a communications library in the MATLAB software environment, specifically designed for the SOFIA project of the Carlos III University of Madrid. Based on the CANopen communication protocol, used by the TEO humanoid robot, this library allows efficient communication between the control drivers of the soft robotic neck and the computer.

Throughout the months of development of the work, different solutions were proposed to approach the design of the library until the most suitable one was found for this project. As a result, a software based on object-oriented programming was designed in order to give greater versatility, solidity and scalability to the programs that are created from it. In addition, an application that has been implemented in the created library is the inverse kinematics of a soft robotic neck, this aspect is key to be able to command each of the articulation motors to a correct position.

The tests carried out on the neck and on the soft robotic arm have validated the communications library, demonstrating that it can be used in different joints that will be integrated into the TEO robot.

Keywords: SOFIA, HUMASOFT, TEO, OSI, CAN, CANopen, MATLAB, OOP.

DEDICATORIA

A mi madre, por su apoyo y consejo, los cuales han sido clave para lograr las metas que me he propuesto. Su ejemplo de esfuerzo, perseverancia y trabajo duro me ha mostrado el camino a seguir y me ha empujado día a día. Gracias mamá, por todo lo que me has enseñado todos estos años, por el amor que me has dado, que me han convertido en la persona que soy hoy.

Agradezco a mi tutor, Luis, su tiempo, dedicación, enseñanzas y orientación en la elaboración de este trabajo. Gracias por su paciencia, por su capacidad para escuchar mis dudas y por su habilidad para guiarme hacia el resultado correcto.

Y finalmente, a todas aquellas personas que me han apoyado y caminan junto a mí. A todos, gracias.

"Cualquier tecnología lo suficientemente avanzada es indistinguible de la magia."

Arthur C. Clarke

ÍNDICE GENERAL

1. INTRODUCCIÓN	1
1.1. Cuello robótico blando.	2
1.2. Objetivos del trabajo	3
1.3. Marco regulador	4
1.4. Introducción al desarrollo de la librería de comunicación	5
1.5. Recursos utilizados.	6
1.5.1. Hardware	6
1.5.2. Software.	8
1.6. Estructura de la memoria	9
2. ESTADO DEL ARTE	10
2.1. Sistemas de comunicación empleados en la robótica	10
2.1.1. Protocolo EtherCAT	12
2.1.2. Protocolo RS-232	13
2.1.3. Protocolo RS-485	15
2.2. Librería desarrollada en el proyecto HUMASoft para la comunicación de los drivers.	17
2.3. Elección de MATLAB para diseñar una nueva librería de comunicaciones. . .	19
3. COMUNICACIÓN EN EL PROTOCOLO CANOPEN	21
3.1. El protocolo CAN	21
3.1.1. Origen del protocolo CAN	21
3.1.2. Capas del modelo OSI implementadas en el protocolo CAN	22
3.2. Protocolo CANopen	28
3.2.1. Estructura de los mensajes	30
3.2.2. Modelos de comunicación.	31
3.2.3. Tipos de mensajes u objetos de comunicación.	32
4. DESARROLLO DE LA LIBRERÍA.	35
4.1. Plataforma de MATLAB y su relación con el protocolo CANopen	36
4.2. Primeros desarrollos de la librería	39

ÍNDICE GENERAL

4.3. Plataforma de MATLAB y la programación orientada a objetos.	44
4.4. Librería de comunicación CANopen en MATLAB	45
4.4.1. Funciones de configuración del canal de comunicación	46
4.4.2. Clases desarrolladas para la comunicación con el cuello robótico.	47
4.4.3. Cinemática inversa	58
4.4.4. Sensor inercial	59
5. PRUEBAS Y RESULTADOS EXPERIMENTALES	62
6. PRESUPUESTO DEL PROYECTO	66
6.1. Impacto socio-económico	66
6.2. Presupuesto	66
7. CONCLUSIONES Y TRABAJOS FUTUROS	70
7.1. Conclusiones	70
7.2. Trabajos futuros	71
BIBLIOGRAFÍA	72

ÍNDICE DE FIGURAS

1.1	Imagen del robot TEO.	1
1.2	Cuello robótico blando incorporado al robot TEO [7].	2
1.3	Detalle de la inclinación y orientación de la articulación.	3
1.4	Adaptador PCAN-USB de PEAKSystem [11].	6
1.5	Etiquetas identificadoras iPOS4808 MX-CAN [13].	7
1.6	Sensor 3DM-GX5-10 [14].	7
1.7	Interfaz gráfica del software PCAN-View [11].	8
1.8	Datos recibidos por el sensor inercial.	9
2.1	Robot humanoide iCub.	10
2.2	Robot humanoide HRP-2.	11
2.3	Robot humanoide TEO.	12
2.4	Ejemplo de topología de conexión en EtherCAT [22].	13
2.5	Señales de la norma V.24 y tipos de conectores [26].	14
2.6	Conexión serial DB9 para comunicaciones RS-232 [27].	14
2.7	Interfaz de control de posición.	19
3.1	Transmisión por diferencia de tensión.	23
3.2	Sucesión de bits que representan niveles de tensión.	24
3.3	Distribución del bus CAN de alta velocidad. ISO 11898-2 [36].	25
3.4	Distribución del bus CAN de baja velocidad. ISO 11898-3 [36].	25
3.5	Método de arbitraje [39].	26
3.6	Estructura de trama de datos.	27
3.7	Ejemplo de representación de parámetros del diccionario de objetos [44].	29
3.8	Formato de trama CANopen.	30
3.9	Máquina de estados de un nodo [45].	32
3.10	Estructura para el protocolo SDO [46].	34
4.1	Esquema de conexión de los elementos [9].	35

ÍNDICE DE FIGURAS

4.2	Compatibilidad de la herramienta de PEAK-System en MATLAB.	36
4.3	Información facilitada tras realizar la llamada a la función 'canChannelList' [48].	37
4.4	Máquina de estados del driver dentro de CiA402 [54].	39
4.5	Tabla en la que se describe el significado de cada uno de los bits de la word de control.	41
4.6	Diagrama de clases.	45
4.7	Relación entre los métodos de la clase CiA402Device.	49
4.8	Orden de ejecución de los métodos tras llamada al constructor de la clase.	50
4.9	Métodos empleados tras la ejecución del método resetSetPoint	52
4.10	Métodos empleados tras la ejecución del método setPosition	52
4.11	Métodos empleados tras la ejecución del método setupPositionMode . . .	52
4.12	Métodos empleados tras la ejecución del método getPosition	53
4.13	Métodos empleados tras la ejecución del método printStatusWord	53
4.14	Diagrama de la clase Elemento.	54
4.15	Orden de ejecución tras uso del método ConfiguracionDriver	55
4.16	Orden de ejecución tras uso del método setupPositionMode	55
4.17	Orden de ejecución tras uso del método busSetPosition	55
4.18	Orden de ejecución tras uso del método cambioSentidoGiro	56
4.19	Orden de ejecución tras uso del método busGetPosition	56
4.20	Orden de ejecución tras uso del método busChangeVelocity	56
4.21	Orden de ejecución tras uso del método busGetStatusWord	57
4.22	Orden de ejecución tras uso del método busGetTargetPosition	57
4.23	Diagrama del cuello robótico blando [63].	58
4.24	Diagrama de clases relacionadas con el sensor inercial.	60
5.1	Imagen del cuello robótico durante las pruebas.	63
5.2	Gráficas de los datos obtenidos por el sensor inercial en el cuello.	64
5.3	Brazo robótico blando del proyecto SOFIA.	64

ÍNDICE DE TABLAS

2.1	Señal de transmisión RS-232 desglosada por bits.	15
2.2	Diferencias entre RS-232 y RS-485.	16
3.1	Relación de longitud y velocidad de transmisión.	23
3.2	Asignación predefinida de identificadores de mensaje.	31
4.1	Tamaño del mensaje.	37
4.2	Estructura de los datos dentro de los mensajes CANopen.	38
4.3	Atributos utilizados y sus valores de la clase CiA402DeviceDictionary. . .	48
6.1	Costes totales de la mano de obra.	67
6.2	Costes totales del software.	68
6.3	Costes totales del hardware.	68
6.4	Resumen de costes por capítulo.	69

1. INTRODUCCIÓN

La presente memoria forma parte del proyecto SOFIA desarrollado por la Universidad Carlos III de Madrid. El proyecto mencionado tiene como objetivo el desarrollo de eslabones y articulaciones blandas que permitan movimientos suaves y precisos, y en la integración de sensores inteligentes para mejorar la percepción y el control de robots humanoides [1]. Estos nuevos elementos podrán ser implementados en varias extremidades tales como brazos, cuello y espina dorsal.

La investigación en articulaciones blandas y flexibles es un área emergente en la robótica blanda que está tomando gran importancia, captando la atención en muchas aplicaciones, como en la manipulación, locomoción y rehabilitación [2]. Tiene el potencial de revolucionar la manera en que los robots interactúan con su entorno y con las personas.

El desarrollo de eslabones blandos se contrapone con los eslabones rígidos. Estos últimos están diseñados para soportar cargas y transmitir movimientos y fuerzas entre las diferentes partes del robot. Los eslabones blandos permiten más grados de libertad en los movimientos y permiten la reconfiguración de las articulaciones.

Se busca la inspiración en organismos biológicos para su diseño, que ha permitido dotar a los robots de una mayor flexibilidad y adaptabilidad. Estos aportes son características esenciales para operar en entornos o situaciones que suponen un reto para robots convencionales. Entre los escenarios donde estos equipos podrían actuar se encontraría: cirugías mínimamente invasivas, inspección de tuberías, entre otros [3].

La integración de materiales blandos ocasiona que la interacción entre robots y humanos sea más segura. Investigadores en un estudio [4] han evaluado una plataforma robótica que funciona mediante un sistema inflado y que incorpora sensores de presión para detectar colisiones.

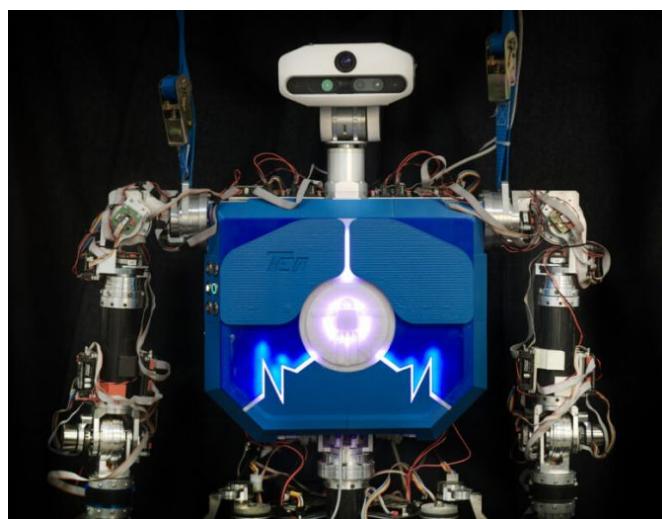


Fig. 1.1. Imagen del robot TEO.

El robot-humanoide TEO (Figura 1.1) del Robotics Lab de la Universidad Carlos III, que se integra dentro del proyecto HUMASoft [5], cuenta con articulaciones de este tipo, como es el caso del cuello. Este elemento, en particular, se encuentra en un alto grado de desarrollo, como se observa en los artículos donde se valida su funcionamiento [6].

1.1. Cuello robótico blando

En esta sección se presenta una descripción general de un prototipo de cuello robótico blando, que se basa en la configuración de robots con mecanismos paralelos conducidos por cable. La estructura principal se compone de una base fija, un eslabón blando y una plataforma móvil. En la figura 1.2 se observa este elemento incorporado al robot TEO.

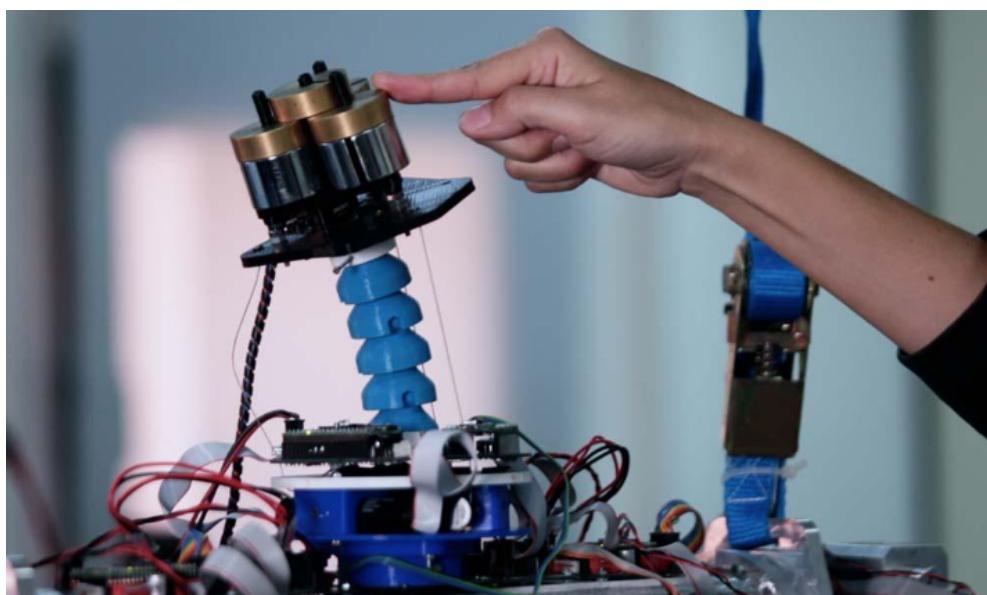


Fig. 1.2. Cuello robótico blando incorporado al robot TEO [7].

El eslabón blando realiza la función de una columna central que es capaz de flectar en múltiples direcciones. La dirección y fuerza del movimiento se ve marcada por la combinación de tensiones de los cables con los que cuenta la articulación. Este modelo cuenta con un diseño ligero y compacto que le permite moverse 360° alrededor del eje del cuello.

La acción de liberar y recoger cada cable viene dada por un actuador electrónico (motor) incorporado a la base del cuello. La articulación diseñada en el Trabajo Fin de Máster [8], cuenta con tres motores conectados a tres cables que ayudarán a comandar el movimiento del cuello.

Para realizar el control sobre este elemento se utilizan los parámetros de inclinación y orientación. El factor de inclinación corresponde con el movimiento de la parte superior del cuello. Este podrá variar hasta los 40° del ángulo con respecto a la posición de reposo.

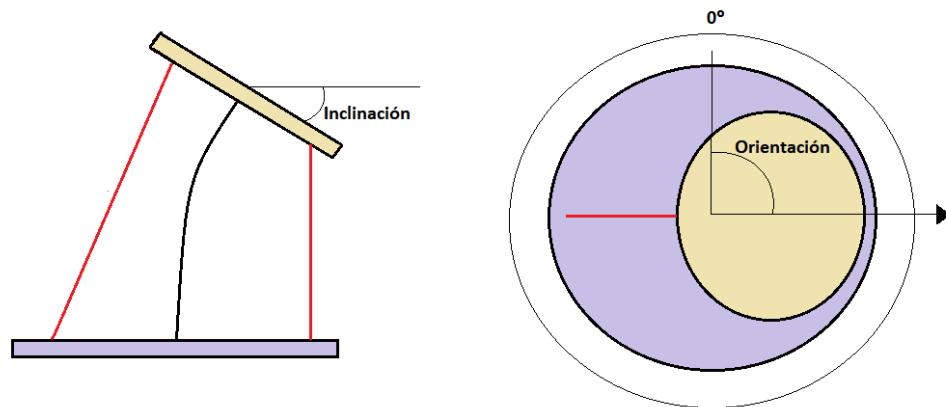


Fig. 1.3. Detalle de la inclinación y orientación de la articulación.

Por otro lado, la orientación hace referencia a la dirección angular en la que se realiza la inclinación y cuenta con un rango de 360° [9]. En la figura 1.3 se muestra una imagen descriptiva de una posición concreta del cuello, señalando la zona de actuación de cada parámetro.

1.2. Objetivos del trabajo

El propósito de este Trabajo de Fin de Grado consiste en crear una biblioteca de comunicación para la plataforma de software de MATLAB, que permitirá la interacción con los controladores iPOS4808 MX-CAN que componen el cuello robótico blando.

Como primer paso, se investigarán los estándares de comunicación utilizados en la robótica humanoide, prestando especial atención al protocolo CAN y CANopen. Aunque este no es el objetivo principal de esta memoria, es necesario conocer las especificaciones técnicas del protocolo para entender cómo se realiza la transferencia de datos entre dispositivos en la articulación del cuello robótico blando. Una vez que se tenga un conocimiento sólido del protocolo CAN y CANopen, se procederá al desarrollo de la propuesta de este documento.

Esta librería será compatible con el protocolo de comunicación CANopen, que se utiliza en el proyecto de HUMASoft [1] para la transferencia de datos entre los dispositivos que forman la articulación. La programación se adaptará a las necesidades actuales del proyecto, como el control y la obtención de datos de posición de los distintos servomotores que conforman la parte de los actuadores del sistema.

Se desarrollarán funciones para efectuar el enlace de comunicación con los drivers y obtener información de los encoders. Además, en una capa superior, se crearán funciones específicas para poder controlar simultáneamente los tres motores y lograr alcanzar la posición indicada por el usuario.

El usuario ingresará por pantalla las consignas de entrada de posición (tanto de incli-

nación como de orientación) y recibirá los datos de los encoders con respecto a la posición angular de cada uno de los servomotores para su posterior procesamiento. Se mostrará una aplicación de esta librería de comunicación, el cual consiste en comandar la articulación mediante el uso de la cinemática inversa.

Para obtener la medición real de posición del cuello robótico se incorporará a la programación un sensor inercial 3DM-GX5-10 que tiene una comunicación serial RS232, el cual proporcionará las mediciones de inclinación y orientación necesarias para el análisis de los datos. La programación para trabajar con el sensor ya existe en versión C++, pero es necesario adaptarla también a ©MATLAB.

Finalmente, todas las funciones se someterán a pruebas experimentales para evaluar su correcto funcionamiento.

El listado que se observa a continuación sirve para resumir los hitos propuestos para el desarrollo de este proyecto:

- Investigación y comprensión sobre los estándares de comunicación. En especial CAN y CANopen.
- Diseño y desarrollo de la librería de comunicaciones de CANopen:
 - Funciones de comunicación usando el bus CAN para interactuar con los drivers.
 - Funciones para el control de 3 motores de forma simultánea utilizando la cinemática inversa. Se utilizará el modo de funcionamiento por posición con el que cuentan los drivers.
- Adaptación de la librería del sensor inercial de su versión C++ a la plataforma de MATLAB.
- Ensayo y validación del correcto funcionamiento de las funciones desarrolladas.

1.3. Marco regulador

Dentro de esta sección se van a analizar las distintas cuestiones que se deben considerar a la hora de realizar este trabajo a nivel legislativo y técnico.

El desarrollo de esta librería de comunicaciones se encuentra del proyecto SOFIA de la UC3M, como se ha mencionado con anterioridad. Al tratarse de un proyecto que principalmente desarrolla prototipos con fines académicos no se ve afectado por la aplicación de ninguna legislación específica para este tipo de dispositivos ni por cuestiones de propiedad intelectual.

El cuello robótico no presenta riesgos de gran consideración. Al trabajar con la articulación de forma aislada del robot en un entorno acotado y seguro, el rango de movimientos

no es lo bastante amplio como para ocasionar situaciones peligrosas. La robótica blanda está pensada para que la interacción con personas se realice de forma segura sin causar daño alguno durante la interacción. Sí que se debe tener en cuenta que las articulaciones de este tipo, en caso de un mal cálculo en la trayectoria, pueden llegar a deformarse.

Por el contrario, al hacer uso de los estándares CiA402 y CiA301 que son los considerados por CANopen, si se ve afectado por un marco regulador. En este caso, están englobados en el modelo OSI (Open Systems Interconnection) que es un marco de referencia teórico para describir la arquitectura de las redes de comunicación, recogido en la ISO/IEC 7489. Este modelo OSI se verá con más detalle en el capítulo 3.

1.4. Introducción al desarrollo de la librería de comunicación

En primera instancia, y como se ha comentado en la sección 1.2, es necesario una investigación acerca de cómo funciona el protocolo de comunicaciones CANopen esto se realizó buscando información vía Internet y con ayuda de los manuales de funcionamiento de los dispositivos que el fabricante dispone.

Los drivers de ©Technosoft que son utilizados para comandar a los servomotores del cuello blando, utilizan el protocolo CANopen para realizar los enlaces de comunicación con el maestro de la red. Según la organización CAN in Automation (CiA), CANopen es una especificación de comunicación, derivado del protocolo CAN (Controller Area Network) [10], que define un conjunto de objetos y procesos para implementar sistemas de control distribuidos y en red. Está diseñado para ser altamente flexible y escalable, y se puede utilizar en una amplia gama de aplicaciones, desde sistemas de automatización industrial hasta vehículos móviles y sistemas de transporte público.

A la hora de realizar la programación se optó, en primer lugar, por un sistema en el cual se realizarían distintas funciones, independientes entre sí, en las que se debería indicar el dispositivo al cual se querría comunicar y el mensaje que se le debería enviar para obtener la respuesta de alguno de los registros.

Este planteamiento presenta un problema fundamental, y es lo complejo de su uso, ya que para cada petición era necesario definir mucha información que el usuario debería recordar e indicar. Algunos de los parámetros que debería ser escritos en cada mensaje como, por ejemplo: la dirección del dispositivo, el canal utilizado, la dirección del objeto del protocolo CAN al que se desea acceder o el mensaje que se deseaba enviar, entre otros.

Para facilitar la interacción con el dispositivo se encontró un planteamiento alternativo que resolvería las dificultades de uso por el paradigma anterior, este era la programación orientada a objetos. Al utilizar este tipo de programación, se podía definir como atributos de clase la dirección del dispositivo y el canal de comunicaciones utilizado. Al trabajar de esta forma se reduce los datos que el usuario deberá indicar en cada interacción con el programa.

Este no era el único beneficio que traía consigo esta forma de diseñar el programa. Utilizando la potencia que ofrece el polimorfismo, se podrían crear elementos más complejos compuestos por los objetos creados para cada uno de los drivers, con lo que se permitiría una escalabilidad a la hora de realizar la programación de elementos completos, como es el caso del cuello.

La utilización de la aplicación por parte del usuario será mucho más sencilla y cómoda, permitiéndole un control completo de la articulación utilizando menos comandos.

1.5. Recursos utilizados

En esta sección se dará a conocer los medios más relevantes utilizados, tanto hardware como software, para el desarrollo de la librería . En el apartado del hardware, aparte de los equipos empleados en el diseño de la aplicación, se describirán los equipos esenciales del cuello robótico que intervienen en la red de comunicaciones.

1.5.1. Hardware

PCAN-USB: el adaptador de la figura 1.4 es un dispositivo utilizado para conectar dispositivos a un bus CAN, permitiendo la transmisión de datos entre equipos. En este caso, se utilizó para comunicar el ordenador que aloja el programa ©MATLAB con los drivers del cuello robótico.

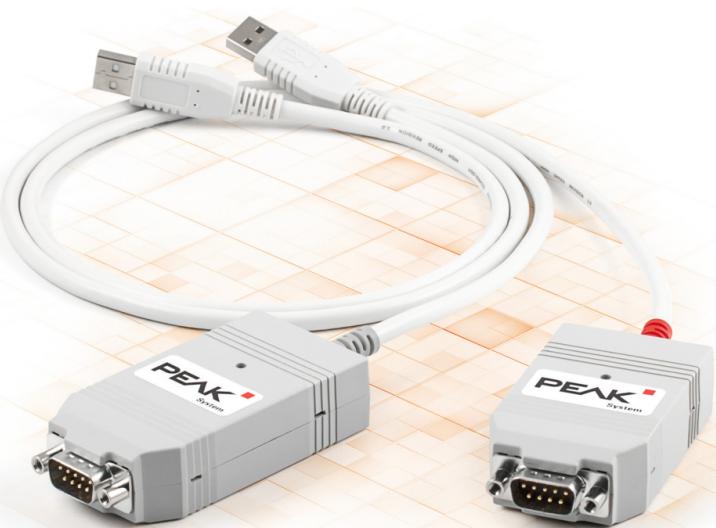


Fig. 1.4. Adaptador PCAN-USB de PEAKSystem [11].

Driver iPOS4808 MX-CAN: es un controlador de motor de alta precisión, de reducido tamaño que es utilizado en aplicaciones de robótica para el control de motores de corriente continua y motores paso a paso [12]. En la figura 1.5 se muestra el esquema de identificación, extraído del manual técnico del driver.



Fig. 1.5. Etiquetas identificadoras iPOS4808 MX-CAN [13].

Es compatible con el protocolo de comunicación CANopen lo que implica que cumple con las capas especificadas en los estándares CiA402 y CiA301. Cuando trabaja en una red CAN ofrece al nodo maestro la opción de llamar a los objetos de dirección almacenados en su memoria EEPROM. Esto le da acceso una amplia gama de características avanzadas, como la regulación de velocidad y posición, el control de par, la detección de errores y la protección de sobrecarga, entre otros.

Sensor inercial 3DM-GX5-10 [14]: es un dispositivo (figura 1.6) que combina la medición de la velocidad angular y la aceleración y que tras un procesado de la información proporciona mediciones de la orientación y la posición en tiempo real de un objeto . Se conecta por el puerto USB del ordenador, aunque trabaje bajo las condiciones del puerto serie. Este está situado en la plataforma superior del cuello detectando cada uno de los cambios de posición de este durante la ejecución del programa.



Fig. 1.6. Sensor 3DM-GX5-10 [14].

1.5.2. Software

©MATLAB: es el software sobre el que se desarrolla la librería de comunicaciones para el cuello robótico. Es una plataforma de programación y cálculo que permite realizar análisis, simulaciones, creación de algoritmos y desarrollar aplicaciones en diversos campos, como la ingeniería. Es una herramienta ampliamente utilizada en investigación debido a sus múltiples funcionalidades y su capacidad para procesar grandes cantidades de datos. La versión utilizada en este trabajo es la R2022b para Windows de 64 bits.

PCAN-View: es un software desarrollado por la empresa alemana PEAK-System Technik GmbH para la visualización y análisis de mensajes transmitidos a través del bus CAN, que es la misma que fabrica el adaptador que veremos en la sección 1.5.1.

La herramienta permite visualizar los mensajes CAN en tiempo real, detectando errores y fallos en la comunicación. Además, PCAN-View cuenta con una interfaz gráfica de usuario (GUI) que, como se observa en la figura 1.7, facilita la configuración de los dispositivos y la interpretación de los datos.

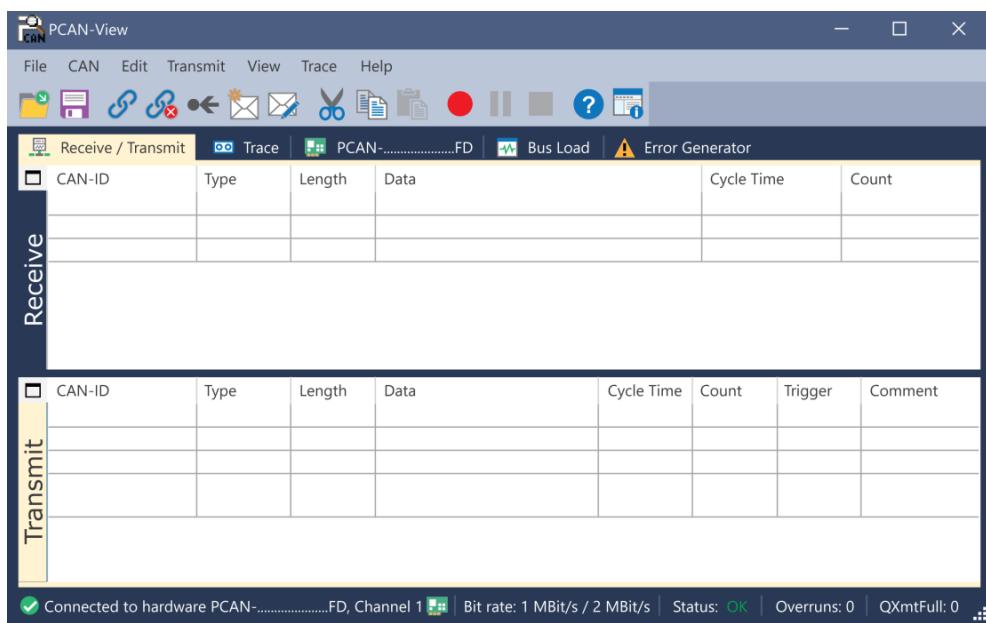


Fig. 1.7. Interfaz gráfica del software PCAN-View [11].

Serial Port Monitor: es un software que permite realizar la monitorización y el análisis de datos que se envían y reciben a través del puerto serie del ordenador. Muestra la información en tiempo real y se utilizó para comprobar los mensajes recibidos por parte de los drivers.

SensorConnect: es un software utilizado para configurar y monitorear en tiempo real el sensor inercial. En la figura 1.8 se observa una medición puntual de los datos que son enviados por el sensor durante su utilización.



Fig. 1.8. Datos recibidos por el sensor inercial.

1.6. Estructura de la memoria

Este documento está dividido en siete capítulos, incluido el presente **capítulo 1, Introducción**. En esta sección se incluye un breve resumen del contenido de los capítulos restantes:

Capítulo 2 Estado del arte: en este capítulo se busca dar un contexto a la situación de partida de este trabajo. Para ello se describirán algunos de los sistemas de comunicación empleados en la robótica, mencionando los casos de otros robots humanoide que existen en la actualidad. Se comentará la librería de comunicaciones en C++ que utilizan en el proyecto HUMASoft y la interfaz gráfica desarrollada. Por último, se darán los motivos por los cuales se ha escogido MATLAB como soporte para el desarrollo de la librería.

Capítulo 3 Comunicación en el protocolo CANopen: se explicarán los aspectos que engloban a el protocolo CANopen, desde sus orígenes en el desarrollo del modelo OSI, pasando por las características del protocolo CAN que le sirven de base. Se indicarán las capas de las que están compuestos estos protocolos y cuál es el tratamiento que le dan a los mensajes durante las transmisiones.

Capítulo 4 Desarrollo de la librería: en este capítulo se describe las distintas etapas de desarrollo del proyecto, así como la información necesaria que permita avanzar al proyecto en sus distintas fases hasta llegar al resultado final. Se explicarán los métodos de las clases creadas para el control del cuello robótico, aquellas por las cual se establece la comunicación, se realizan los intercambios de información y se controlan los motores. Adicionalmente, se hablará sobre la cinemática inversa utilizada para el correcto manejo de las consignas y las funciones del sensor inercial.

Capítulo 5 Resultados experimentales: se presentarán los resultados y conclusiones obtenidos durante las pruebas de validación del funcionamiento del programa creado.

Capítulo 6 Presupuesto del proyecto: se indicará el impacto socioeconómico en el que este proyecto podría incurrir y se incluyen los costes de ejecución del mismo. Al ser este un proyecto de desarrollo de software las fuentes de gasto se engloban en la mano de obra y los distintos softwares empleados.

Capítulo 7 Conclusiones y trabajos futuros: contiene las conclusiones de los hitos cumplidos y se aporta una sección de propuestas para trabajos futuros.

2. ESTADO DEL ARTE

2.1. Sistemas de comunicación empleados en la robótica

El sistema de comunicación o bus es un medio por el cual diferentes dispositivos son conectados mediante líneas a una unidad principal con el fin de permitir la transferencia de información entre los elementos que lo componen. Se puede monitorizar y sirve para ayudar en la coordinación de las acciones de los nodos que intervienen en los procesos.

Aunque algunos robots humanoides pueden utilizar el protocolo de comunicaciones CAN (Control Area Network) en alguna parte de su sistema de comunicaciones, este protocolo no es el único utilizado en robots humanoides. Ya que otros protocolos de comunicación son igualmente válidos y adecuados para la comunicación entre los componentes de los robots.

Estos utilizan diferentes protocolos y arquitecturas de comunicación a nivel de hardware para interconectar sus componentes electrónicos, sensores y actuadores. A continuación, veremos algunos robots humanoides y los protocolos que emplean.

- **Robot humanoide iCub:** el robot de la figura 2.1 fue construido por el Instituto Italiano de Tecnología. Utiliza una arquitectura modular y distribuida para la comunicación a nivel de hardware entre los diferentes componentes del robot, incluyendo los sensores y actuadores [15]. Para ello utiliza un conjunto de tarjetas de control diseñadas a medida para encajar en el robot basadas en tecnología **DSP** o **Digital Signal Processor**.

Los **DSP** son unos sistemas basados en procesadores que poseen un conjunto de instrucciones, software y hardware optimizados para aplicaciones que trabajen en tiempo real. Las tarjetas de DSP se encargan del control de bajo nivel en tiempo real y se comunican entre sí a través de un bus CAN. El humanoide cuenta con cuatro líneas de bus CAN que conectan partes del robot.

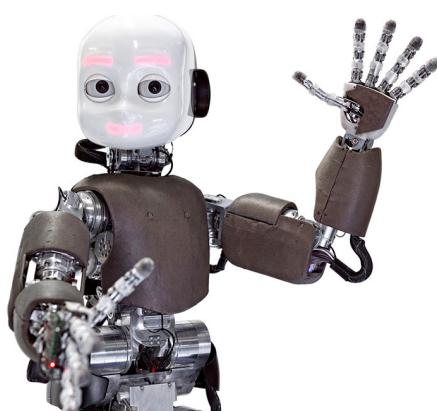


Fig. 2.1. Robot humanoide iCub.

Además, durante parte del desarrollo de este proyecto, han estado utilizando la arquitectura de software de robot YARP (Yet Another Robot Platform) que es un paquete de software de código abierto escrito en C++. Los métodos de este paquete sirvieron de ayuda al acoplamiento y organización de la comunicación entre sensores, procesadores y actuadores [16].

- **Robot humanoide TALOS:** creado por la empresa española PAL Robotics utiliza el bus CAN para la comunicación de sus sensores, pero no en todos los casos. En los sensores de torque se está utilizando el protocolo EtherCAT (Ethernet for Control of Automation Technology) ya que estos dispositivos requieren velocidades de acceso a los datos de muy alta frecuencia, superiores a 1 KHz que sí soportaría CAN [17].
- **Robot humanoide HRP-2:** diseñado por la empresa japonesa Kawada Industries, presenta el aspecto que se puede observar en la figura 2.2. A pesar de trabajar con el protocolo de comunicaciones CANBus, para tratar de reducir los posibles ruidos generados por los controladores de los servomotores, se han incorporado a la placa del robot un bus PCI (Peripheral Component Interconnect). Este es un bus estándar que tiene como objetivo conectar dispositivos periféricos directamente a la placa base [18]. Además, como también se indica en el artículo del robot Talos [17], HRP-2 utiliza tecnología EtherCat para la comunicación entre sus sensores.



Fig. 2.2. Robot humanoide HRP-2.

- **Robot humanoide TEO:** en la figura 2.3 se muestra el robot humanoide desarrollado por la Universidad Carlos III de Madrid el cual utiliza el protocolo CANopen para el bus de comunicación interna de sus componentes eléctricos y sensores.

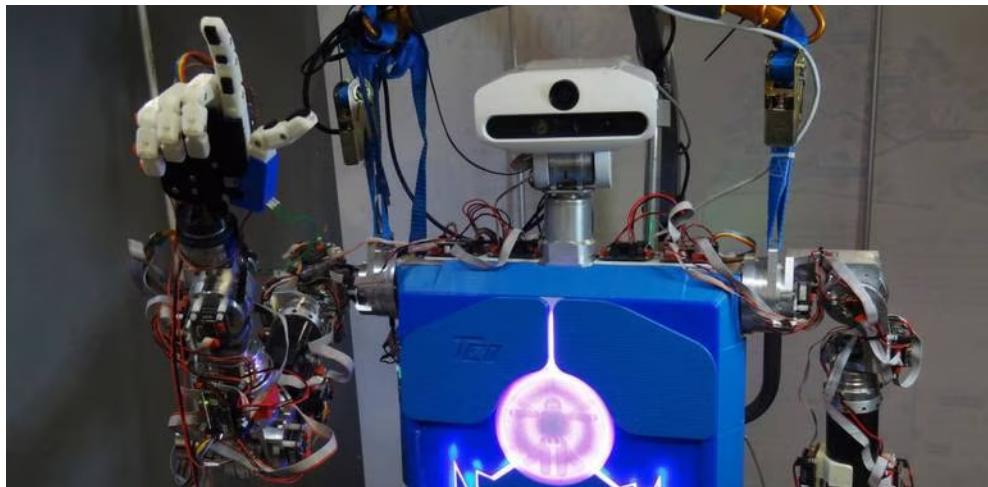


Fig. 2.3. Robot humanoide TEO.

Se puede ver que los distintos proyectos presentan diversas soluciones para la comunicación interna de los elementos que conforman cada robot. Se muestra claramente la popularidad del protocolo CAN, ya que se encuentra presente en todos los desarrollos y por ello se le dedicará el capítulo 3 de esta memoria para profundizar en sus características.

A continuación, se van a describir brevemente alguno de los buses de comunicación más utilizados en robótica y alguno de los mencionados en los robots antes descritos.

2.1.1. Protocolo EtherCAT

Es un protocolo informático para la industria, de código abierto, desarrollado por Beckhoff Automation [19] que trataba de buscar un sistema de bus de campo que fuera capaz de trabajar en tiempo real que hiciera un buen uso del ancho de banda, sí lo comparamos con los estándares de Ethernet clásicos.

Bajo estas premisas nació EtherCAT siendo uno de los sistemas de comunicación más rápidos actualmente que amplía el estándar IEEE 802.3 Ethernet. Este protocolo permite una comunicación de sincronización precisa y con una temporización predecible.

Entre sus ventajas cabe destacar que soporta casi cualquier tipo de topología convirtiéndolo en un sistema flexible [20] pudiendo ser cableado en línea, árbol o estrella. Puede utilizar todo el ancho de banda de una red de Ethernet estándar. Bajo este tipo de red se pueden conectar hasta 65535 dispositivos sin importar la topología. Además, es compatible con las especificaciones de CANopen, esto ayuda a una fácil configuración de las redes de comunicación [21].

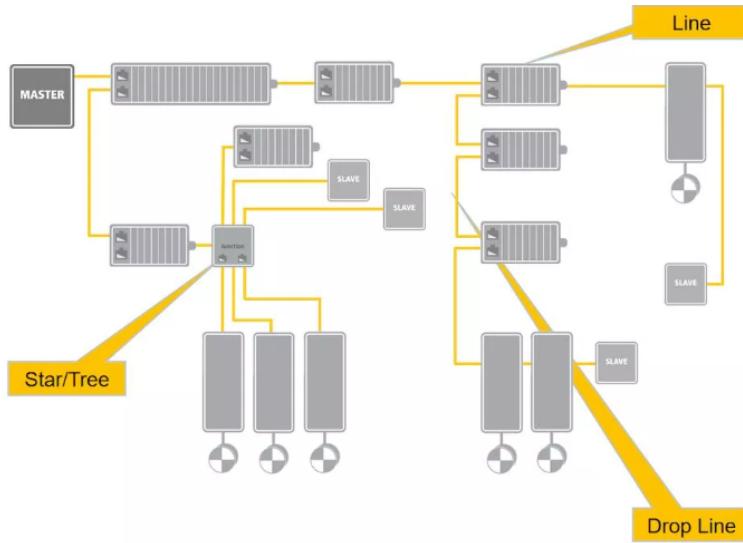


Fig. 2.4. Ejemplo de topología de conexionado en EtherCAT [22].

En la capa de datos del protocolo EtherCat se codifican los datos en paquetes, pero se diferencia con el protocolo de Ethernet en que este omite las capas de red (IP) y de transporte (TCP y UDP). El maestro envía un telegrama que pasa por cada nodo. Los dispositivos de la red EtherCAT lee los datos direccionalmente a él e inserta sus datos en una trama nueva, con lo que los retardo se deberán únicamente a los tiempos de retardo del hardware. Se produce una alta tasa de transferencia de datos, incluso superior a 100 Mbit/s. El maestro EtherCAT es el único nodo que puede enviar activamente tramas en la red, mientras que los demás únicamente reenviarán tramas [20][23].

2.1.2. Protocolo RS-232

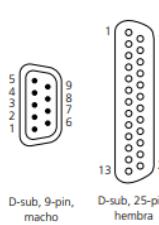
La interfaz RS232 o TIA/EIA-232 es un protocolo de transferencia de datos diseñado para controlar la comunicación serie creado en la década de los 60 [24], que originalmente se usaba para conexiones a módems, impresoras y otros dispositivos periféricos de ordenadores. Es utilizado en transmisión de datos para comunicaciones punto a punto, donde la información es transmitida bit a bit a través de un par de hilos que conforman la línea y por unos pines específicos de los conectores [25].

Los circuitos que siguen este protocolo están formados por un cable de conexión con conectores RS-232 que tendrá un hilo de transmisión de datos, que será el cable de recepción en el otro extremo del circuito, y un hilo de recepción, que corresponde con la transmisión en el otro extremo.

Los conectores RS-232 utilizados para la conexión de los distintos elementos se denominan conectores DB-25 (compuestos por 25 pines), aunque comúnmente se usa una versión más pequeña que esta formada por 9 pines y es denominado DB-9. A continuación, se muestra en la figura 2.5 los dos tipos de conectores y las diferencias de funcionalidad de sus patillas.

CAPÍTULO 2. ESTADO DEL ARTE

Séñal	Patilla D-Sub -9	Patilla D-Sub -25	Dirección de la señal DTE	Dirección de la señal DCE	Norma CCITT V.24	Nombre / función
TXD	3	2	Salida	Entrada	103	Datos transmitidos
RXD	2	3	Entrada	Salida	104	Datos recibidos
RTS	7	4	Salida	Entrada	105	Petición de envío
CTS	8	5	Entrada	Salida	106	Preparado para enviar
DTR	4	20	Salida	Entrada	108	Terminal de datos preparado
DSR	6	6	Entrada	Salida	107	Equipo de datos preparado
DCD	1	8	Entrada	Salida	109	Detección de portadora de datos
SGND	5	7	—	—	102	Toma de tierra de la señal



D-sub, 9-pin, macho D-sub, 25-pin, hembra

Fig. 2.5. Señales de la norma V.24 y tipos de conectores [26].

En el conector clásico, el pin número 2 de estos conectores es el de transmisión de datos (Tx), mientras que el pin 3 es el de recepción (Rx). Pero en el caso de los conectores DB-9 estos pines están invertidos como se puede observar en la figura 2.6.

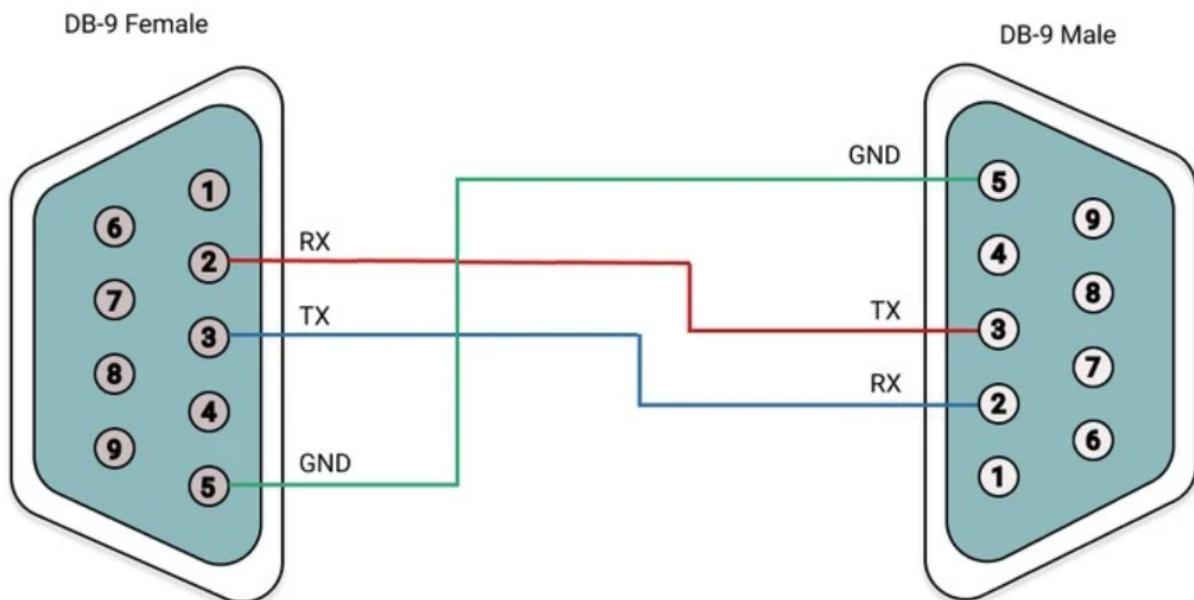


Fig. 2.6. Conexión serial DB9 para comunicaciones RS-232 [27].

Los valores de tensión que representarán la comunicación binaria deben encontrarse entre unos márgenes adecuados en el que se tiene en cuenta las caídas de voltaje a lo largo del cable: por lo que la norma, CCIT V.24, indica que para las tensiones de +3V y +15V se considerarán como un cero lógico y para el uno lógico se usan valores de -3V y -15V.

Las tramas de comunicación RS-232 se componen de varios bits, como se indica en la tabla 2.1. En primer lugar se encuentra el bit de inicio que indica al receptor que está a punto de recibir datos, preparándolo para leer los datos con la sincronización correcta. Continúa con el frame de datos que cuenta con 8 bits, aunque esto puede variar entre 5 y 9 bits.

Después de los datos transmitidos aparece el bit de paridad, que conforma el método de detección de errores más simple. El bit de paridad especifica si el número de bits de los datos recibidos es par (0) o impar (1). Por último, recibirá el bit de parada que marca el fin de la transmisión.

Frame:	Start Bit	Data Bits	Parity Bit	Stop Bit
size [# of bits]:	1	5-9	0-1	1-2

TABLA 2.1. SEÑAL DE TRANSMISIÓN RS-232 DESGLOSADA
POR BITS.

Se debe tener en cuenta que un circuito realizado con la conexión RS-232 tiene una longitud limitada, que según la norma V-24, se corresponde a unos 15 metros. Si el cable excede esa distancia pueden producirse pérdidas de datos debido a la falta de calidad en las señales que circulan por el circuito. Aunque la velocidad de transmisión máxima del sistema sea de 20 kbits/s en las situaciones en las que las conexiones entre equipos alcancen el límite la velocidad de transmisión será de unos 38,4 kbit/s.

La comunicación puede realizarse asíncrona o síncrona en diferentes tipos de canal [28]:

- Simplex: es un canal de comunicación que envía información en una sola dirección.
- Half duplex: en este modo se pueden transmitir datos en ambos sentidos del circuito, pero no simultáneamente. El circuito se compone físicamente de dos hilos.
- Full duplex: se proporciona una mejora considerable en la transmisión de datos debido a que se puede transmitir en los dos sentidos del circuito , ya que éste se compone de cuatro hilos, mejorando la velocidad de transmisión así como la calidad de los datos.

Como inconveniente, estos sistemas se encuentran con la falta de capacidad de comunicarse en tiempo real. Sin embargo, la tasa de baudios o la velocidad de comunicación son fijos, independientemente del tráfico de la red.

2.1.3. Protocolo RS-485

El protocolo RS485 o EIA-485 es similar al RS-232, pero con mayores velocidades, capacidad multipunto y señalización diferencial lo que la ha convertido en la interfaz más utilizada en la industria. Fue publicado como estándar en 1983 conjuntamente por Electronic Industries Alliance (EIA)¹ y la Telecommunications Industry Association (TIA/EIA)²

¹EIA es una organización formada por la asociación de las compañías electrónicas y de alta tecnología de los Estados Unidos

²TIA/EIA es una asociación que tiene como fin el desarrollo de normas industriales en el ámbito de las tecnologías de la información y la comunicación (TIC)

como una especificación de la norma RS-449 [25].

En esta norma se especifican características eléctricas del sistema de comunicaciones serie digital. Los cables RS-485 solo tienen tres hilos, 2 para transmisión de datos y 1 para tierra. Pueden ser conectados en topologías multipunto, se pueden conectar múltiples receptores y transmisores, hasta un total de treinta y dos elementos de cada rol en full duplex. Se logra enlazar los procesadores de comunicación principal, denominados maestros, con los esclavos subordinados de la red.

Para una mayor coherencia, la transmisión de datos se realiza mediante señales diferenciales. Se llama señal diferencial a aquella que viaja por dos conductores, llamados A y B, en lugar de hacerlo por uno solo, de tal modo que las tensiones y corrientes en los conductores sean simétricas. Se emplean señales diferenciales debido a que son más robustas frente a las interferencias. El alcance de la transmisión de estos sistemas refleja que se pueden construir redes de una longitud máxima de 1200 m y una velocidad máxima de 10 Mbits/s que se obtiene a una distancia de 12 m.

- A: tensiones positivas para cero lógico y tensiones negativas para uno lógico.
- B: tensiones negativas para cero lógico y tensiones positivas para uno lógico.

Parámetros	RS-232	RS-485
Configuración	Único	Diferencial
Modo habitual de operación	Simplex o full duplex	Simplex o half duplex
Número de elementos por línea	1 emisor y 1 receptor	32 emisores y 32 receptores
Máxima longitud de cable	15 m	1200 m
Velocidad máxima de datos	20 kbits/s	50 kbits/s
Niveles lógicos típicos	± 5 a ± 15 V	$\pm 1,5$ a ± 6 V
Impedancia mínima de entrada del receptor	3 a 7 Ω	12 Ω
Sensibilidad del receptor	± 3 V	± 200 mV

Fuente: Virtual serial port [29]

TABLA 2.2. DIFERENCIAS ENTRE RS-232 Y RS-485.

Entre las principales diferencias entre los puertos serie RS-232 y RS-485 se encuentran: el modo de funcionamiento, los rangos de distancia de los circuitos, los niveles de voltaje para la comunicación binaria o el número de elementos conectados, entre otros. En la tabla 2.2 se muestran algunos de los parámetros que difieren entre ambos.

2.2. Librería desarrollada en el proyecto HUMASoft para la comunicación de los drivers

Dentro del marco del proyecto Humasoft se ha desarrollado una librería de comunicación para la implementación y puesta en marcha de prototipos en el robot TEO. Esta se encuentra en un continuo proceso de desarrollo, la cual debe responder a las necesidades de los distintos equipos de investigación.

Es fundamental el desarrollo de este tipo de librerías para que el usuario efectúe de una forma sencilla y eficiente el posicionamiento de toda parte robótica móvil controlable. Por lo que el objetivo principal del desarrollo de estas librerías es lograr la correcta comunicación, la monitorización y el control de los elementos actuadores que se comunican siguiendo el protocolo CANopen. Para facilitar la comprensión y modificación de las funciones a voluntad, la librería ha sido realizada en lenguaje de programación C++.

La programación en C++ se concibió como una extensión del lenguaje de programación C, al que se le añaden mecanismos que permiten la manipulación de objetos. A parte de habilitar la programación orientada a objetos, la programación bajo este lenguaje dota al usuario de otras ventajas:

- Existe una comunidad que sustenta y da soporte a este lenguaje, lanzando una gran cantidad de actualizaciones que lo mantiene vigente.
- Es multiplataforma, al poder ejecutarse fácilmente en una amplia variedad de hardware y software. Esto lo lleva a ser empleado en el desarrollo de una gran variedad de productos: creación de videojuegos, aplicaciones de escritorio, bases de datos, sistemas operativos, entre otros.
- Este lenguaje es bastante bueno en el momento de trabajar o desarrollar sistemas de gestión de base de datos, garantizando el intercambio, consulta o la actualización de datos.
- No está supeditado a un único compilador, si no que se pueden utilizar diferentes.
- Lenguaje versátil que tiene soporte para los distintos paradigmas de la programación como lo son: modular, lógico, funcional, imperativo, estructurado o la orientación a objetos ya mencionada.

La librería de comunicaciones [9] se ha diseñado para que sea compatible para cualquier dispositivo CANopen que posea un motor. En consecuencia, se han seguido los estándares CiA301, en el que se describen los tipos de datos que se pueden utilizar, y CiA402 [30]. Este último es una norma específica para accionamientos y controles de movimiento.

Las clases que componen la librería cuentan con una serie de métodos que gestionan las comunicaciones con los drivers, el envío y el cambio de configuración.

La clase principal de la aplicación **CiA402Device** utilizado para comandar los motores, permite configurar los distintos modos de operación y lleva a cabo las acciones demandadas por el usuario. La clase **CiACommPort** sirve de apoyo a la clase CiA402Device y es la que permite el envío y recepción de los distintos tipos de mensajes CAN.

Posteriormente, existen una serie de clases que trabajan a más bajo nivel en las que se opera sobre los puertos y el bus de datos. La clase **PortBase** es la clase principal de la que heredan de las demás, y es la encargada de permitir escribir en el puerto, limpiar los mensajes remanentes y obtener el mensaje que haya en el bus. De la clase PortBase heredan otras tres clases:

- **CanBusPort**: planteada para la comunicación byte a byte, pero es poco utilizada.
- **SocketCanPort**: los métodos de esta clase se utilizan para inicializar los puertos en la aplicación, para la recepción y el envío de datos.
- **TestPort**: clase utilizada para realizar pruebas en los puertos de comunicación durante el desarrollo inicial de la librería. No se utiliza por parte del usuario.

Los métodos utilizados en las clases siguen una estructura jerárquica en las que las funciones complejas de alto nivel se ven complementadas por métodos más sencillos. Por esta razón los métodos de alto nivel, que son accesibles por los usuarios, se encargan de la administración de las tareas generales, como podría ser la petición del valor del setpoint de posición. Mientras tanto, los métodos sencillos se encargarían de tareas específicas que llevarían a cabo la acción, como podrían ser: enviar la orden de activación de los motores o el envío del setpoint deseado por el usuario.

Interfaz de usuario

Las interfaces de usuario permiten a los usuarios interactuar de forma más intuitiva y natural con los programas. En lugar de tener que escribir comandos o código, los usuarios pueden hacer clic en botones, arrastrar elementos y ajustar controles deslizantes para realizar tareas y obtener resultados en tiempo real. Además, las interfaces de usuario pueden ayudar a simplificar tareas complejas y reducir los errores del usuario. También pueden proporcionar una mejor experiencia de usuario al agregar elementos gráficos y visuales, lo que puede mejorar la comprensión de los usuarios sobre la información presentada.

Con el objetivo de poder interactuar de una forma más fluida con el programa que controla el cuello robótico, se creó una interfaz de control en el año 2020 [31]. La interfaz de la figura 2.7 se realizó sobre los lenguajes de programación Python y C++.

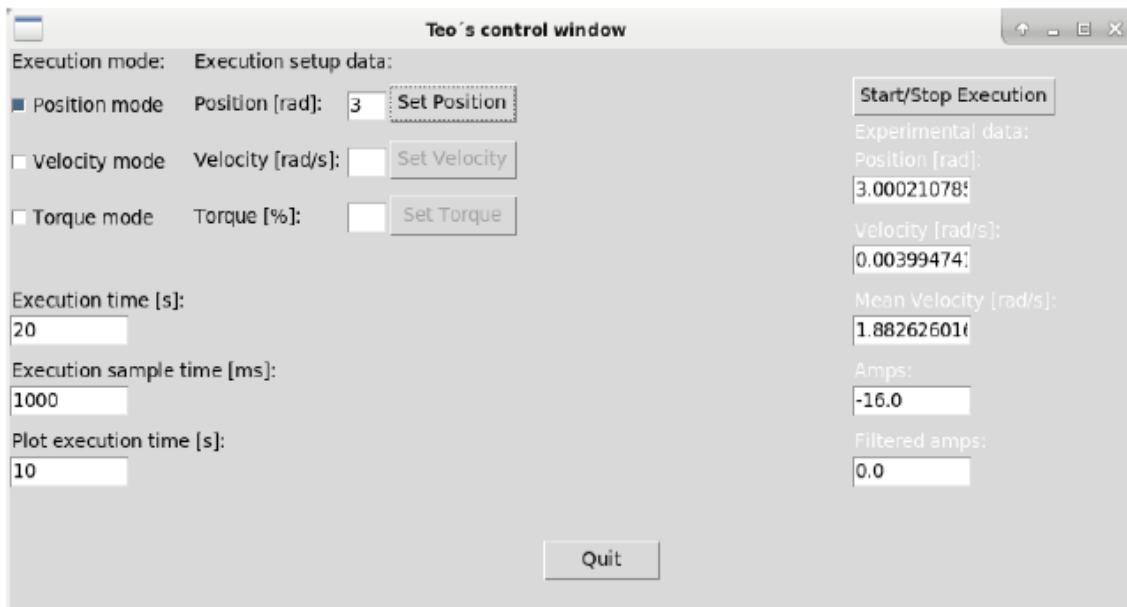


Fig. 2.7. Interfaz de control de posición.

Para la fase desarrollo de la interfaz se añadió un nuevo lenguaje de programación Python, el cual se podía integrar fácilmente con la librería ya desarrollada en el proyecto HUMASoft. Al ser un sistema multiplataforma las aplicaciones de GUI (Graphical User Interface) escritas en Python pueden ser portables y ejecutarse en diferentes sistemas operativos.

Este lenguaje de programación es una excelente opción para el desarrollo de interfaces gráficas de usuario o GUI. cuenta con una gran cantidad de bibliotecas y herramientas para la creación, incluyendo PyQt, PySide, wxPython, Tkinter, Kivy, y muchas más. Estas bibliotecas proporcionan una variedad de componentes de interfaz de usuario, como botones, cuadros de texto, menús y gráficos, lo que permite una gran flexibilidad en el diseño y la funcionalidad.

2.3. Elección de MATLAB para diseñar una nueva librería de comunicaciones

Actualmente no existe ninguna librería de **MATLAB** dentro del proyecto SOFIA, ya que la librería de C++ ha cubierto las necesidades que han surgido hasta el momento. En esta sección se tratará de presentar el potencial de este software para el desarrollo de aplicaciones abriendo nuevos horizontes al desarrollo de nuevo equipamiento para TEO en un entorno virtual.

MATLAB es una herramienta popular en la ingeniería y la ciencia debido a su amplia funcionalidad para el análisis y procesamiento de datos, así como a su facilidad de uso y programación. Se puede realizar la aplicación en el lenguaje de programación propio de MATLAB, que es cómo se ha realizado en este trabajo, y a partir de este punto se abre la posibilidad a la utilización de las demás herramientas que posee esta plataforma.

Una de las ventajas de utilizar MATLAB para realizar programas en tiempo real es que tiene una biblioteca extensa de funciones matemáticas y herramientas de simulación que permiten la implementación de algoritmos complejos de manera rápida y eficiente. Además, tiene herramientas para la visualización de datos, lo que facilita la depuración de los programas.

Otra ventaja es que proporciona interfaces para una variedad de hardware y sistemas embebidos, lo que permite la conexión de sistemas en tiempo real.

Cuenta con la herramienta denominada **Simulink**, que proporciona una interfaz gráfica para diseñar y simular sistemas de control en tiempo real. Simulink permite la simulación de modelos de sistemas de control en tiempo real y la integración con hardware en tiempo real para la implementación del sistema de control diseñado.

El **Instrument Control Toolbox** es una herramienta de que permite controlar dispositivos y sistemas en tiempo real a través de diferentes buses de comunicación, como USB, Ethernet, RS-232, entre otros. Esta toolbox proporciona una interfaz de programación para controlar hardware y adquirir datos, lo que facilita el proceso de diseño de sistemas de control y monitoreo en tiempo real.

Incluye una amplia gama de funciones y herramientas para la comunicación con equipos. Permitiendo interactuar con dispositivos a través de una variedad de protocolos de comunicación, con el envío y recepción de comandos se pueden adquirir datos en tiempo real y guardarlos para su posterior procesamiento.

Otra de las aplicaciones es **Data Acquisition Toolbox** que es una herramienta que permite adquirir y analizar datos de diferentes tipos de dispositivos. Entre las funciones que ofrece se encuentran la capacidad de configurar canales de entrada y salida, adquirir datos de forma en tiempo real, la posibilidad de realizar pruebas de hardware y software, la posibilidad de controlar dispositivos remotamente y la integración con otras herramientas de MATLAB para análisis y visualización de datos.

Estas han sido tres de las múltiples aplicaciones que los usuarios pueden llegar a utilizar y que se encuentran disponibles en diversas plataformas como Windows, macOS y GNU/Linux. Para comenzar a utilizar la amplia gama de herramientas, la compañía MathWorks pone a disposición de los usuarios su **Help Center**. Dentro de esta plataforma se accederá a la documentación de MATLAB y de sus productos complementarios, incluyendo ejemplos de código, tutoriales, videos y otros recursos de aprendizaje.

A diferencia de la librería de C++, la programación realizada en MATLAB contará únicamente con una clase para el control de los dispositivos del bus. La configuración del puerto se realizará una vez se creen los objetos de la aplicación y se realizará toda la gestión de la información y de las órdenes desde la clase creada.

3. COMUNICACIÓN EN EL PROTOCOLO CANOPEN

3.1. El protocolo CAN

En esta sección se va a explicar el funcionamiento y la historia del protocolo CAN, uno de los protocolos de comunicación más utilizados en la industria automotriz, aeroespacial, robótica y otros campos.

3.1.1. Origen del protocolo CAN

El protocolo CAN es un protocolo de comunicación diseñado por la compañía alemana Robert Bosch Stiftung GmbH, en la década de 1980, para el intercambio de información entre distintos dispositivos electrónicos. Desde entonces, ha evolucionado y se ha convertido en un estándar internacional que se utiliza en una amplia variedad de aplicaciones industriales y comerciales.

Fue rápidamente introducido en el sector de la automoción para responder a la necesidad de reducir la cantidad del cableado utilizado para comunicar el creciente número de dispositivos y microcontroladores que debían interactuar entre sí. Para dimensionar el problema que este sistema resolvió, se debe entender que en aquella época cada dispositivo contaba con una conexión punto a punto independiente. Por lo que, al conseguir conectar todos aquellos elementos a un bus de comunicación en serie, se redujo considerablemente el peso de los vehículos.

El protocolo CAN sigue el modelo OSI (Open Systems Interconnection) que es un marco de referencia teórico para describir la arquitectura de las redes de comunicación. Este es un modelo de referencia para la comunicación de red desarrollado y promovido por el ingeniero de software francés Hubert Zimmermann, el cual fue uno de los primeros miembros de la Organización Internacional de Normalización (ISO)[32].

El objetivo principal del modelo OSI es estandarizar la forma en la que los dispositivos de red se comunican entre sí, permitiendo la interconexión entre equipos de diferentes fabricantes y sistemas operativos sin problemas. Para ello consta de siete capas, cada una de las cuales desempeña una función específica en el proceso de comunicación de red.

Capa de aplicación: es la capa más alta en la arquitectura OSI en la cual se definen los protocolos de comunicación específicos utilizados por las aplicaciones para intercambiar datos. Comprende las funciones de inicio, manutención, finalización y registro de datos relacionados con las conexiones de la transferencia de datos entre los equipos de la aplicación.

Capa de presentación: encargada de proporcionar a la capa de aplicación la posibilidad de interpretar el significado de los datos intercambiados entre los equipos de la red.

Con lo que gestiona el intercambio, la visualización, el formato y el control de datos.

Capa de sesión: establece, mantiene y termina las sesiones de comunicación entre los dispositivos de la red, para ello puede emplear los servicios proporcionados por la capa de transporte.

Capa de transporte: proporciona un servicio de transporte confiable y garantiza que los datos lleguen correctamente al destino, liberando a los dispositivos de cualquier preocupación por la forma en la que se logra una óptima transferencia de datos.

Capa de red: maneja la comunicación entre dispositivos de la red, independientemente de la ruta o destino, y se encarga de enrutar los paquetes de datos a través de esta.

Capa de enlace de datos: se encarga de la transmisión fiable de los datos a través del medio físico: estableciendo, manteniendo y liberando los enlaces de datos entre las entidades de la red.

Capa física: define los medios físicos de transmisión de datos, como los cables o la fibra óptica.

3.1.2. Capas del modelo OSI implementadas en el protocolo CAN

En cuanto al protocolo en cuestión, la compañía Bosch estuvo publicando nuevas versiones de las especificaciones de CAN hasta que en 1993 se estandarizó el protocolo de comunicaciones en la ISO 11898. Caben destacar algunas de sus principales propiedades:

- Cada uno de los elementos que componen la red posee la capacidad de enviar y recibir información, realizándose por medio de mensajes estructurados, de una longitud determinada y formados por cadenas binarias.
- Cuenta con medios para asegurar que las transmisiones de las tramas se realicen correctamente, como dando prioridad a los mensajes para no ocasionar una colisión en el bus. Si alguna de las unidades que componen la red tuviera algún problema, esta quedaría fuera de servicio y el bus continuaría funcionando.
- En este tipo de estructuras se le asigna un número identificador (ID) único a cada nodo. La distribución de los identificadores se asocia a una estructura maestro-esclavo, en la que el maestro conoce los ID de todos los esclavos conectados, con un máximo de 127 elementos.
- Una de las características importantes de los buses CAN es que en los estándares ISO 11898 se definen las dos capas del modelo OSI que este va a implementar, la capa física y la de enlace de datos [33]. Posteriormente la capa de aplicación se definirá en distintos estándares que estarán estrechamente relacionadas con su campo de aplicación.

Capa física

En la capa física se definen los aspectos físicos para la transmisión de datos entre los nodos de la red. Estos engloban las conexiones, la sincronización y los tiempos en los que los bits se transfieren al bus.

Para la conexión de los elementos al bus no existen unos conectores definidos en el estándar. Se suele utilizar conectores D-sub de 9 pines, como el utilizado en el adaptador Peak-System. Para la conexión al bus de las placas de los drivers se utilizan conectores JST.

Los dispositivos creados para trabajar bajo estos estándares cuentan con una amplia variedad de frecuencias en las que pueden realizar las transmisiones. Es necesario tener en cuenta las dimensiones de la línea de transmisión, ya que esta limita el ancho de banda al que pueden trabajar de forma óptima. En la tabla 3.1 se observa una serie de valores límite de longitud en función de la velocidad de transmisión [34].

Longitud del bus (m)	Velocidad de transferencia (kbits/s)
40	1000
100	500
200	250
500	100
1000	50

Fuente: Texas Instrument

TABLA 3.1. RELACIÓN DE LONGITUD Y VELOCIDAD DE TRANSMISIÓN.

Para este proyecto se ha establecido esta distribución con una velocidad de transmisión de 1 Mbit/s para comunicar con los drivers, al ser las longitudes del bus mucho menores a 40 metros.

Cable H

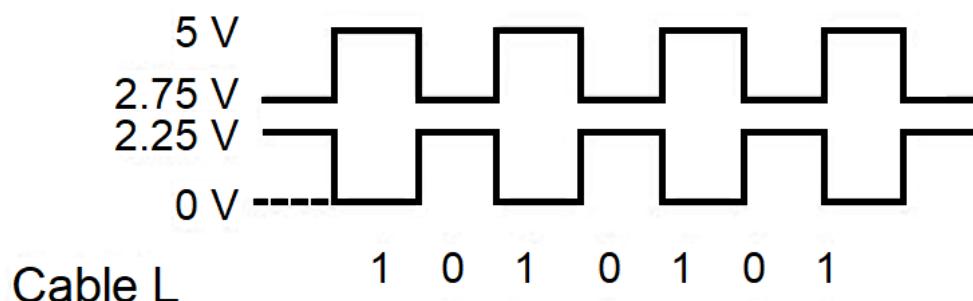


Fig. 3.1. Transmisión por diferencia de tensión.

La información circula por dos cables trenzados y se transmite por diferencia de tensión entre los cables, de forma que: un valor alto de diferencia de tensión representa un 1 y un valor bajo representa un 0, como se observa en la figura 3.1. En uno de los cable los valores oscilan entre 0-2,25V (cable L-Low) y en el otro lo hacen entre 2,75-5V (cable H-High).

Los datos en la línea son transmitidos en forma de mensajes estructurados. Todas las unidades de control reciben el mensaje, lo filtran y solo lo emplean las que necesitan dicho dato. El proceso de transmisión de información se desarrolla siguiendo un ciclo de varias fases.

El mensaje es una sucesión de '0' y '1', que representan diferentes niveles de tensión como los representados en la figura 3.2. Los mensajes son introducidos en la línea con una cadencia de 7-20 milisegundos dependiendo de la velocidad de área y la unidad de mando. Los niveles de tensión representados con '0' se denominan «dominante», mientras que aquellos interpretados con '1' son conocidos como «recesivos».

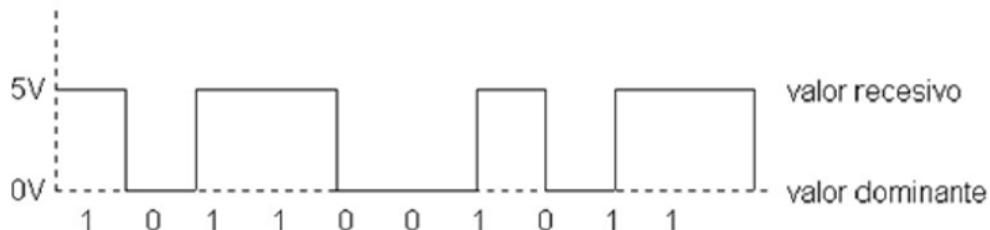


Fig. 3.2. Sucesión de bits que representan niveles de tensión.

Por otro lado, todos los nodos de la red deben trabajar con la misma tasa o frecuencia de transferencia de datos. Dado que es necesario que todos los elementos de la red estén sincronizados, cada bit del bus se divide en al menos 4 cuantos de tiempo, con el objetivo de que los nodos en posiciones extremas sean capaces de recibir y leer correctamente los mensajes transmitidos:

- **Segmento de sincronización:** periodo de tiempo en el que se sincronizan los relojes.
- **Segmento de propagación:** es aquella en la que se compensan los retardos de la propagación de la línea de bus.
- **Segmentos de fase:** se compone de dos segmentos (fase 1 y fase 2) que tienen como objetivo alargar o acortar la duración del bit para la resincronización.

Atendiendo a la velocidad de transmisión de datos, los buses permiten ser distribuidos en diferentes topologías y se pueden catalogar en dos tipos:

CAN de alta velocidad[35]: en la figura 3.3 se observa que la topología está compuesta por una única línea de bus en cuyos extremos se colocan unas resistencias que cierran

el bus, situando los nodos en una conexión en paralelo entre sí. Para evitar perturbaciones en la comunicación, se debe implementar un valor en las resistencias de cierre de $120\ \Omega$, que es la característica de la impedancia del bus. Esta arquitectura de comunicación permite velocidades de transmisión de entre 40 Kbit/s y 1 Mbit/s, con la posibilidad de desconectar alguno de los equipos conectados a esta sin provocar ningún perjuicio a la comunicación con los demás elementos.

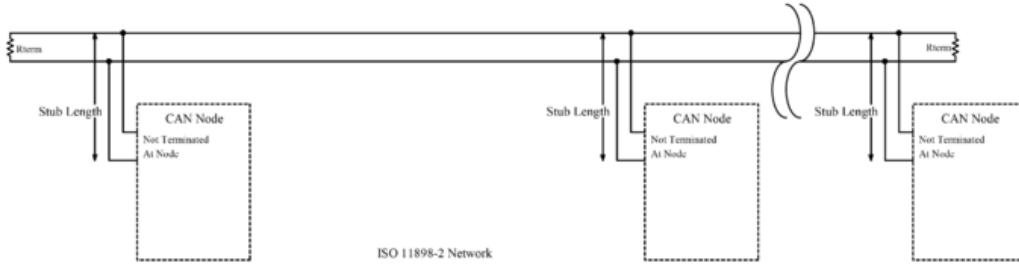


Fig. 3.3. Distribución del bus CAN de alta velocidad. ISO 11898-2 [36].

CAN de baja velocidad tolerante de fallos[37]: bajo esta tipología se posibilita una mayor diversidad de tipos de conexiones tales como buses lineales, buses en estrella (figura 3.4) o la conexión de múltiples buses en estrella. En esta modalidad difiere de la anterior ya que cada nodo actúa como final de línea y cuenta con una resistencia terminadora de un valor próximo a $100\ \Omega$. Este estándar permite la transmisión de mensajes a una velocidad máxima de 125 Kbit/s.

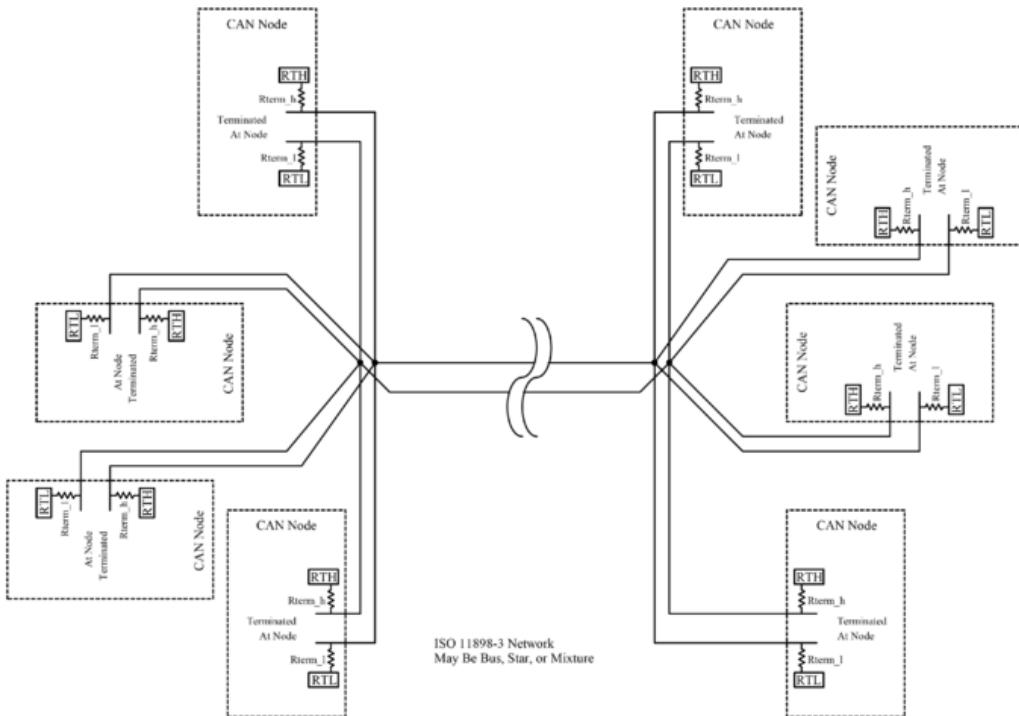


Fig. 3.4. Distribución del bus CAN de baja velocidad. ISO 11898-3 [36].

Capa de enlace de datos

Define las tareas independientes del método de acceso al medio, el intercambio de mensajes que demanda dicho procesamiento requiere de un sistema de transmisión a frecuencias altas y retrasos mínimos. Además una red CAN brinda soporte para procesamiento en tiempo real a todos los sistemas que la integran,

En el 2011 se desarrolló una versión del protocolo denominado CAN FD (flexible data-rate), que es capaz de comprender los mensajes CAN clásicos, favorece un aumento del número máximo de bytes por trama en cada transmisión y que posee la capacidad de transmitir datos por encima de 1 Mbit/s, llegando a ser 8 veces mayor. Esta nueva especificación se recoge en la norma ISO 11898-1:2015[38].

Anteriormente se ha comentado que en el protocolo CAN todos los elementos que conforman el bus pueden transmitir información y esto lo pueden realizar en cualquier momento. Es necesario evitar un conflicto de transmisión en el bus cuando varios de los nodos pretenden emitir de forma simultánea. Para ello se utiliza un método, conocido como de arbitraje, que utiliza la representación lógica dominante y recessiva de los niveles de tensión. El estado base del bus en espera es recessivo, por lo que cuando varios dispositivos pretendan transmitir simultáneamente se producirá una colisión en la que el que aquel que haya enviado un bit dominante prevalecerá. El elemento contrario pasará a un estado pasivo para ceder el paso al nodo dominante. En este apartado es donde la sincronización, antes vista, juega un papel fundamental, ya que la transmisión de tramas ha de ser sincronizada.

El arbitraje se encuentra al inicio de la trama, en el identificador de mensaje. La figura 3.5 es un claro ejemplo de como actúa el método de arbitraje cuando varios nodos tratan de transmitir al mismo tiempo. Los dos nodos que han pasado a un modo pasivo esperarán a que la línea esté disponible para volver a transmitir.

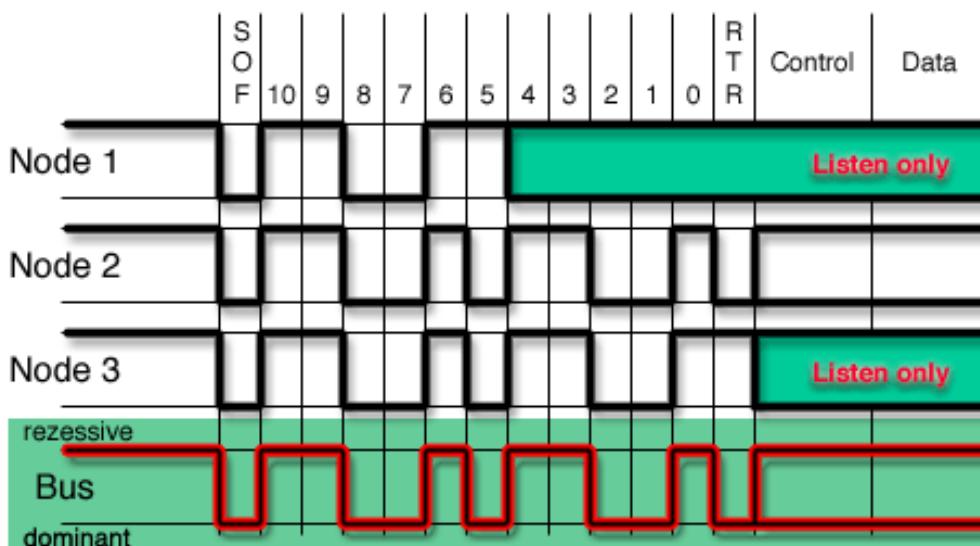


Fig. 3.5. Método de arbitraje [39].

Las tramas que un nodo puede enviar se engloban en cuatro modelos.

Trama de datos: es la trama utilizada para la transmisión de información entre nodos y que pueden presentarse en dos formatos: el **básico** que cuenta con un identificador de 11 bits y el **extendido** con 29 bits.

La estructura del mensaje se compone de una serie de campos en los que cada uno tiene un objetivo. En la figura 3.6 se muestra la estructura en formato básico.

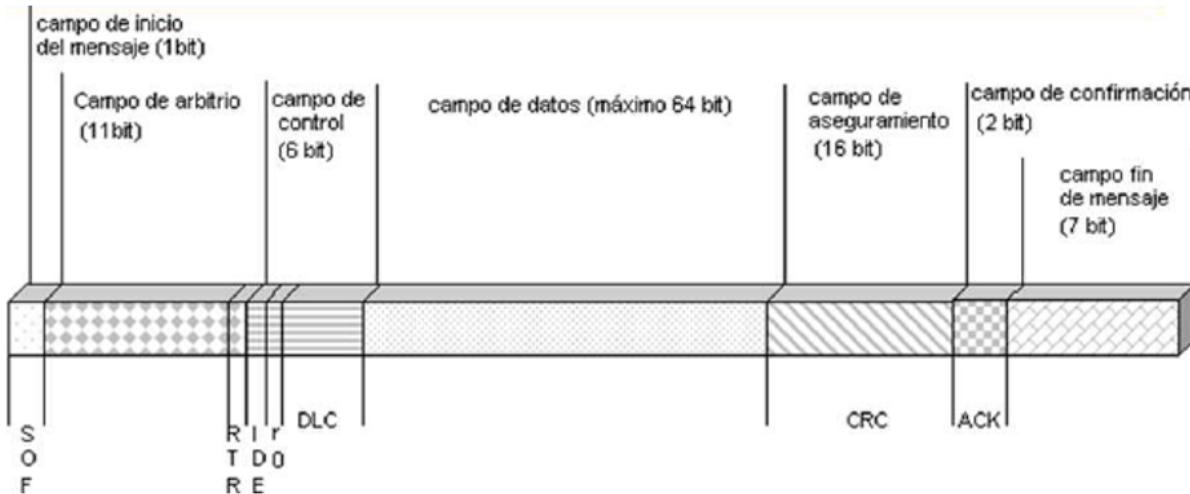


Fig. 3.6. Estructura de trama de datos.

- **Campo de inicio del mensaje:** el mensaje se inicia con un bit dominante, cuyo flanko descendente es utilizado por las unidades para sincronizarse entre sí.
- **Campo de arbitrio:** 11 bits que se emplean como identificador que permite reconocer a las unidades la prioridad del mensaje. El bit RTR indica si la trama contiene datos (RTR=0) o si no los contiene (RTR=1). Una trama de datos tiene prioridad más alta que una trama remota.
- **Campo de control:** característica del campo de datos. El bit IDE indica con un '0' que se trata de una trama estándar y con un 1 si es extendida. Los cuatro bits que componen el campo DLC indican el número de bytes contenido en el campo de datos.
- **Campo de datos:** en este campo aparece la información del mensaje con los datos que se han introducido en la línea CANBus. Puede contener entre 0 y 8 bytes (0-64 bits).
- **Campo de aseguramiento (CRC):** cuenta con una longitud de 16 bit y es utilizado para detección de errores con los primeros 15, mientras que el último siempre es un bit recesivo 1 que delimita el campo CRC.
- **Campo de confirmación (ACK):** está compuesto por dos bits que son siempre transmitidos como recesivos '1'. Todas las unidades de mando que reciben el mismo

CRC modifican el primer bit del campo ACK por uno dominante '0', de forma que la unidad de mando que todavía está transmitiendo reconoce que al menos alguna unidad ha recibido el mensaje correctamente. De no ser así, se interpreta que su mensaje presenta un error.

- **Campo final de mensaje (EOF):** indicador del final del mensaje con 7 bits recessivos.

Trama remota: utilizada cuando un nodo requiere datos desde otro nodo. El primer nodo envía una trama remota para pedir el envío de algún dato concreto del nodo receptor. En dicha trama se realiza la petición de transmisión remota, RTR recessivo, y no contienen campo de datos.

Trama de error: es una trama especial que se envía cuando un nodo detecta un mensaje erróneo que no cumple las reglas de formato de las tramas. El error provoca que los demás nodos transmitan una trama de error. Esta propagación de mensajes de error puede llegar a producir una saturación en la red. El estándar CAN tiene estipulado un sistema de conteo de errores que gestiona el controlador y asegura que un nodo no bloquee el bus con este tipo de tramas.

Trama de sobrecarga: es enviada cuando un nodo se encuentra saturado y es necesario un retardo adicional entre tramas.

Capa de aplicación

Esta capa engloba los programas del usuario y se encuentra definida en diferentes estándares, los cuales están relacionados con sus campos de aplicación. El fin de esta capa es la de definir como se van a interpretar los campos de las capas inferiores y que se han mencionado anteriormente. Cabe mencionar que entre los estándares de las capas de aplicación más utilizadas se encuentran CANopen, CANopen FD o DeviceNet. Este último protocolo fue desarrollado con el objetivo de facilitar la automatización de fábricas y está estandarizado en el IEC 62026-3 [40].

En la próxima sección vamos a profundizar en el protocolo CANopen, que es el utilizado en este proyecto.

3.2. Protocolo CANopen

CANopen es un protocolo de alto nivel que se utiliza en automatización y tiene como objetivo el de definir la capa de aplicación. Comenzó su desarrollo dentro del paraguas del programa ESPRIT³ bajo la presidencia de la compañía Bosch [41]. Partiendo de este

³ESPRIT (European Strategic Programme for Research in Information Technology) fue un programa de investigación y desarrollo tecnológico en el campo de las tecnologías de la información de la Comunidad Europea

plan europeo nació el proyecto ASPIC⁴ donde se desarrolló una capa de aplicación que fuera sencilla de implementar, tanto en los nuevos sistemas como en los existentes, y que estuviera dedicada al control integrado de máquinas. [42]

A partir de 1994 se realizaron las publicaciones de las distintas versiones de la especificación de CANopen en el CiA301. En esta especificación se define el dispositivo CANopen básico y los perfiles de comunicación sobre los que trabajará. La definición para equipos más especializados se desarrollan sobre este perfil básico y se especifican en otros estándares publicados por CAN in Automation, como es el caso del CiA402 que fue creada para el control de movimiento de dispositivos. [30]

Todos los dispositivos que se han desarrollado bajo este protocolo deben contar con un diccionario de objetos, que es utilizado para la configuración y comunicación entre nodos. Cada entrada al diccionario esta definida por una serie de parámetros que comentaremos a continuación: [43]

- **Índice:** campo que contiene la dirección del objeto en el diccionario y que está compuesto por 16 bits.
- **Subíndice:** campo extendido de 8 bits que hace referencia a uno de los elementos de la estructura de datos del índice anterior.
- **Tipo o Tamaño del objeto:** hace referencia al tipo del objeto de entrada, ya sea una matriz, un registro o una variable simple. En la figura 3.7 viene referido como 'Object code'.
- **Nombre:** cadena que describe la entrada.
- **Tipo:** determina el tipo de dato de la variable que contiene el objeto. Los tipos de datos básicos utilizados pueden ser booleanos, enteros, flotantes, entre otros. Correspondiente al 'Data type' de la figura 3.7.

Index	2075 _h
Name	Position triggers
Object code	ARRAY
Data type	INTEGER32
Sub-index	00 _h
Description	Number of sub-indexes
Access	RO
PDO mapping	No
Default value	4

Fig. 3.7. Ejemplo de representación de parámetros del diccionario de objetos [44].

⁴ASPIC es el acrónimo Automation and Control Systems for Production Units using an Installation Bus Concept cuyo objetivo es el desarrollo de nuevas arquitecturas y componentes de control para permitir la combinación modular y flexible de componentes de las celdas de fabricación existentes

- **Atributo:** en este campo se le da al usuario la información sobre el derecho de acceso con el que cuenta el objeto, ya sea de lectura/escritura, de solo lectura o solo escritura. Mencionado como 'Access' en la figura 3.7.

Al realizar las correspondientes llamadas a los distintos objetos del diccionario es necesario que el usuario conozca la información de estos. Debido a lo cual esta información se proporciona en los manuales de usuario de los fabricantes como se puede ver en el ejemplo de la figura 3.7, extraída del manual de usuario del iPOS de ©Technosoft.

3.2.1. Estructura de los mensajes

El estándar define la transmisión de los mensajes con una estructura determinada, como se muestra en la figura 3.8, con la que cuenta de 11 a 29 bits para definir el ID, un bit de petición de transmisión remota (RTR), 4 bits que indican la longitud del mensaje que se va a enviar y de 0 a 8 bytes que contienen los datos.

Estos primeros 11 bits que componen el ID están divididos a su vez en dos campos:

- Uno compuesto de 4 bits que hace referencia al código de función de modo que se da prioridad a las funciones críticas.
- Un segundo campo formado por 7 bits que hacen referencia al identificador de los nodos de la red. Al estar el identificador limitado a 7 bits solo se permitirá un máximo de 127 elementos en una red CANopen.

Debido a que este ID se logra extender hasta los 29 bits se puede determinar el arbitraje. Esta extensión se la denomina COB-ID y todos deben ser únicos para evitar conflictos en el bus.

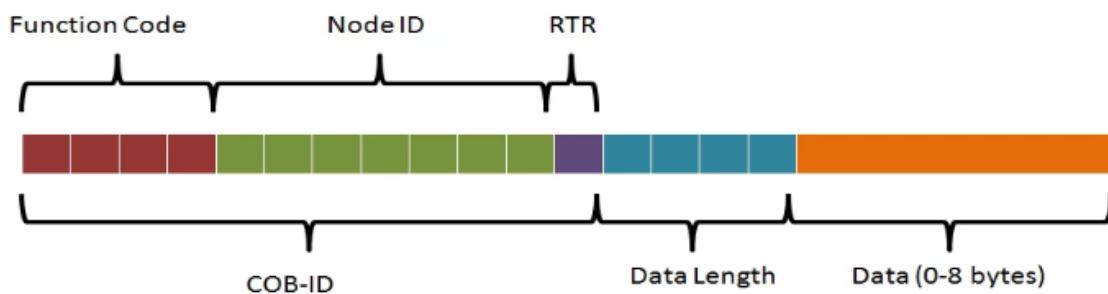


Fig. 3.8. Formato de trama CANopen.

Para formar el COB-ID se admite una asignación predefinida de identificadores en el que a cada objeto de comunicación le corresponde un rango de valores, como se muestra en la tabla 3.2.

Objeto de comunicación	Carácter	COB-ID hexadecimal	Rango de COB-ID
Control de nodos NMT	Recibir	0x000	-
Sincronizar	-	0x080	-
Emergencia	Recibir	0x80+ID de nodo	0x81 - 0x0FF
Time Stamp	0x100	Recibir	-
PDO1	Transmitir	0x180+ID de nodo	0x181 - 0x1FF
PDO1	Recibir	0x200+ID de nodo	0x201 - 0x27F
PDO2	Transmitir	0x280+ID de nodo	0x281 - 0x2FF
PDO2	Recibir	0x300+ID de nodo	0x301 - 0x37F
PDO3	Transmitir	0x380+ID de nodo	0x381 - 0x3FF
PDO3	Recibir	0x400+ID de nodo	0x401 - 0x47F
PDO4	Transmitir	0x480+ID de nodo	0x481 - 0x4FF
PDO4	Recibir	0x500+ID de nodo	0x501 - 0x57F
SDO	Transmitir	0x580+ID de nodo	0x581 - 0x5FF
SDO	Recibir	0x600+ID de nodo	0x601 - 0x67F
Control de errores NMT	Transmitir	0x700+ID de nodo	0x701 - 0x77F

TABLA 3.2. ASIGNACIÓN PREDEFINIDA DE IDENTIFICADORES
DE MENSAJE.

3.2.2. Modelos de comunicación

Para la transmisión entre los nodos se utilizan diferentes tipos de modelos de comunicación:

- **Maestro-Eslavo:** un nodo es designado como el maestro del bus, el cual es el elemento activo de la comunicación. Su función es la de enviar y solicitar datos a los esclavos de la red. Los objetos de comunicación del tipo NMT serían un buen ejemplo de este modelo de comunicación.
- **Cliente-Servidor:** el servidor da soporte a aquellos elementos que ejercen de clientes. El cliente envía información (índice y subíndice del diccionario) al servidor y este responde con uno o varios paquetes con la información solicitada. Los mensajes SDO utilizan este modelo para operar.
- **Productor-Consumidor:** este modelo consta de dos modos que tratan la transmisión de las tramas de forma distinta y que son empleados en los protocolos Node Guarding y Heartbeat.
 - **Modo push:** el productor envía información al consumidor, aunque esta no haya sido solicitada.
 - **Modo pull:** el consumidor debe solicitar información al servidor para que este transmita la información requerida.

3.2.3. Tipos de mensajes u objetos de comunicación

En la tabla 3.2 se han representado una serie de objetos de comunicación que no se habían mencionado previamente y hacen referencia a la clase de mensajes que se pueden transmitir. CANopen tiene establecidos 4 tipos de objetos de comunicación que tienen una prioridad diferente. En esta sección se van a describir estos mensajes:

Network Management Protocol (NMT): utilizado para emitir comandos de cambio de máquina de estado, detectar arranques de dispositivos remotos y condiciones de error.

El COB-ID de este protocolo siempre es 0, con lo que todos los nodos de la red procesarán los mensajes que reciban de este registro. El ID del nodo al que va dirigido el mensaje será enviado en el segundo byte del campo de datos del mensaje. Si no se especifica ningún ID, es decir se envía un 0, todos los dispositivos del bus deberán ir al estado indicado.

El código de comandos NMT que se puede enviar es reducido, ya que solo cuenta con cinco órdenes: ir a 'operativo' (0x01), ir a 'detenido' (0x02) ir a 'pre-operacional' (0x80), ir a 'restablecer nodo' (0x81) e ir a 'restablecer comunicación' (0x82). Estas se manejarán dentro de una máquina de estados como la que se representa en la figura 3.9.

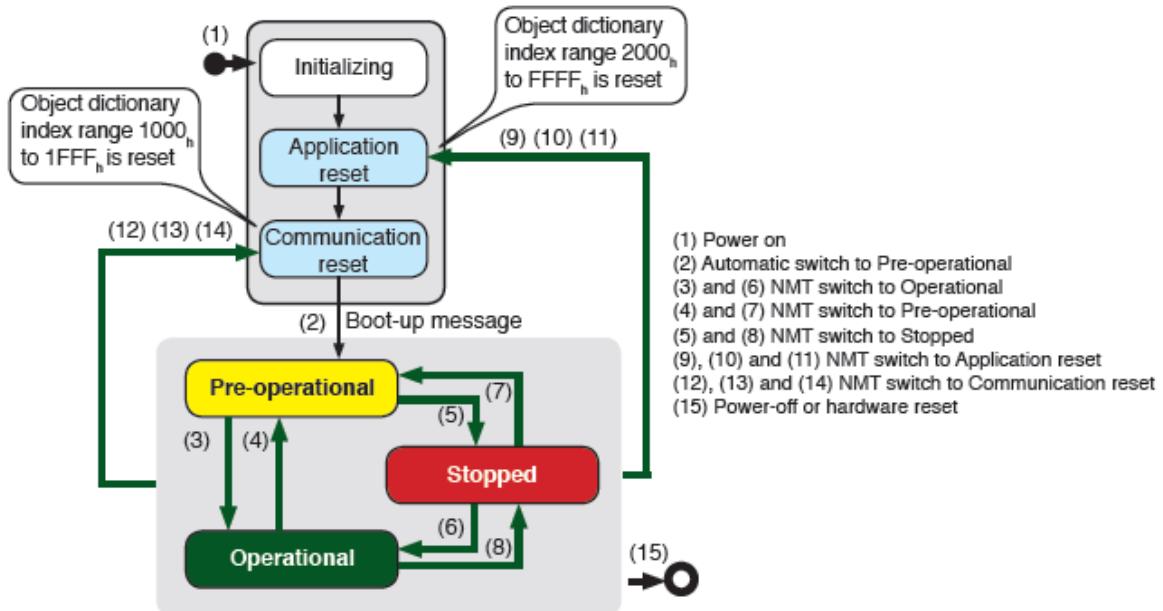


Fig. 3.9. Máquina de estados de un nodo [45].

Mensajes de sincronización: proporciona la señal de sincronización de los elementos de la red. Una vez que los nodos hayan recibido la señal, comienzan a realizar sus tareas síncronas. Este tipo de mensajes está orientado a fijar el tiempo de transmisión de mensajes PDO junto con la garantía de que los dispositivos, tanto sensores como actuadores, actúen de forma coordinada.

Mensajes de emergencia: estos mensajes se activan cuando se produce un error fatal de alguno de los nodos y se envía la información del fallo a los dispositivos de mayor prioridad del bus. Una trama de emergencia solo puede enviarse una vez por cada evento de error. Por medio de los códigos de error de emergencia definidos en el protocolo se puede acotar la causa del error del esclavo afectado.

Mensaje Time Stamp: representa la cantidad de tiempo transcurrida en milisegundos desde el 1 de enero de 1984. Algunas aplicaciones en las que el tiempo es crítico requieren de una sincronización muy precisa y por medio de este tipo de mensaje se realiza la coordinación de los nodos.

Process Data Object (PDO): utilizados para procesar datos en tiempo real entre varios nodos donde se pueden transmitir hasta 8 bytes de datos y utilizan los objetos del diccionario del protocolo. Al ser utilizados en comunicación de alta prioridad para un control en tiempo real no se permite la fragmentación de los mensajes, con lo que toda la información que deba transmitirse ha de hacerse en una única transmisión.

Sigue el modelo de comunicación de Productor-Consumidor por el que existen dos tipos de PDO que separan los datos de transmisión y de recepción. Los datos de transmisión, denominados TPDO, son utilizado para los datos que provienen del dispositivo (el elemento es un productor de datos). Los RPDO hacen referencia a los datos de recibidos por el dispositivo (el nodo es un consumidor de datos).

Además, los mensajes cuentan con la posibilidad de ser enviados con dos métodos de transmisión:

- **Síncrona:** los PDO son enviados después del mensaje de sincronización.
- **Asíncrona:** la inicialización del envío del mensaje se realiza con un evento, con un intervalo de tiempo fijo o tras la solicitud por parte de otro dispositivo de una trama RTR.

Service Data Object (SDO): se utiliza para configurar y leer valores del diccionario de objetos de un dispositivo remoto. A continuación, se va a indicar el significado de los grupos de bits que conforman la trama, así como se muestra en la figura 3.10.

- **CCS:** el Client Command Specifier utiliza tres bits para indicar el tipo de mensaje transferido (0- lectura; 1- iniciar lectura; 2- iniciar escritura; 3- escritura; 4- parar transferencia; 5- bloquear lectura; 6- bloquear escritura;).
- **n:** es el número de byte que no contienen datos y solo es válido si se indican los bits *e* y *s*.
- **e:** cuando el bit se encuentra activo indica que los datos transferidos se encuentran en contenidos en el mensaje. En el caso de tener un valor cero, los datos no caben en el mensaje y se utilizarán múltiples mensajes para la transmisión.

- **s:** hace referencia al tamaño del mensaje. Cuando el bit está en nivel alto indica que el tamaño se encuentra especificado en el campo *n*. Por el contrario, la información del tamaño vendrá contenida en la parte de datos.
- **Índice:** es el índice del diccionario de objetos de los datos a los que se acceden.
- **Subíndice:** es el subíndice de la variable del diccionario de objetos.
- **Dato:** contiene los datos que serán escritos en el mensaje.

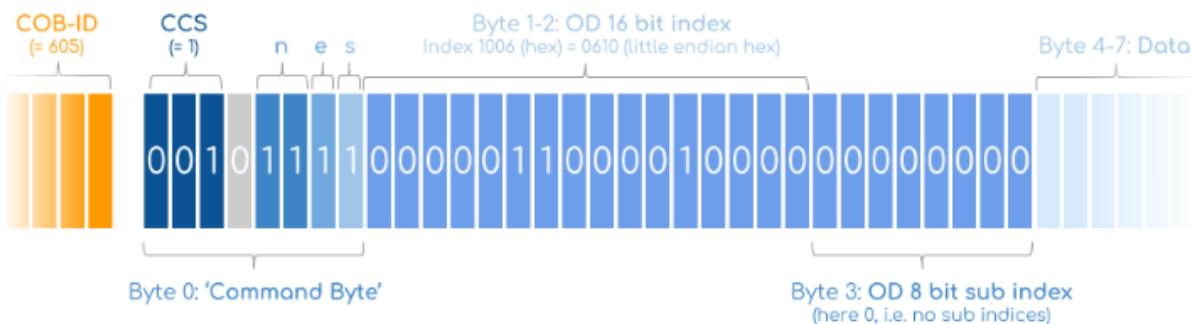


Fig. 3.10. Estructura para el protocolo SDO [46].

Mensajes Node Guarding y Heartbeat: mensajes supervisados por el dispositivo maestro y que le permiten conocer si un nodo ha dejado de estar operativo. Existen dos formas de implementar la evaluación de la operatividad de los nodos.

- **Node Guarding:** el maestro envía mensajes con el COB-ID 0x700+Id de nodo solicitando respuesta por medio de la trama RTR. Los nodos responden en el mismo rango COB-ID seguido de un bit de vida *toggle bit*, que se irá alternando tras cada solicitud, e indicando el estado operativo. Si la recepción del mensaje no es la correcta o si no se recibe, el maestro asume que el nodo ha dejado de funcionar.
- **Heartbeat:** los nodos de la red deben enviar mensajes periódicos y si alguno tarda en llegar el maestro puede tomar acción sobre él.

4. DESARROLLO DE LA LIBRERÍA

En el presente capítulo se explicarán los pasos que se realizaron durante el desarrollo de la librería que comunica los actuadores del cuello con el software de MATLAB. Para ello seguiremos el orden en el que se produjo la investigación y el desarrollo de la librería:

- Se tomará como punto de partida en esta sección la información más relevante de la plataforma de MATLAB y que es utilizado en el proyecto.
- Se mencionará la idea inicial de cómo se desarrolló la librería. Además, de indicar el punto de inflexión que llevó a un cambio de paradigma en el desarrollo de la librería. Así como se realizará la explicación de la solución adoptada.
- Finalmente se realizará la presentación de la librería de comunicaciones definitiva para este proyecto.

En la figura 4.1 se muestra el esquema de conexiones realizado entre los distintos equipos que intervienen en el funcionamiento del cuello. En él se muestran los elementos motores, los drivers y la conexión por medio del adaptador de PEAKSystem con el ordenador que aloja la plataforma de MATLAB.

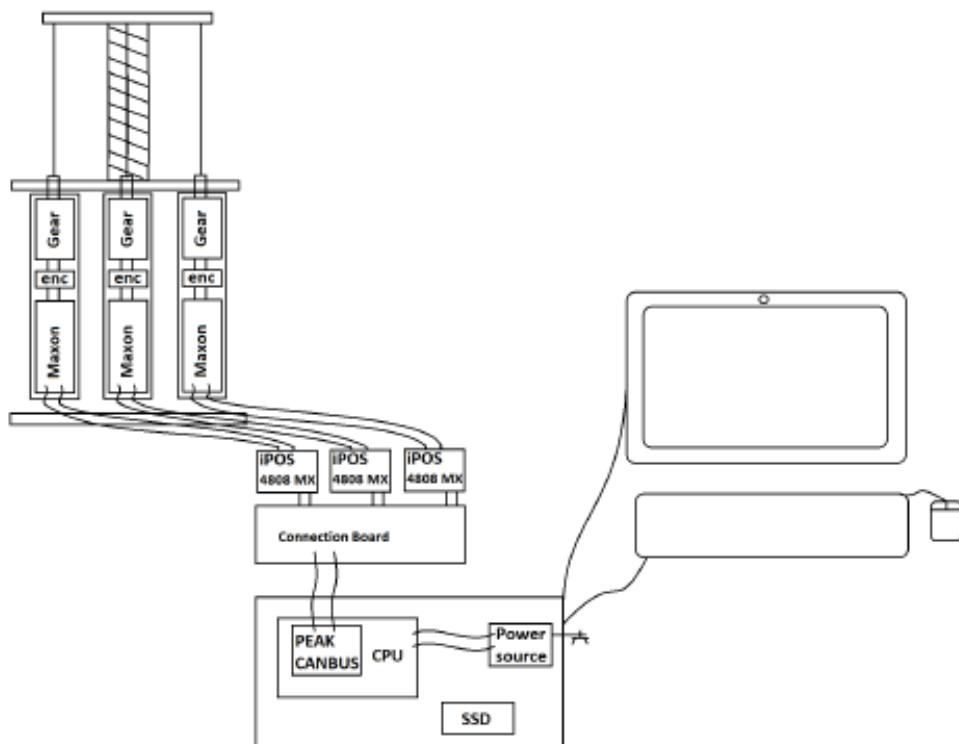


Fig. 4.1. Esquema de conexionado de los elementos [9].

4.1. Plataforma de MATLAB y su relación con el protocolo CANopen

El protocolo de comunicaciones, propiamente dicho, fue explicado a lo largo del capítulo 3. Por lo que, a continuación, esta sección se centrará en exponer cómo la plataforma MATLAB hace uso de sus funciones nativas y como son utilizadas para comunicar, por medio del protocolo CAN, con elementos externos.

Estas funciones serán utilizadas como la base sobre las que se construirá posteriormente la librería y están incluidas dentro de Vehicle Network Toolbox. La cual proporciona las funciones necesarias para enviar y recibir mensajes CAN.

Por otro lado, como para la comunicación entre el pc y los drivers se va a utilizar el adaptador de PEAK-System se debe tener en cuenta las compatibilidades de los paquetes de MATLAB con el sistema operativo empleado. Como se puede observar en la figura 4.2, el apoyo para hardware que facilita MATLAB para esta herramienta, únicamente se encuentra disponible para sistemas operativos basados en Windows.

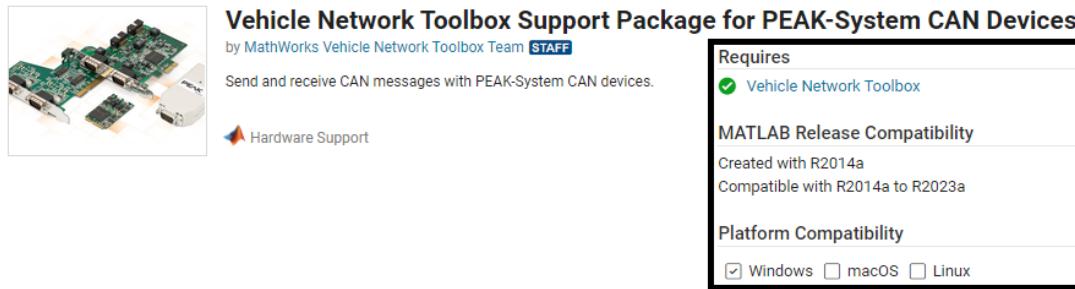


Fig. 4.2. Compatibilidad de la herramienta de PEAK-System en MATLAB.

Durante el desarrollo del proyecto se han utilizado cuatro de estas funciones básicas para las cuales se pueden dividir en dos familias: una de ellas dedicada a la configuración y otra dedicada a la gestión.

Las funciones que permiten la configuración de los parámetros de comunicación y la especificación de los datos del mensaje, tienen como objetivo la creación de objetos de tipo **canChannel** y **canMessage**.

El primer tipo de función se utiliza para crear un objeto de canal de comunicación CAN que responderá a los parámetros requeridos por el usuario. Para ello, y como se muestra en la web de ayuda de Matlab [47], se puede crear un canal de comunicaciones para cada uno de los proveedores que estén conectados al ordenador o generar un canal virtual. En este proyecto, al utilizar específicamente equipo de uno de estos proveedores, se ha utilizado la configuración requerida para el adaptador de Peak-System, que como se muestra en el HelpCenter [47] cuenta con su propia configuración. Cabe señalar que MATLAB facilita una función específica que da información a cerca de los equipos disponibles en los puertos de comunicaciones, **canChannelList**. En la figura 4.3 se muestra un ejemplo de cómo se presenta la información, con lo cual tras la consulta de esta in-

formación se asegura la correcta configuración del puerto a través del cual se realizará la comunicación.

Vendor	Device	Channel	DeviceModel	ProtocolMode	SerialNumber
"MathWorks"	"Virtual 1"	1	"Virtual"	"CAN, CAN FD"	"0"
"MathWorks"	"Virtual 1"	2	"Virtual"	"CAN, CAN FD"	"0"
"Vector"	"Virtual 1"	1	"Virtual"	"CAN"	"0"
"PEAK-System"	"PCAN_USBBUS1"	1	"PCAN-USB"	"CAN"	"0"

Fig. 4.3. Información facilitada tras realizar la llamada a la función 'canChannelList' [48].

Una vez creado el objeto del canal de comunicaciones se deberá ajustar la velocidad del bus de comunicaciones al valor que requiera la aplicación. Para realizar este cambio, MATLAB proporciona la función **configBusSpeed** en la que se tendrá que indicar el objeto **canChannel**, el cual se quiera modificar, así como el nuevo parámetro de velocidad [49]. Cuando se haya configurado todos los parámetros necesarios se podrá utilizar la función **start** para iniciar la transmisión y recepción a través del canal de comunicaciones. Si se desea detener el envío y recepción de los mensajes se utilizará la función **stop** y de esa forma se libera el uso de los recursos de hardware que se estaban utilizando por parte del canal de comunicaciones.

Si se realiza una llamada al objeto creado desde la plataforma de MATLAB, este le dará al usuario información de cierta utilidad como pueden ser: el estado de la comunicación, la cantidad de mensajes disponibles en el buffer, la cantidad de mensajes que han sido recibidos, la cantidad de mensajes transmitidos por el canal, los datos del proveedor del canal de comunicaciones o la velocidad de transmisión. Todos estos parámetros informativos y otros pueden ser consultados en la web de ayuda [47].

En referente a la función **canMessage**, esta crea un objeto que contendrá la información que compone un mensaje CAN, como la ID del mensaje, la longitud de los datos y los datos en sí. Este objeto será posteriormente enviado por las funciones encargadas de la gestión de la transmisión de mensajes. Los mensajes generados por la librería utilizan el primer formato sugerido por el HelpCenter de MathWorks [50], en la que se indicará el COB-ID del mensaje, que este es de tipo estándar de 11 bits (false) y la longitud de datos esperada, que será una de las indicadas en la tabla 4.1.

Tamaño (bits)	0	8	16	32
Valor	40h	2Fh	2Bh	23h

TABLA 4.1. TAMAÑO DEL MENSAJE.

Una vez creado el objeto del mensaje, este contará con un atributo en el que se almacenará el mensaje que se desea transmitir. El formato del campo de 'Data' [51], en el

cual se guardan los datos del mensaje, es una matriz fila de valores unit8 compuesta por tantas columnas de datos como se haya indicado en el parámetro de la longitud de datos esperada en el constructor. En la tabla 4.2, se observa la distribución de los datos dentro de los mensajes.

Tamaño	Índice	Subíndice	Datos
1 byte	2 byte	1 byte	4 bytes

TABLA 4.2. ESTRUCTURA DE LOS DATOS DENTRO DE LOS MENSAJES CANOPEN.

Cabe aclarar que MATLAB trata todos los datos numéricos de entrada, tanto en los constructores de los objetos como en los datos de los mensajes, como valores decimales. Este hecho entra en contradicción con el hecho de que la mayor parte de la documentación referente al protocolo CAN utiliza el formato hexadecimal como base numérica para las direcciones PDO, SDO y registros (como podría ser la dirección de la WordControl). Para solventar este problema, en la librería del proyecto se contará con un listado de los registros utilizados en el proyecto en formato hexadecimal y las funciones encargadas del envío de información se encargarán de cambiar el formato de hexadecimal a decimal.

Por otro lado, las funciones que pertenecen a la familia que se decida a la gestión son **transmit** y **receive**. Estas permiten el envío y recepción de la información a través del canal de comunicaciones CAN.

La función **transmit** envía un mensaje a través del canal de comunicación y este será recibido por los nodos conectados en la red para ser atendido por el elemento destinatario. Como parámetros de entrada a la función el usuario debe especificar el objeto `canChannel`, por qué canal se va a transmitir, y el objeto `canMessage`, el mensaje que se enviará a un receptor concreto, ambos deberán haber sido creados con anterioridad al envío [52].

En cuanto a la recepción de mensajes por medio de la función **receive**, este será la encargada de recibir los mensajes que se encuentran almacenados en el buffer de recepción del objeto `canChannel` en cuestión. Para ello se deberá indicar el objeto `canChannel` y el número de mensajes que se desean recibir. En el caso de especificar que se desea recibir '`inf`', la función recogerá todos los mensajes disponibles [53]. Como resultado del uso de esta función se obtiene una matriz con tantas filas como mensajes estén disponibles. Cada columna de esta matriz se corresponderá con un byte de información recibida en la que, por orden, se podrá llegar a tener la siguiente información: COB-ID (1 byte), tamaño del mensaje (1 byte), índice del mensaje (2 bytes), subíndice (1 byte) y los datos del mensajes (hasta 4 bytes).

4.2. Primeros desarrollos de la librería

En primera estancia se propuso la realización de la librería como un conjunto de funciones independientes que realizarán una acción concreta para la cual fueron diseñadas. Esto haría que el usuario utilizará la función concreta que necesitará en cada momento. Tras recopilar la información necesaria de cómo MATLAB gestiona las comunicaciones a través del protocolo CANopen, la cual hemos explorado en la sección anterior, le tocaba el turno a la recopilación de la información de los drivers de control de cada uno de los motores del cuello.

Los drivers utilizados para la gestión de los movimientos del cuello son los driver inteligente iPOS4804 MX-CAN. Toda la información necesaria para la manipulación de los drivers por medio de los canales de comunicación se recoge en el manual de usuario que facilita el fabricante [44].

Para realizar el control del driver se debe tener en cuenta que en este se puede encontrar una serie de posibles estados de operación durante su periodo de funcionamiento. Estos estados se ven regidos por una máquina de estado, como la que se observa en la figura 4.4, en la que cada evento genera una acción que provoca una transición en la máquina de estado. Los estados se ven reflejados en el objeto **StatusWord** (correspondiente con el objeto 6041h) del iPOS y se pueden modificar por medio de la **ControlWord** (correspondiente con el objeto 6040h) del dispositivo. Esta información se puede encontrar en la documentación del manual del fabricante en la sección 5.1 [44].

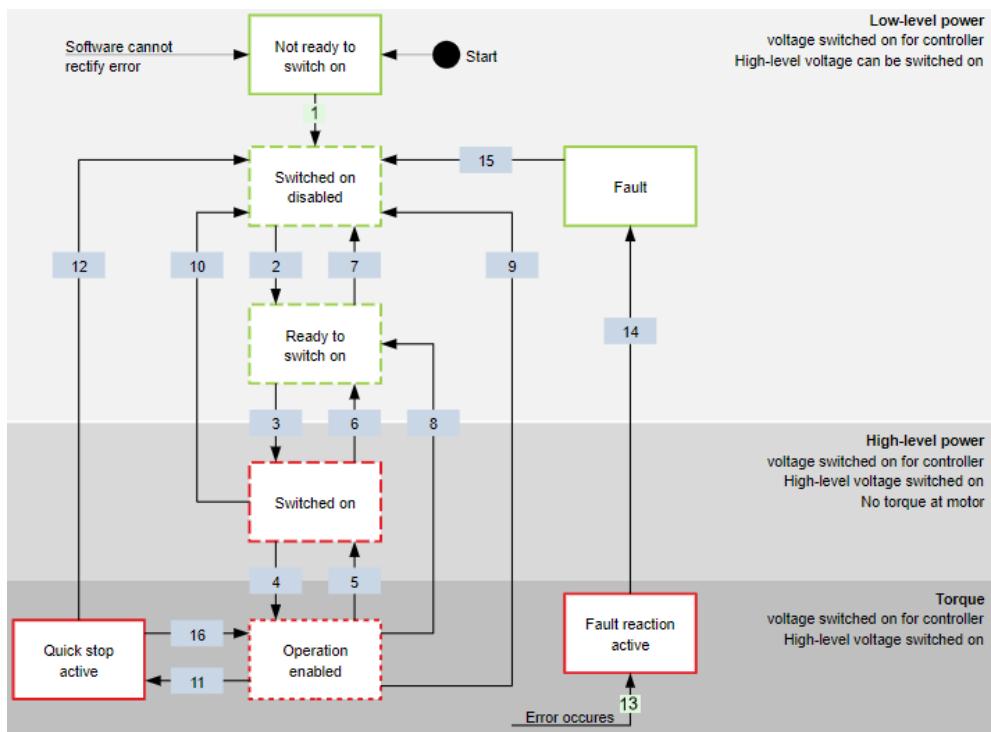


Fig. 4.4. Máquina de estados del driver dentro de CiA402 [54].

La StatusWord del driver representará la situación en la que se encuentre la máquina de estados a través de los bits que la conforman. En la figura 4.5 se muestran estos bits y su significado. De todas formas los estados del driver se pueden identificar cuando la codificación de la palabra baja de StatusWord es la siguiente:

- **Not Ready to switch on** [xxxx xxxx x0xx 0000b]: el driver ha realizado el arranque tras el encendido pero aun no está habilitado para funcionar.
- **Switch on disabled** [xxxx xxxx x1xx 0000b]: las inicializaciones básicas han sido realizadas. El led con el que cuenta el driver debe estar encendido de color verde para indicar que no se ha detectado ningún fallo. El motor no está listo para realizar movimientos, pero sí se pueden modificar parámetros del driver y se puede conectar la alimentación del motor.
- **Ready to switch on** [xxxx xxxx x01x 0001b]: puede haber presencia de alimentación en el motor y la mayoría de los ajustes de parámetros se pueden modificar. Aun no se pueden realizar movimientos.
- **Switched on** [xxxx xxxx x01x 0011b]: debe haber presencia de tensión en el motor y la etapa de potencia está habilitada. El motor se mantiene a la espera hasta que esté disponible para operar. En esta etapa todavía no se pueden realizar movimientos.
- **Operation enabled** [xxxx xxxx x01x 0111b]: desde este estado el motor ya puede realizar movimientos. Para ello, realizará el control de los movimientos necesarios para alcanzar las consignas indicadas según su perfil o modo de funcionamiento. Para que esta etapa se lleve a cabo no debe haber presencia de fallo, el motor tiene que estar alimentado y el usuario ha debido de habilitar las funciones de movimiento.
- **Quick stop active** [xxxx xxxx x00x 0111b]: el motor ha sido detenido por medio de una orden de parada rápida.
- **Fault reaction active** [xxxx xxxx x0xx 1111b]: tras la detección de un fallo el driver detiene el motor.
- **Fault** [xxxx xxxx x0xx 1000b]: la alimentación del motor se detiene y el driver permanece en condición de fallo hasta que se reinicie el registro de errores por medio de un ResetFault.

Bit	Value	Description
15	0	Axis off. Power stage is disabled. Motor control is not performed
	1	Axis on. Power stage is enabled. Motor control is performed
14	0	No event set or the programmed event has not occurred yet
	1	Last event set has occurred
13..12		Operation Mode Specific. The meaning of these bits is detailed further in this manual for each operation mode
11		Internal Limit Active – see Remark 1 below
10		Target reached
9	0	Remote – drive is in local mode and will not execute the command message.
	1	Remote – drive parameters may be modified via CAN and the drive will execute the command message.
8	0	No TML function or homing is executed. The execution of the last called TML function or homing is completed.
	1	A TML function or homing is executed. Until the function or homing execution ends or is aborted, no other TML function / homing may be called
7	0	No Warning
	1	Warning. A TML function / homing was called, while another TML function / homing is still in execution. The last call is ignored.
6		Switch On Disabled.
5		Quick Stop. When this bit is zero, the drive is performing a quick stop
4	0	Motor supply voltage is absent
	1	Motor supply voltage is present
3		Fault. If set, a fault condition is or was present in the drive.
2		Operation Enabled
1		Switched On
0		Ready to switch on

Fig. 4.5. Tabla en la que se describe el significado de cada uno de los bits de la word de control.

Para el control de los movimientos del motor, los drivers cuentan con una serie de perfiles de funcionamiento: **Position Profile Mode**, en el que el driver controla la posición del motor, **Velocity Profile Mode**, donde el driver realiza un control de la velocidad a la que se realiza el movimiento, y el **Torque Profile Mode**, encargado de realizar un control sobre el parámetro de la aceleración con el objetivo de regular el par que se ejercerá durante el movimiento.

Para este proyecto se va a realizar el control de posición del driver o **Position Profile Mode** en la que se buscará que el motor llegue a la posición determinada por el usuario tras cada orden. El sistema realizará un movimiento de tipo trapezoidal. Este se define de esta forma al dividir su movimiento en tres etapas:

- Inicia el movimiento con una aceleración constante hasta alcanzar una velocidad determinada.
- Mantiene dicha velocidad hasta que el driver detecte que se encuentra cerca de la posición objetivo.
- Finaliza el movimiento con una deceleración constante hasta alcanzar el punto designado.

Una vez conocida esta información, se procede al desarrollo de las primeras funciones y a comprobar que la comunicación con los equipos es correcta. Para la obtención de los objetos del canal de comunicación, mensajes y control, se optó por la realización de funciones independientes, como se indicó en el primer párrafo de esta sección, en las

que el usuario debería indicar todos los parámetros necesarios para la realización de cada mensaje.

Se creaba un objeto propio del canal de comunicación, denominado **canch**, que debería ser especificado en cada transmisión. Para poner un ejemplo de las funciones utilizadas en esta primera etapa de desarrollo vamos a comentar las funciones creadas para ordenar el movimiento a una posición:

1. Creación del canal de comunicaciones, indicando la velocidad de transmisión, con la función **createChannelPEAKSystem(Velocidad del bus)** y asociándolo al parámetro **canch** que guardará el objeto creado. Tras esto se inicializa la actividad del canal de comunicaciones con la función nativa **start(canch)**. Se crea una variable que contenga el número de esclavo ID del driver sobre el que se va a trabajar.
2. Se realiza la llamada a la función **startMotor(id,canch)**. Esta función contiene una sucesión de acciones que activan el motor. Todas las transmisiones de información se realizarán por medio de la función **PutMsg(id,COBID,data,canch)**, la cual crea los mensajes y realiza la transmisión con las funciones nativas **canMessage** y **transmit** a las que hicimos referencia en la sección 4.1
 - Configuración de un nodo de la línea para que reciba órdenes de forma remota. Los datos de entrada en la función PutMsg serán: '0' para el id y el COBID al tratarse de un mensaje que se envía de forma general al bus, como data se enviará una matriz con la información del id y la acción a realizar [1 id] y como canal de comunicaciones el parámetro canch.
 - Se envía la orden para que pase al estado **Ready to switch on**. En la función PutMsg se indicará: el id del nodo, la dirección 200 referente a las PDO1, el dato del mensajes será la matriz [6 0] que representa la orden del cambio de estado y por último el canal de cmomunicaciones canch.
 - Se envía la orden para que pase al estado **switch on** saliendo del estado anterior 'Ready to switch on'. En la función PutMsg se vuelve a indicar el nodo, la PDO asociada, el mismo canal de comunicaciones y la matriz del cambio de estado [7 0].
 - Tras esto, se habilita el driver para poder realizar operaciones, cambiando de estado mediante la orden de **Enable operation**. PutMsg tendrá los mismos parámetros de entrada que en los casos anteriores, salvo por la matriz de datos que será [0x0F 0].
3. Una vez que el driver esta en la etapa que le habilita a realizar movimientos se especifica el modo de operación. En este caso el perfil escogido es el del modo posición. El tipo de mensaje enviado será SDO en el que se pretende acceder al objeto 6060h, al que se le enviará un mensaje de 8 bits (0x2f) que representarán la opción del modo posición con el valor '1'. Con lo que la función tendrá el siguiente aspecto: **PutMsg(id,600,[0x2F 0x60 0x60 0 1 0 0 0],canch)**

4. A continuación se definirán la consigna de posición y una configuración de velocidad inicial. Para ello existen dos funciones llamadas `setPosition(targetPosition)` y `setVelocity(targetVelocity)`, en las que el usuario indicará las consignas deseadas y estas devolverán una matriz fila, la cual estará compuesta por:

- El tamaño esperado de los datos. En ambos casos 32 bits (0x23).
- El índice del objeto al que se enviarán los mensajes con subíndice con valor '0'. 607Ah para el target de posición y 6081h para la configuración de la velocidad.
- En los últimos 4 bytes del mensaje se guardará la información de la consigna introducida, transformada para que se adapte al formato de 4 bytes.

5. Por último se ordenará el inicio del movimiento por medio de un mensaje de tipo PDO y una matriz de datos con la orden que inicia el movimiento.

Durante las fases tempranas de desarrollo, se concluyó que esta solución era demasiado confusa para que un usuario pudiera trabajar con ella. El hecho de que para cada mensaje el usuario tuviera que indicar un gran número de parámetros de entrada, tales como: el canal de comunicaciones, el ID del nodo de comunicación, el tipo de mensaje que se quisiera comunicar (NMT, SDO o PDO) y el mensaje de envío, requería por parte de la persona que utilizara la aplicación estar pendiente de demasiados parámetros en cada transmisión. Esto es un inconveniente y por tanto era necesario cambiar la perspectiva y el trato de los datos.

Para solventar este problema se optó por la incorporación de variables tipo **global**. Si en MATLAB se declara una variable u objeto como global, se permite a todas las funciones del proyecto acceder a dicho parámetro. Para que estas variables puedan ser usadas en las funciones en las que no están declaradas, se debe indicar al inicio de estas funciones cuales serán las variables globales utilizadas en la función en cuestión.

Como variables globales a las que pudiesen añadir se indicaron: el objeto del canal de comunicaciones, el ID de los equipos, los tipos de mensajes de comunicación (NMT, SDO o PDO) y los objetos de registros a los cuales el usuario pudiera acceder, los cuales vienen indicados en el manual (Controlword, Statusword, Target position, Position actual value, Switch On, entre otros). Tras añadir esta modificación al programa se produjo una reducción en el uso de parámetros de entrada en las funciones.

Las funciones se especializaron y cada llamada a una función encerraba en sí el canal de comunicaciones, el tipo de mensaje específico y el ID del equipo al que se quería manipular. El usuario solo tendría que indicar la consigna que quisiera transmitir al driver.

Aun así, esta solución no era la más adecuada, como principal inconveniente se encuentra que al poder consultar el parámetro ID dentro de cada función, el usuario estaría obligado a modificar esta variable global cuando quisiera hacer referencia a cualquier otro

de los drivers de los cuellos. Además, de que esta forma de programar las funciones obligaría a tener que indicar todas y cada una de las variables globales en las futuras funciones creadas, conllevando posibles olvidos de variables globales con su consiguiente fallo de funcionamiento.

Era necesario encontrar una forma en la que solventar los problemas que persistían a la hora del manejo de las funciones. Se encontró una posible alternativa que daría solución a todos los inconvenientes encontrados. Esta solución sería la de aplicar la programación orientada a objetos (POO) en MATLAB. Como resultado se produjo el cambio del paradigma que se había fijado en un primer momento, pasando de funciones independientes con poca interrelación a la construcción de una aplicación en la que se genera interdependencia entre los distintos métodos.

No obstante, esta nueva visión trajo consigo la posibilidad de escalar el programa, en el cual se podrían crear objetos sencillos para el control individual de los motores y que posteriormente estos compongan objetos mayores que sirvieran para controlar el cuello.

4.3. Plataforma de MATLAB y la programación orientada a objetos

La plataforma de MATLAB soporta la programación orientada a objetos (POO), habilitando a los usuarios a definir y manipular objetos en sus programas. La POO permite a los programadores crear programas más eficientes, modulares, reutilizables y escalables, lo que puede reducir el tiempo y el costo de desarrollo. Además, puede mejorar la comprensión y la mantenibilidad del código.

MATLAB ofrece un conjunto de herramientas para crear clases, con sus respectivos métodos y propiedades. Las clases son plantillas para crear objetos, y las propiedades son variables que describen el estado de un objeto. Los métodos son funciones que definen el comportamiento de un objeto y pueden ser llamados por otros objetos o por el programa principal.

Para definir una clase se debe utilizar la palabra clave **classdef** [55] seguida del nombre de la clase que se va a diseñar. Esta a su vez heredará de la clase **handle** [56], la cual se utiliza para crear objetos cuyas instancias pueden ser modificadas por referencia. La asignación de objetos permite modificar el objeto original a través de cualquier instancia que haga referencia a él.

Dentro de la clase se definirán las propiedades que reflejarán el estado del objeto. Para definir las propiedades en MATLAB, se debe utilizar la palabra clave **properties** [57]. Se le pueden indicar una serie de propiedades de atributos [58], a los que deberán responder todas las variables que se definan en ellos, tales como pueden ser: el tipo de acceso (público, privado o protegido), abstractos, constantes. En cada clase se pueden definir varios apartados de propiedades con atributos distintos unos de otros. Todas las variables que se creen bajo el apartado de propiedades podrán tener valores iniciales si así lo considera el usuario [59].

Los métodos son las funciones que actúan sobre el objeto. Para definir un método en MATLAB, se debe utilizar la palabra clave **methods** [60]. Es importante tener en cuenta que cuando se definen todos los métodos de una clase deben tener como primer argumento el objeto de la clase (generalmente se llama **obj**) [61], y que este objeto se pasa automáticamente como argumento cuando se llama al método. Tras esto, el programador podrá definir cuantos argumentos considere necesarios a continuación del argumento **obj**.

Cabe indicar, que a la hora de realizar la llamada a los métodos, es decir cuando se declara un objeto y se usa en otro apartado del programa, no es necesario indicar el primer argumento. El usuario comenzará indicando los argumentos a partir del segundo argumento definido en la clase.

Tras esto, ya se podrán crear objetos de las clases desarrolladas y hacer uso de sus métodos.

4.4. Librería de comunicación CANopen en MATLAB

Como propuesta de desarrollo de la librería por medio del uso de la programación orientada a objetos que permite MATLAB, se obtiene como resultado el diagrama de clases que se muestra en la figura 4.6. Este está compuesto por cinco clases y cinco funciones, con las que se satisfará las necesidades de gestión del cuello robótico por parte del usuario, bajo las indicaciones antes dadas en la sección 1.2.

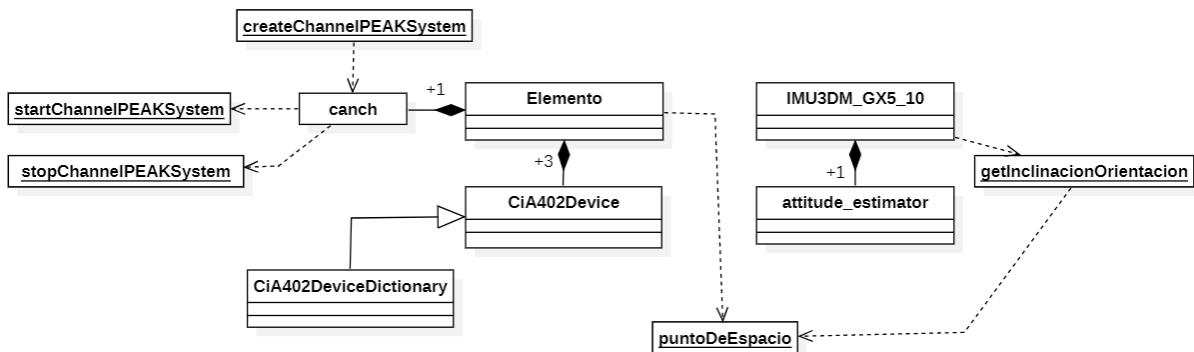


Fig. 4.6. Diagrama de clases.

Las clases se pueden dividir en dos grandes familias encargadas de un conjunto de dispositivos concretos. En primer lugar, nos encontraríamos con aquellas clases utilizadas para conformar la comunicación y control del cuello robótico:

- La clase **CiA402DeviceDictionary** alberga los registros de los objetos con los que puede comunicarse el programa.
- Para realizar la comunicación directa con el driver conectado a la red bus se hará uso de la clase **CiA402Device**, que a su vez heredará de la clase anterior.

- Por otro lado, la clase **Elemento**, la cual está compuesta por objetos de la clase **CiA402Device**, será con la que actuará el usuario y en la que podrá realizar las configuraciones que este necesite y enviar las consignas de posición.

Estas clases cuentan el soporte de las tres funciones utilizadas para la configuración del canal de comunicación: **createChannelPEAKSystem**, **startChannelPEAKSystem** y **stopChannelPEAKSystem**.

La segunda familia, se centra en la obtención de los datos del sensor y la transformación de la información recibida por este en datos que el usuario pueda interpretar:

- La clase **IMU3DM_GX5_10** cuenta con los métodos necesarios para comunicar al sensor con el software, realizando las lecturas de los acelerómetros y los giróscopos.
- Una vez obtenido la lectura del sensor se debe tratar la información recibida. En la clase **attitude_estimator** se encuentran los métodos necesarios para realizar la transformación de la información recibida en los ángulos de Euler (pitch, roll y yaw).

Además, se cuenta con la función **getInclinacionOrientacion** que toma los datos de los ángulos de Euler (en concreto pitch y roll) y los transforma en información de inclinación y orientación. De esta forma se obtiene que los datos de salida del programa cuenten con el mismo formato que las consignas de entrada introducidas por el usuario.

Posteriormente, por medio de la función **puntoDeEspacio** se utilizan los resultados de la función anterior y los parámetros constructivos extraídos del cuello (guardados en el objeto de la clase **Elemento**) para asignar un punto en el espacio tridimensional a la posición de la articulación.

En las próximas secciones de este capítulo se explicarán con más detalle las clases y funciones vistas en la figura 4.6, mencionando los aspectos más relevantes relacionados con el objetivo propuesto de obtener una librería de comunicaciones para el cuello robótico.

4.4.1. Funciones de configuración del canal de comunicación

Este tipo de funciones tienen como objetivo crear, encender y apagar el canal de comunicaciones en el protocolo CANopen para el modelo de adaptador PEAKSystem. El objeto creado será añadido al objeto de la clase **Elemento** como parámetro de referencia para realizar las transmisiones de mensajes correspondientes durante el uso de la librería.

A continuación, se indicará el marco de actuación de cada una de las funciones de este tipo.

createChannelPEAKSystem (BusSpeed): esta función crea y asigna a una variable el canal de comunicaciones, específico del adaptador PEAKSystem, que utilizará

MATLAB. Este canal será el utilizado para mantener la comunicación entre los drivers de los motores y el software de MATLAB del ordenador.

A la hora de crear el canal se debe indicar la misma velocidad de transmisión a la que está configurada en los drivers. Al ser una función específica para un adaptador en concreto, internamente, se configura el canChannel con los datos 'PEAK-System' y 'PCAN_USBBUS1' que hacen referencia al modelo y tipo de conexión del adaptador. Posteriormente se ajusta la velocidad a la indicada por el usuario con el método config-BusSpeed, propio de MATLAB.

Como resultado del uso de esta función se crea un objeto denominado **canch**, que guarda todos los datos necesarios para realizar la comunicación y que se añadirá como atributo al objeto creado de la clase **Elemento**.

startChannelPEAKSystem (canch): función utilizada para inicializar la transmisión de datos del canal de comunicaciones. Tras crear el canal de comunicaciones se deberá realizar la llamada a esta función para abrir el canal de comunicación con los nodos del sistema. El usuario debe indicar el canal que quiere inicializar, una vez activo aparecerá el mensaje 'Canal activado.'. Si ya estuviera previamente activado será notificado mediante el aviso 'Canal actualmente activado.'. En el caso de que la variable introducida no se corresponda con un canal previamente creado, se mostrará el siguiente mensaje: 'Canal no creado.'

stopChannelPEAKSystem (canch): función utilizada para detener la transmisión de datos del canal de comunicaciones. El usuario debe indicar el canal que quiere parar, una vez desactivado aparecerá el mensaje 'Canal desactivado.'. Si ya estuviera desactivado será notificado mediante el aviso 'Canal actualmente desactivado.'. En el caso de que la variable introducida no se corresponda con un canal previamente creado, se mostrará el siguiente mensaje: 'Canal no creado.'

4.4.2. Clases desarrolladas para la comunicación con el cuello robótico.

En esta sección no solo se van a presentar las clases dedicadas a la interacción directa con los driver IPOS4808 MX-CAN utilizados en el proyecto SOFIA para el control de articulaciones blandas. Además, se mostrará la clase diseñada para englobar varios de estos drivers en un único elemento.

Clase diccionario: CiA402DeviceDictionary

Esta clase fue diseñada para establecer una librería con registros de mensajes a los que se puede tener acceso por comunicaciones. Para ello, únicamente está provista de un constructor vacío llamado '**CiA402DeviceDictionary()**', en el cual se han almacenado los registros. Estos estarán disponibles en la tabla 4.3, pero el usuario no contará con un acceso directo, deberá realizarlo a través de los métodos de la clase hija.

Atributo	Valor del mensaje	Definición
targetPosition	0x7A 0x60 0	Dirección del Target de posición 607Ah 0h
targetVelocidad	0x81 0x60 0	Dirección del Target de velocidad 6081h 0h
targetAceleration	0x83 0x60 0	Dirección del Target de aceleración 6083h 0h
OperationMode	0x60 0x60 0	Dirección del Modo operación 6060h 0h
positionmode	1	Setup del modo de funcionamiento: posición 1h
controlword	0x40 0x60 0	Dirección de la word de control 6040h 0h
statusword	0x41 0x60 0	Dirección de la word de estado 6041h 0h
readySwitchOn	6 0	Orden para ordenar que el Switch este listo 0006h
switchOn	7 0	Orden de activación del Switch 00007h
goswitchondisable	0 0	Orden de apagado del Switch 0000h
enable	0x0F 0	Orden de habilitación de las operaciones 000Fh
starProfile	0x1F 0	Inicio de la acción del driver para que se sitúe en la posición indicada 001Fh
quickstop	2 0	Orden de parada de emergencia 0002h
OperationModeDisplay	0x61 0x60 0	Dirección de la petición de información de modo de operación activo 6061h 0h
getCurrentPosition	0x64 0x60 0	Dirección de la petición de información de la posición actual 6064h 0h
resetPosition	0x0F 0	Dirección del reset de posición 000Fh
getDemandPosition	0x62 0x60 0	Dirección de la petición de información de la posición de demanda 6062h 0h

TABLA 4.3. ATRIBUTOS UTILIZADOS Y SUS VALORES DE LA CLASE CIA402DEVICEDICTIONARY.

Por otro lado, está clase hereda de la clase 'handle' de MATLAB. Es necesario realizar

esta herencia para que el sistema de clases funcione y no se considerén como funciones abstractas.

Clase CiA402Device

Clase que hereda de CiA402DeviceDictionary. El objetivo de esta clase es la de proporcionar al usuario las herramientas necesarias para poder interactuar con los drivers de forma directa. Los objetos que se generen a raíz de esta condición serán los encargados en trabajar sobre la capa del enlace de datos de los drivers desde la aplicación en MATLAB.

En la figura 4.7 se muestra la interrelación entre los distintos métodos programados en esta clase, los cuales se explicarán a continuación y se mostrará una figura descriptiva en aquellos métodos que cuenten con funciones subordinadas.

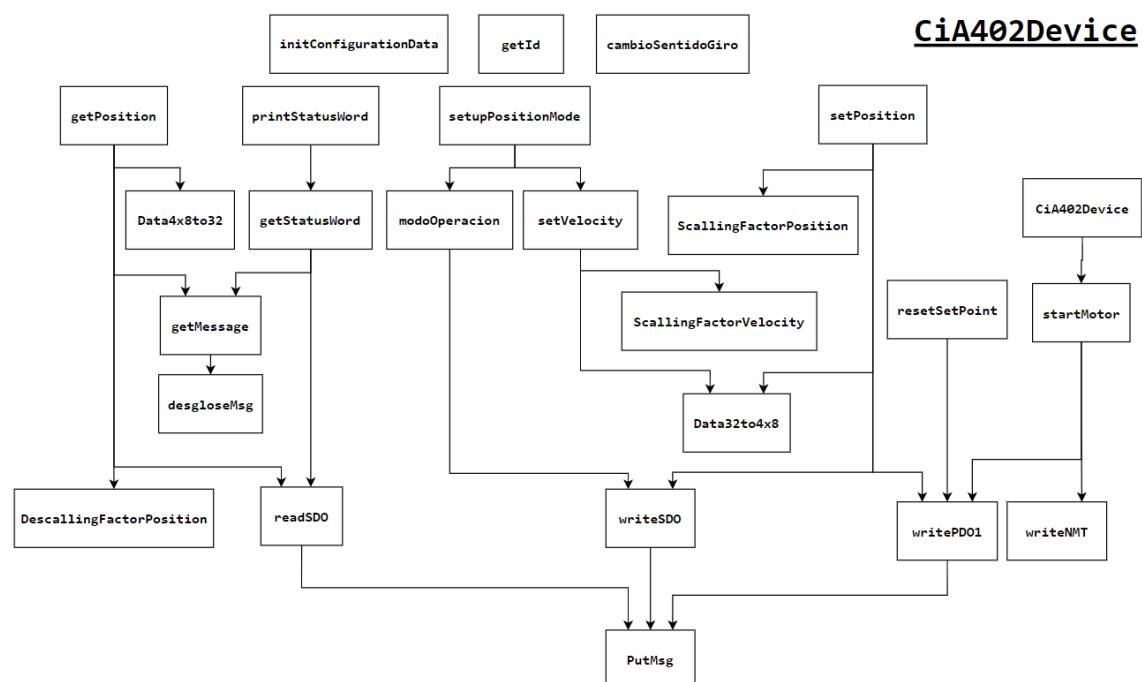


Fig. 4.7. Relación entre los métodos de la clase CiA402Device.

CiA402Device(inputId,inputCanch): constructor de la clase del driver de cada motor que componen una articulación. El usuario tendrá que indicar el Id o dirección del nodo del dispositivo y el canal por el cual se realizará la comunicación (canch). Tras esto, se realizará una llamada al método 'startMotor' para indicar al driver que será comandado de forma remota y que debe activar su switch interno.

En la figura 4.8 se muestra el orden de ejecución de los métodos de la clase **CiA402-Device** para crear el objeto que utilizará el usuario para controlar el driver del bus CAN.

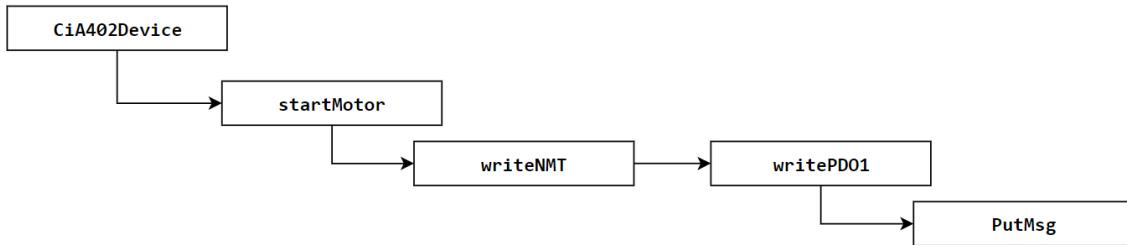


Fig. 4.8. Orden de ejecución de los métodos tras llamada al constructor de la clase.

initConfigurationData (encoder, ratio, sample, limitCurrentMotor, limitCurrentDriver): todas las operaciones de escalo/desescalado de las consignas enviadas a los motores deben ser tratadas previamente.

Es necesario especificar una serie de parámetros previamente. Por lo que el usuario podrá utilizar este método para definir los valores de cada uno de los atributos: resolución del encoder (encoder), ratio de desplazamiento (ratio), periodo de muestreo (sample), límite de corriente del motor (limitCurrentMotor) y límite de corriente del driver (limitCurrentDriver). Todos los atributos deben ser no nulos cuando se realiza la llamada al método.

cambioSentidoGiro(): método muy sencillo que modifica el atributo de sentido de giro.

writeSDO (direccion, comando): escribe un comando de distintas longitudes en una dirección específica del dispositivo. Para ello se utilizan como parámetros de entrada la dirección del objeto de dirección y el comando con la orden requerida.

En este método se ejecutan distintos pasos antes de poder enviar el mensaje:

- En primer lugar, se debe comprobar el tamaño que tendrá el mensaje que se va a enviar para inicializar una matriz fila que contendrá el mensaje a transmitir. Se sumarán las longitudes de la entrada dirección, la de comando y se añadirá una posición extra. Esto se hace así para respetar la estructura de un mensaje SDO en CANopen. La longitud de la dirección aporta el tamaño de bytes del índice y del subíndice. La entrada de comando indica la longitud de bytes de datos y por último el valor unitario extra hace referencia al tamaño de datos.
- Se consulta la longitud del comando para definir la primera posición de la matriz con el tamaño del dato que se va a transmitir: 0 bits (40h), 8 bits (2Fh), 16 bits (2Bh) o 32 bits (23h).
- Se añade a la matriz la información del índice y del subíndice.
- Se añade toda la información referente al comando introducido. Una vez esté completo el mensaje se envía especificando que el COB-ID será el 600 (el característico de los SDO).

writePDO1 (comando): por medio de las PDO se mandan mensajes a objetos de dirección específicos de cada driver. Al utilizar la PDO1 se enviará al COB-ID 200 el dato introducido en en el parámetro de entrada 'comando'.

writeNMT (comando): utiliza el servicio de red (NMT: network management) para cambiar los estados del dispositivo de destino y pasar de un NMT a otro. El mensaje NMT consta de una trama de 2 bytes en el que el primer byte hace referencia al nuevo estado y el segundo al ID del nodo. Por ejemplo, se utiliza este mensaje para iniciar el nodo en remoto e inicializa la comunicación con este.

Se diferencia de los intercambios PDO en que estos no necesitan acceder a un registro concreto en el que se especifique el registro+ID (COB-ID) ya que se manda un mensaje a la red general y el segundo byte del comando indica el nodo afectado.

readSDO (comando): método utilizado para realizar la lectura de los datos guardados en un registro del diccionario de objetos concreto. Para ello se especificará el registro que se quiere consultar, para que el driver envíe los datos solicitados. El mensaje tiene la misma longitud que cuando se escribe en una SDO, en la que la transmisión será: Primer byte indica el tamaño de datos (0 bytes: 40h), 3 bytes para el registro y 4 bytes rellenos con valor 00h ya que se solicita información.

PutMsg (COBID,data): este es el método utilizado para realizar la transmisión de los SDO y PDO. Como entrada, se indicará el COB-ID y la cadena que se quiere transmitir.

La función combinará el COB-ID introducido como entrada y el ID del driver que está guardado en el atributo id del objeto. Generará la estructura del mensaje y se transmitirá por el bus para que el nodo al que se le solicite la información genere una respuesta.

getMessage(): recoge todo los mensajes que se encuentran en el buffer de entrada del canal de comunicación y devuelve una matriz con todos los mensajes desglosados para su posterior consulta.

desgloseMsg(messagein): recibe como entrada todos los mensajes del buffer de entrada del canal de comunicaciones. Una vez recibido estos mensajes se comprueba la cantidad de mensajes que se han recibido y se genera una matriz. Dicha matriz constará de 8 columnas y tantas filas como mensajes recibidos. La primera columna guardará el COB-ID de cada mensaje mientras que las 7 restantes guardan los datos.

startMotor(): método utilizado durante la construcción del objeto de la clase **CiA402-Device**, para indicar al driver que debe pasar al modo de arranque remoto. Para ello realiza las llamadas a los métodos writeNMT y writePDO1. Con el segundo método se envía los mensajes para activar el switch del driver.

resetSetPoint(): método utilizado para reiniciar el valor del setpoint de posición por la PDO1 (COB-ID=200+ID). A continuación, en la figura 4.9 se muestran los tres métodos implicados en esta acción.

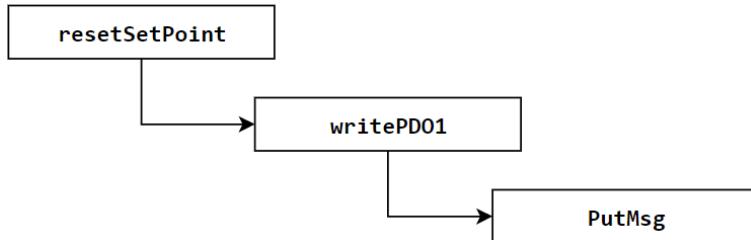


Fig. 4.9. Métodos empleados tras la ejecución del método **resetSetPoint**.

setVelocity(consigna): esta función es utilizada para definir la velocidad a la que el driver comandará el movimiento de los motores. El usuario introducirá la velocidad que desee en rad/s y este método realizará un escalado de la velocidad para que se ajuste a las dimensiones del motor. Tras esto, se construirá el mensaje, se adaptará la consigna del formato de unit_32 a una matriz lineal de 4 bytes. La velocidad mínima a la que se deben mover los motores es de 65536(0x10000) que hace referencia a un 1 incremento por sampleo.

setPosition(consigna): el usuario hará uso de este método para indicar la posición que el driver debe alcanzar. Internamente, el método mandará al driver un mensaje para que habilite el movimiento de la futura consigna de posición. Tras esto, tomará el dato del usuario, lo escalará y lo transformará en una matriz de 4 bytes (Figura 4.10). Guardará el dato en un atributo del objeto y enviará dos mensajes: uno para indicar la nueva posición y otro para iniciar el movimiento.

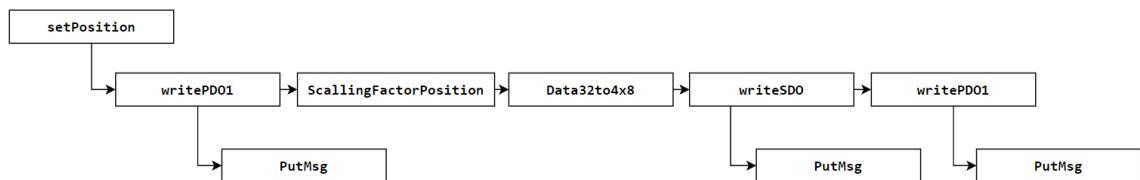


Fig. 4.10. Métodos empleados tras la ejecución del método **setPosition**.

modoOperacion (mode): esta función es empleada para enviar al driver el modo de funcionamiento que debe aplicar para alcanzar las consignas dadas.

setupPositionMode(velocity): método para configurar el funcionamiento del driver en modo posición y el envío de la consigna de la velocidad a la que realizará los movimientos (Figura 4.11).

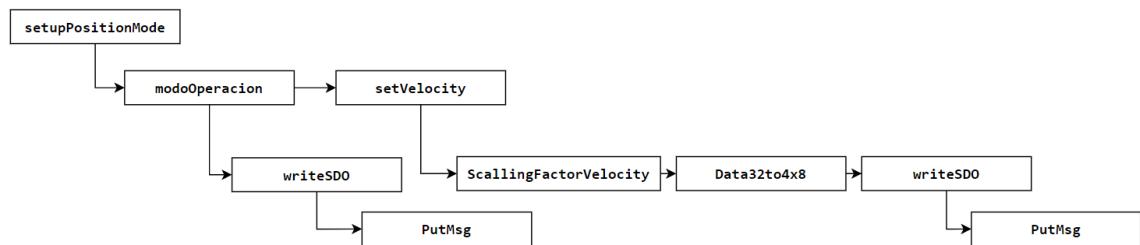


Fig. 4.11. Métodos empleados tras la ejecución del método **setupPositionMode**.

getPosition(): solicitud del dato de posición en el que se encuentra el driver en el momento de realizar la consulta. El método enviará un mensaje solicitando la posición actual y analizará todos los mensajes del buffer de comunicación, devolviendo el dato de la posición actual (Figura 4.12).

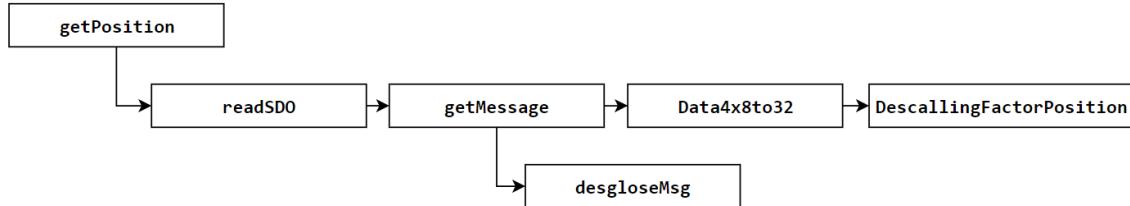


Fig. 4.12. Métodos empleados tras la ejecución del método **getPosition**.

getId(): devuelve la dirección del nodo guardado en el atributo del objeto.

getStatusWord(): solicitud de la word de estado del driver en el momento de realizar la consulta. El método enviará un mensaje solicitando la StatusWord y analizará todos los mensajes del buffer de comunicación, devolviendo el dato de la palabra de estado en formato binario.

printStatusWord(): este método (Figura 4.13) realiza una llamada a la función **getStatusWord()** de la clase para refrescar la información del driver conectado por comunicaciones y posteriormente evalúa la información recibida para mostrarla por pantalla. De esta forma el usuario conocerá el estado en el que se encuentra el driver consultado.

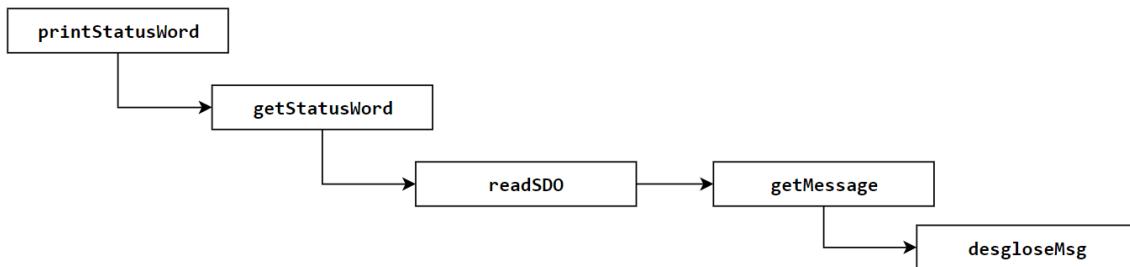


Fig. 4.13. Métodos empleados tras la ejecución del método **printStatusWord**.

ScallingFactorVelocity(): devuelve el valor de escalado de la consigna de velocidad en función de los datos de configuración del driver.

ScallingFactorPosition(): devuelve el valor de escalado de la consigna de posición en función de los datos de configuración del driver.

DescallingFactorPosition(): devuelve el valor de la consigna de posición tras ser desescalado. Es decir, se toma el dato recibido por comunicaciones y se transforma en un valor manejable por el usuario.

Data32to4x8(obj,consigna): este método transforma el valor de una consigna de 32 bits en una matriz de 4 bytes. Esta matriz será la que se envíe en los mensajes del canal de comunicaciones.

Data4x8to32(obj,consigna): este método transforma las matrices de 4 bytes que se reciben por comunicaciones en números de 32 bits, que son interpretables por los usuarios.

Clase Elemento

Es heredera de la clase 'handle' al igual que lo era CiA402DeviceDictionary y por el mismo motivo. La razón por la cual se crea esta clase es para dotar a los usuarios de un elemento de alto nivel por el cual puedan controlar un elemento compuesto de varios dispositivos conectados al bus de comunicaciones CANopen. De esta forma, se aleja al usuario de la capa que se relaciona de forma directa con el protocolo de comunicaciones, permitiéndole trabajar directamente comandando los motores mediante consignas de posición.

En la figura 4.14 se muestra el diagrama de la clase Elemento en la que no existe interrelación entre los métodos, ya que cada uno de las funciones realizará una función única sobre el cuello robótico. Sí que existe relación con los métodos de los objetos de los controladores del cuello y que estos serán accesibles por medio del atributo Driver.

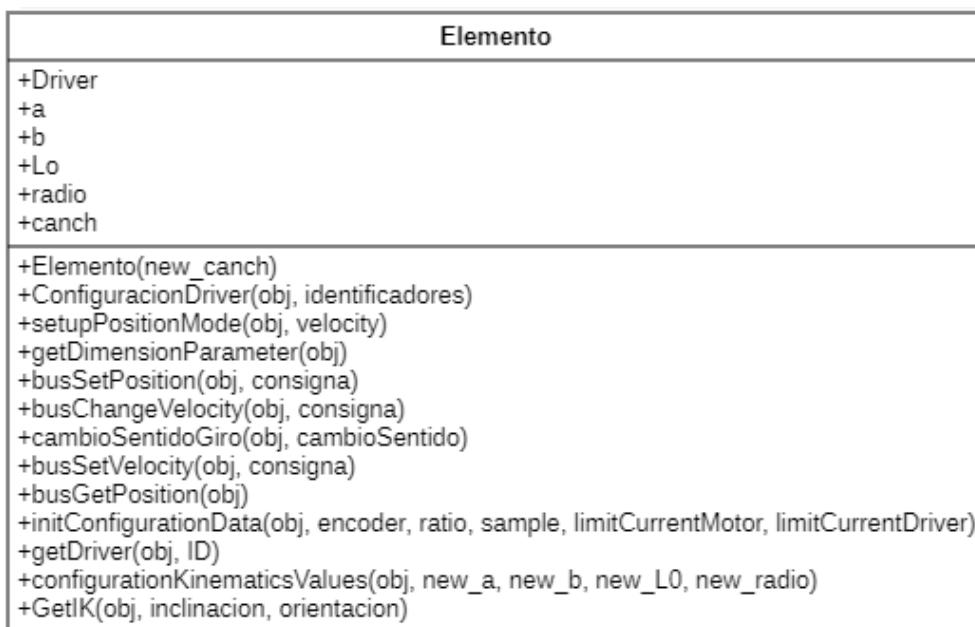


Fig. 4.14. Diagrama de la clase Elemento.

A continuación, se explicará la función de los métodos de esta nueva clase de forma individual. Aquellos que hagan uso de los métodos de los objetos derivados de CiA402Device, tendrán adjunto una figura que refleje su interrelación. Cabe mencionar, que estas funciones realizarán tantas llamadas como controladores (n) tenga la articulación.

Elemento(new_canch): constructor de la clase de un elemento a controlar. El usuario tendrá que indicar el canal por el cual se realiza la comunicación con los nodos del sistema (canch).

ConfiguracionDriver(identificadores): el usuario deberá indicar, por medio de una matriz fila, el número de identificación de los equipos de los nodos conectados en la red CANopen.

Una vez que el usuario haga uso de esta función (fig. 4.15) se crearán los objetos de la clase **CiA402Device** correspondientes a los motores del elemento creado. Al mismo tiempo, se añade en el atributo **Driver** del objeto de la clase elemento el número de los nodos creados.

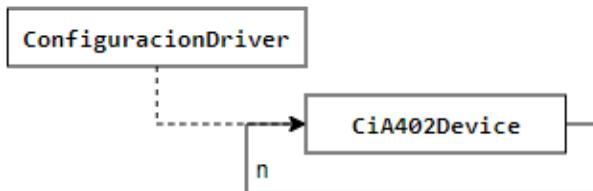


Fig. 4.15. Orden de ejecución tras uso del método **ConfiguracionDriver**.

setupPositionMode(velocity): método (fig. 4.16) utilizado para configurar el modo posición de los drivers que componen el elemento creado de esta clase. Cuenta con un parámetro de entrada referente a la velocidad. Esta velocidad será única y común a todos los motores, rigiendo la rapidez con la que se realizarán los movimientos.

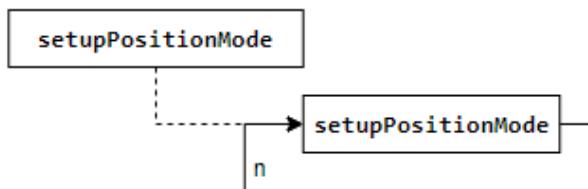


Fig. 4.16. Orden de ejecución tras uso del método **setupPositionMode**.

getDimensionParameter(): devuelve una matriz con la información relativa a los cuatro parámetros configurables para el elemento en cuestión.

busSetPosition(consigna): envío de los nuevos datos de posición a cada uno de los motores que componen el objeto creado (fig. 4.17). En primer lugar, se comprobará que se hayan introducido tantas consignas como motores se hayan configurado en el bus de comunicaciones. De ser así, se enviarán las consignas a todos los elementos del driver. En caso contrario no se realizará el envío de los nuevos setpoints.

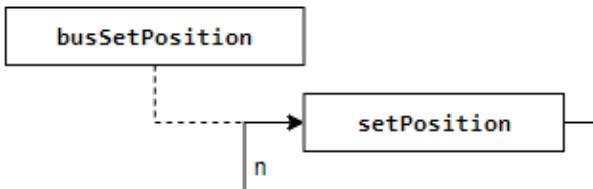


Fig. 4.17. Orden de ejecución tras uso del método **busSetPosition**.

cambioSentidoGiro(cambioSentido): por medio de este método se modifica el sentido de giro de todos los motores que componen el elemento fig. 4.18). El usuario introducirá una matriz en la que indicará con un '1' aquellos controladores que cambien el sentido de giro y con un '0' aquellos que no quiera cambiar. Si las consignas introducidas por el usuario no se corresponden con el número de controladores, se generará un mensaje de advertencia.

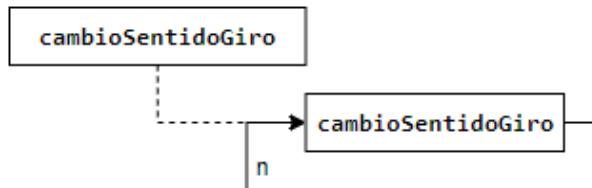


Fig. 4.18. Orden de ejecución tras uso del método **cambioSentidoGiro**.

busGetPosition(): función (fig. 4.19) utilizada para realizar la consulta de la posición de la posición actual de todos los motores que componen el elemento. Se devuelve una matriz fila con tantas columnas como drivers componen el elemento.

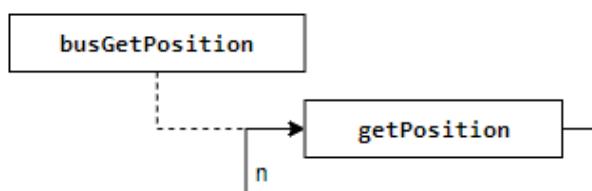


Fig. 4.19. Orden de ejecución tras uso del método **busGetPosition**.

busChangeVelocity(consigna): el usuario, por medio de una consigna, podrá definir la velocidad a la que los motores realicen sus movimientos (fig. 4.20).

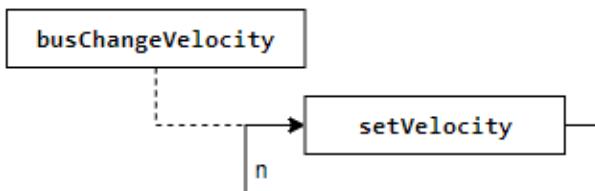


Fig. 4.20. Orden de ejecución tras uso del método **busChangeVelocity**.

busGetStatusWord(): método (fig. 4.21) utilizado para realizar la solicitud de la word de estado de todos los drivers que conforman el elemento. Se realiza una consulta mediante el envío de un mensaje solicitando la StatusWord y se imprime por pantalla el estado de cada uno de los controladores del elemento.

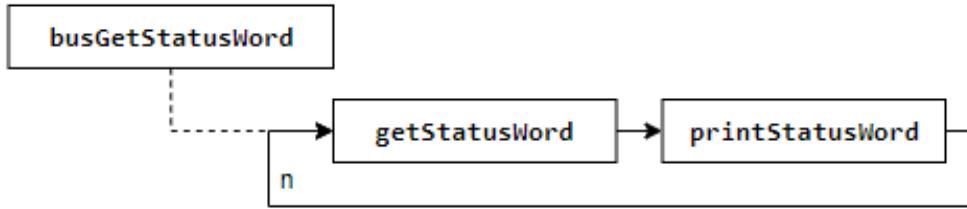


Fig. 4.21. Orden de ejecución tras uso del método **busGetStatusWord**.

busGetTargetPosition(): este método (fig. 4.22) realiza una consulta a todos los controladores para conocer si han alcanzado la posición consignada. Se comprueba la palabra de estado y si todos los motores han alcanzado la posición la función devolverá un '1', mientras que si alguno no ha logrado alcanzar la posición se obtendrá un '0'.

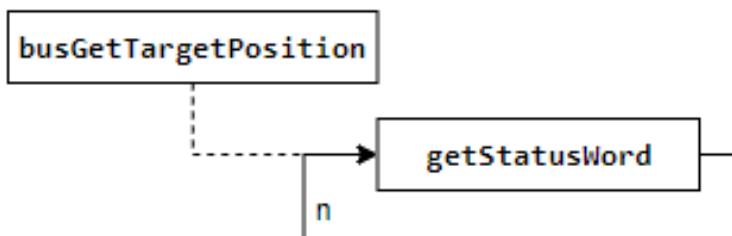


Fig. 4.22. Orden de ejecución tras uso del método **busGetTargetPosition**.

initConfigurationData (encoder, ratio, sample, limitCurrentMotor, limitCurrentDriver): en este método se definen los valores característicos de los drivers utilizados para el correcto funcionamiento de los motores. Los valores indicados serán utilizados para el correcto escalado de las consignas introducidas por el usuario.

getDriver(ID): en caso de que el usuario desee utilizar uno de los drivers de forma independiente, este podrá solicitar el objeto que está asociado a un identificador ID concreto. El método devolverá el objeto solicitado.

Los próximos métodos están relacionados con la cinemática inversa. Tienen el objetivo de estructurar las consignas de entrada en parámetros que puedan ser utilizados por los drivers que componen la articulación. La cinemática inversa se verá en más detalle en la próxima sección 4.4.3.

configurationKinematicsValues(new_a,new_b,new_L0,new_radio): por medio de este método se realiza la configuración de los parámetros dimensionales del elemento a controlar.

GetIK(inclinacion,orientacion): por medio de las consignas de inclinación y orientación introducidas por el usuario, este método devolverá la posición en la que cada motor se debe colocar.

4.4.3. Cinemática inversa

La cinemática inversa es utilizada en el método GetIK(inclinacion,orientacion) transformando las dos consignas introducidas por el usuario en los tres parámetros necesarios para cada uno de los motores de la articulación blanda. Entre los estudios que se han llevado a cabo para aplicar la cinemática en el cuello robótico blando se encuentran las investigaciones [8] [62] y [63], cuyas ecuaciones resultantes se emplearon en este proyecto.

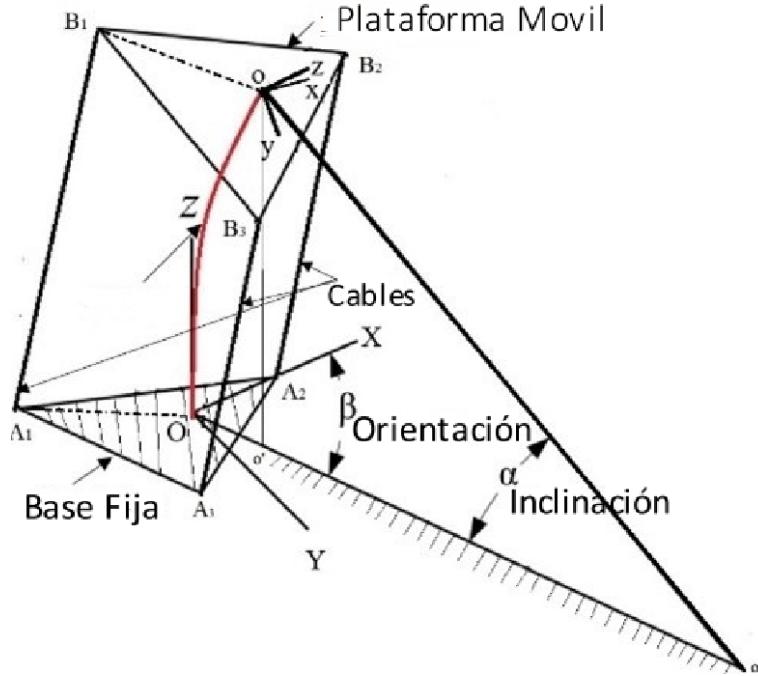


Fig. 4.23. Diagrama del cuello robótico blando [63].

Estas consignas de entrada se introducen como valores angulares, en grados, y que hacen referencia a los parámetros α y β , ambos representados en la figura 4.23. Con la combinación de los ángulos introducidos se determina la longitud de los cables tensores, o tendones, enrollados en los motores para alcanzar la posición deseada. Por medio de las ecuaciones 4.1, 4.2 y 4.3 se obtiene la longitud de los tendones necesaria.

$$L_1 = \sin\left(\frac{\alpha}{2}\right) \left(\frac{2L_0}{\alpha} - 2a \cos\left(\beta - \frac{\pi}{2}\right) \right) \quad (4.1)$$

$$L_2 = \sin\left(\frac{\alpha}{2}\right) \left(\frac{2L_0}{\alpha} - 2a \cos\left(\beta - \frac{7\pi}{6}\right) \right) \quad (4.2)$$

$$L_3 = \sin\left(\frac{\alpha}{2}\right) \left(\frac{2L_0}{\alpha} - 2a \cos\left(\beta - \frac{11\pi}{6}\right) \right) \quad (4.3)$$

Sin embargo, los resultados de estas ecuaciones no son un parámetro valido para ser enviados a los motores del cuello. Al accionarse los motores se va a obtener un movimiento rotatorio que se traduce en una variación de la posición angular del motor, θ . Por

lo que es necesario obtener la nueva posición del motor gracias al uso de la ecuación 4.4, que relaciona la variación de la longitud del tendón con respecto a la posición de reposo L_0 y el radio r de la polea con la que cuenta el motor.

$$\theta_i = \frac{L_0 - L_1}{r} \quad (4.4)$$

Esta última operación se debe realizar para todos y cada uno de los motores que componen la articulación blanda. Para los casos presentados en este proyecto el número de motores ascenderá a tres.

4.4.4. Sensor inercial

Llegados a este punto, ya se han comentado todas las clases y las funciones involucradas en la creación de la librería de comunicación CANopen en MATLAB para un cuello robótico blando.

Aún así, se ha implementado un elemento externo que dará la información de la inclinación y orientación de la articulación. El cual es ajeno al protocolo de comunicación y se encuentra situado sobre la plataforma superior de la articulación. El sensor inercial IMU3DM-GX5-10 es un dispositivo que consta de tres giroscopios y tres acelerómetros, estos son utilizados para medir la velocidad angular y la aceleración en los tres ejes. Son elementos muy sensibles que detectarán los cambios mínimos que se producirán en el movimiento de la parte superior del cuello.

En la versión en C++ del proyecto Humasoft existen una serie de clases en las que se controla la comunicación con el sensor y se realiza el procesamiento de los datos de lectura del equipo.

Debido a que estas clases existen en otra plataforma, en la que están probadas, son funcionales y no es el objeto principal de este proyecto: se procede a realizar la migración del código a MATLAB, adaptándolo al estilo de programación de la nueva plataforma con los comandos correspondientes, dando como resultado las clases antes mencionadas. En la figura 4.24 se muestra la relación de composición de la clase **attitude_estimator** en la clase **IMU3DM_GX5_10**.

Utilizando los métodos de **IMU3DM_GX5_10** se gestionará la interacción con el sensor, ya que el usuario, cuando cree un objeto de esta clase, deberá indicar el puerto de comunicación y la frecuencia a la que se realizarán los muestreos. Una vez creado, se realizará la llamada de forma automática a un método que calibre el sensor. Tras esto, a través del objeto creado de esta clase se podrán realizar peticiones de lectura del sensor para extraer la información necesaria.

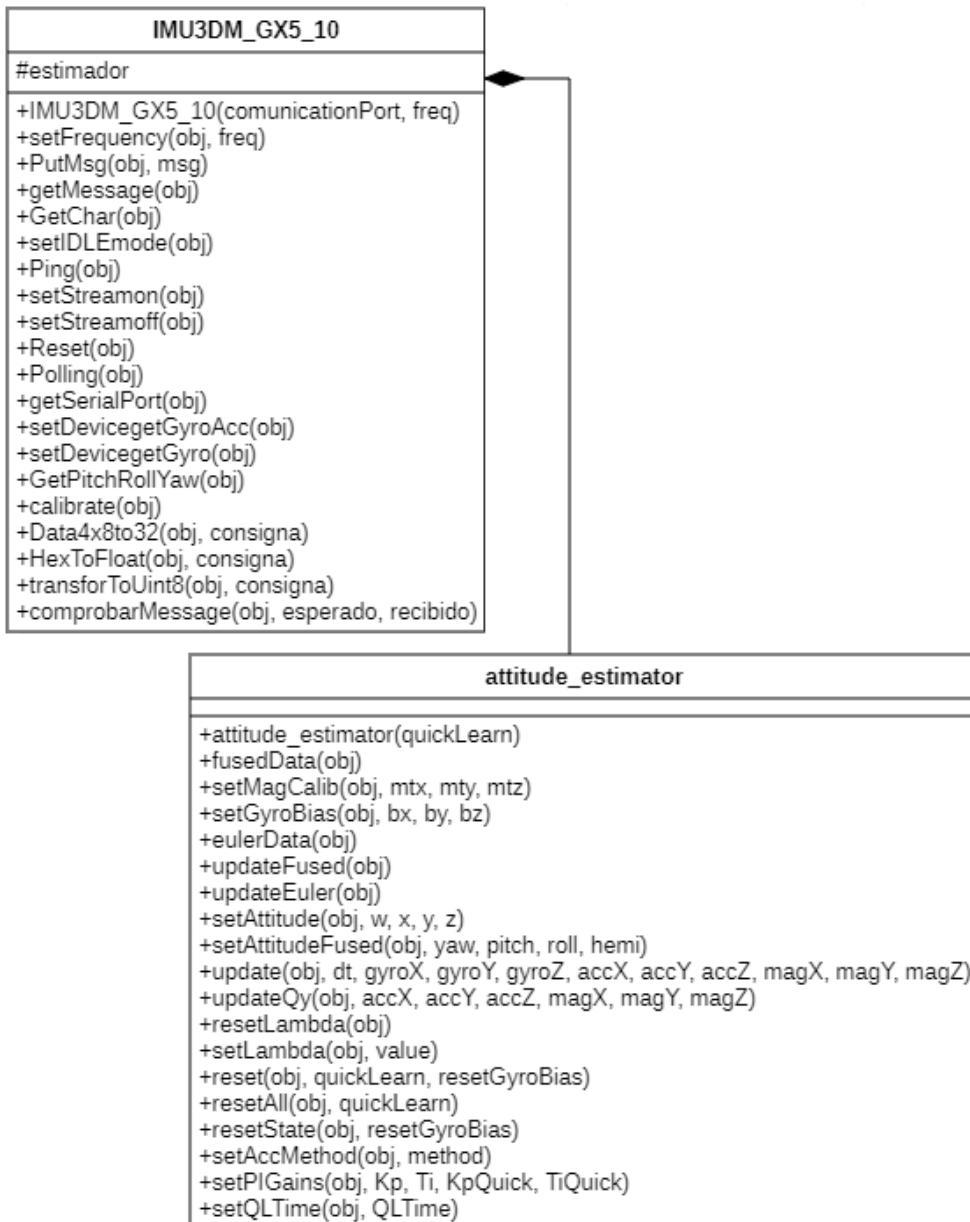


Fig. 4.24. Diagrama de clases relacionadas con el sensor inercial.

La calibración de la IMU se realizará con la articulación en posición de reposo y hará uso de un objeto de la clase **attitude_estimator**, que formará parte del objeto de la clase anterior, el cual contiene los métodos con las ecuaciones fundamentales para procesar los datos obtenidos durante la lectura de datos. Cada vez que el usuario realice la petición de lectura de la medida se hará uso del objeto de esta clase para procesar los datos.

Adicionalmente a estas dos clases, se han programado dos funciones externas que serán utilizadas para transformar los datos procesados por el software en unos parámetros semejantes a las consignas de entrada del usuario y, conociendo la longitud de la articulación, tratar la información para tener la posibilidad de poder recopilar una serie de datos que permitan realizar gráficas en 3D del comportamiento del cuello robótico. Las funciones creadas son las siguientes:

getInclinacionOrientacion: el usuario introducirá los datos de pitch y roll para ser transformados en una matriz de dos posiciones en las que se guardarán la inclinación y la orientación de la articulación.

puntoDeEspacio: esta función combina el dato de la longitud del elemento con la información obtenida de la inclinación y orientación de la articulación en un momento concreto. Se obtiene como resultado una posición en un espacio tridimensional que posteriormente podrá ser usado para realizar una gráfica en 3D para visualizar el movimiento de la articulación robótica.

5. PRUEBAS Y RESULTADOS EXPERIMENTALES

Como se comentó en el capítulo 4 se produjeron distintas etapas a la hora de diseñar la librería de comunicaciones, pero en este apartado se comentarán los puntos más relevantes tratados en las pruebas de desarrollo y validación.

Durante el proceso de desarrollo del sistema de comunicaciones, uno de los problemas que se presentaron fue la discrepancia en el tratamiento de los formatos de los datos en los mensajes enviados y recibidos por parte de MATLAB. La plataforma software muestra los datos numéricos en formato decimal, mientras que en la documentación de los controladores el formato empleado es hexadecimal. Para poder verificar que los mensajes recibidos se correspondían con la información que teníamos del manual de usuario, fue necesario transformar los datos recibidos al formato del manual de usuario de forma externa en Excel. Además, se contó con el software **Serial Port Monitor**, con el que se pueden enviar, recibir y ver mensajes en tiempo real del protocolo CAN, para corroborar los resultados de la transmisión con un programa secundario. Tras validar que la información recibida era la esperada se podía dar por superado este primer paso y proseguir con el desarrollo de los métodos.

Para comprobar que la comunicación entre los distintos componentes del cuello robótico funcionaba correctamente, se llevaron a cabo unas pruebas iniciales con un solo motor, configurando la velocidad de transmisión en 1 Mbit/s. Cabe aclarar que esta velocidad de transmisión es la utilizada para comunicar con los drivers de las articulaciones blandas presentes en este proyecto. Durante esta prueba, se corroboró que el canal de comunicación se establecía correctamente, que se podía mandar las órdenes de encendido del driver y se recibían los mensajes del estado en el que se encontraba el controlador.

Una vez comprobada la funcionalidad del canal de comunicación, se procedió a realizar pruebas para comprobar las funcionalidades del perfil de posición, se enviaron comandos al controlador para que el motor girara unas determinadas vueltas. Esto permitió evaluar que el controlador estaba recibiendo y ejecutando las órdenes correctamente, y que los motores estaban actuando en consecuencia.

Después de haber verificado la funcionalidad del modo posición con un solo motor, se procedió a implementar el código necesario para comandar simultáneamente todos los drivers del cuello robótico.

Finalmente, al haber comprobado el correcto funcionamiento de todos los componentes implicados en la ejecución de los movimientos de la articulación, se procede a la realización de pruebas en las que se produzcan movimientos más amplios. Se llevaron a cabo pruebas en un rango de inclinación de 5 a 35 grados, comandando posiciones de orientación consecutivas para que la articulación diera una vuelta entera alrededor de su eje. En cada ciclo de programa se observa que los motores reajustan su posición hasta

colocarse en la posición adecuada. En la figura 5.1 se muestra al cuello robótico con una inclinación de **35°** y con una orientación de **200°** durante la realización de las pruebas.

Durante la realización de estas pruebas se observó que los motores actuaban para alcanzar las posiciones demandadas. Concluyendo que los controladores estaban actuando correctamente y que a partir de las ecuaciones de la cinemática inversa se estaban generando las consignas correctamente.

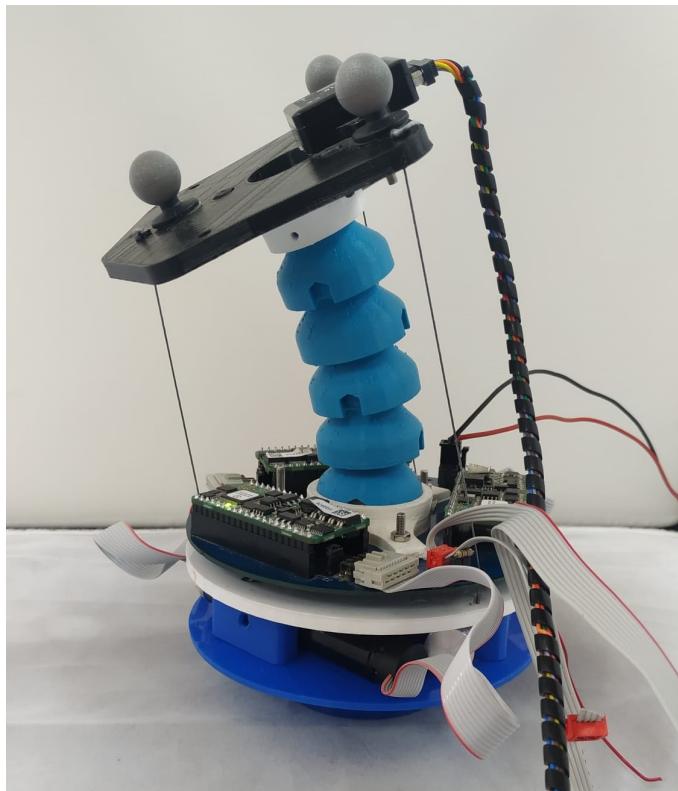


Fig. 5.1. Imagen del cuello robótico durante las pruebas.

A continuación, se aplicó la recolección de datos del sensor inercial, el cual establece una velocidad de comunicación de trabajo de 115200 baudios. Tras el tratamiento de estos por parte de la clase **attitude_estimator**, se presentan las gráficas de la figura 5.2. Las medidas realizadas se presentan junto a las consignas de posición enviadas por el usuario.

Se observa una cierta desviación de los datos recogidos con respecto a las consignas de entrada, especialmente en la gráfica de inclinación donde, a pesar de que observa una cierta tendencia creciente a la hora de aumentar la inclinación del cuello, los datos no son coincidentes a los esperados. Sería necesario el ajuste fino de las ecuaciones de conversión de datos de la IMU que no son objeto de este trabajo.

Por otro lado, aunque los datos de orientación también presentan ligeras diferencias, estos presentan una relación adecuada con los datos consignados por el usuario y que se han podido comprobar de forma visual durante la realización de las pruebas.

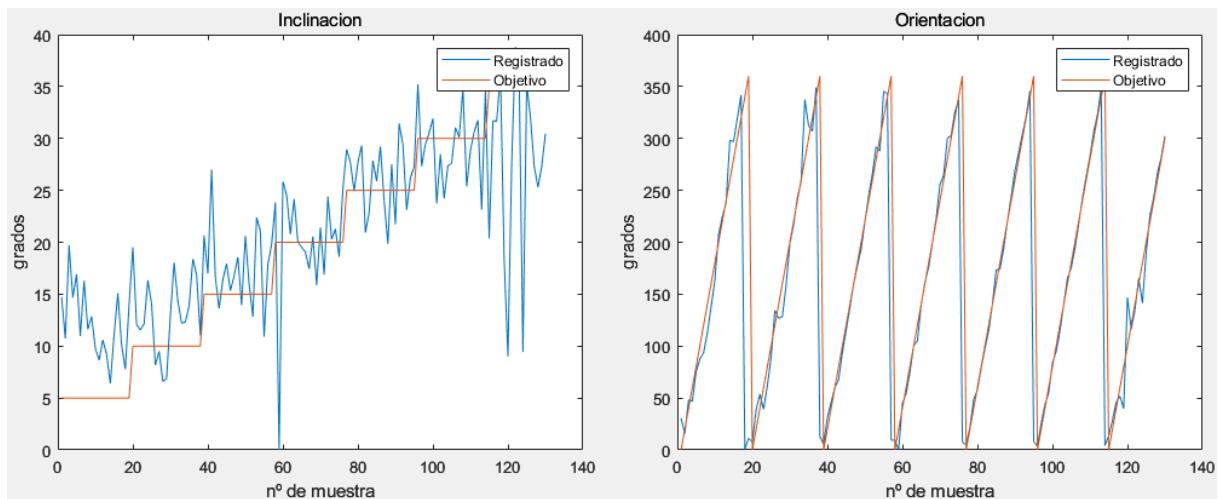


Fig. 5.2. Gráficas de los datos obtenidos por el sensor inercial en el cuello.

Además del cuello robótico, el proyecto SOFIA cuenta con un brazo robótico blando (Figura 5.3) que también utiliza los mismos componentes para su funcionamiento: drivers y sensor inercial. Por esta razón, se decidió probar la librería de comunicaciones diseñada sobre este elemento.



Fig. 5.3. Brazo robótico blando del proyecto SOFIA.

Al igual que las pruebas realizadas sobre el cuello, los resultados obtenidos sobre el

brazo fueron satisfactorios, ratificando que la librería de comunicaciones es funcional sobre distintas articulaciones que utilizan el protocolo CANopen para el comando y control de sus elementos motrices.

En conclusión, las pruebas realizadas demuestran que la comunicación entre los distintos componentes del robot funcionan correctamente y que la librería de comunicaciones diseñada para el proyecto SOFIA es funcional y adaptable a distintas articulaciones robóticas blandas que utilizan el protocolo CANopen para el comando y control de sus elementos motrices. En el vídeo disponible en <https://www.youtube.com/watch?v=6R8J4igELr8> se mostrará una prueba realizada con el cuello robótico durante la validación de la librería.

6. PRESUPUESTO DEL PROYECTO

En el presente capítulo se va a indicar el impacto socio-económico en el que este proyecto podría incurrir, así como los costes asociados a su desarrollo.

En particular, este trabajo está dedicado al diseño y creación de un programa software para dar soporte a un equipo físico ya existente que no debe ser vuelto a construir. Se pueden identificar una serie de gastos relacionados con la mano de obra, el software y el equipo hardware empleado. En la sección 6.2 se van a desglosar estos costes en sus capítulos correspondientes.

6.1. Impacto socio-económico

Al ser un programa utilizado para desarrollar robots blandos, el impacto económico podría ser positivo, ya que estos desarrollos podrían derivar en aplicaciones comerciales o industriales. Los robots blandos tienen el potencial de ser más seguros y más flexibles que los robots rígidos convencionales, lo que puede llevar a una mayor adopción y uso en una variedad de aplicaciones, desde la industria manufacturera hasta la atención médica.

Entre las aplicaciones en las que este tipo de robots pueden verse implicados, se encuentran las interacciones sociales con personas. Estos equipos permiten una interacción más segura con ellos.

En el entorno académico, y en específico en el desarrollo de la robótica blanda en la Universidad Carlos III, este proyecto queda a su disposición para su utilización y desarrollo. Podrá impulsar el desarrollo de nuevos equipos conformados por elementos que se comuniquen vía protocolo CANopen, incluso alejados de la robótica. Al proporcionar una solución en una plataforma tan versátil como es MATLAB, utilizada a su vez por un gran número de ingenieros en el mundo, se podrían reducir el tiempo y el costo de desarrollo de nuevos prototipos, impulsando la innovación. Además, partir de este trabajo podrán sucederse otros que mejoren las capacidades que actualmente posee la aplicación creada.

6.2. Presupuesto

Comenzaremos desglosando los costes de la mano de obra, en la cual se ha contado con la participación de un trabajador, con la categoría de ingeniero junior, el cual podría percibir aproximadamente 15 euros por hora. Este ha realizado un trabajo durante nueve meses y se debe realizar el computo de horas dedicadas, una con la cuantía antes mencionada, se obtendrá el total percibido por su desempeño.

Para conocer el número de horas se recurrirá al Boletín del Estado donde se indica

CAPÍTULO 6. PRESUPUESTO DEL PROYECTO

la equivalencia entre los créditos universitarios matriculados y las horas dedicadas por los estudiantes. Según el **Real Decreto 1125/2003**, por el que se establece el sistema europeo de créditos, se estima que cada crédito corresponde a 25 horas de dedicación por parte del alumnado. La matrícula del Trabajo de Fin Grado consta de 12 créditos lo que equivaldría a un total de 300 horas de trabajo.

Una vez que tenemos todos los datos se procede al realizar el cálculo del coste total de este apartado, arrojando los resultados expuestos en la tabla 6.1.

CAPÍTULO 1: MANO DE OBRA

Descripción	Unidad de medida	Uds.	Precio unitario	Importe total
Ingeniero junior	Horas	300	15,00 €/h	4500,00 €
Coste total				4500,00 €

TABLA 6.1. COSTES TOTALES DE LA MANO DE OBRA.

Una vez presentado el coste de la mano de obra se continuará con los costes relacionados al uso de software. No todas las herramientas utilizadas han generado un coste, algunos de los programas utilizados se pueden obtener de forma gratuita o se puede usar sus versiones Demo. Las aplicaciones que permiten usar sus versiones de prueba tienen un tiempo de uso limitado que no cubre el tiempo completo de dedicación, por lo que para prolongar su uso se debe adquirir una licencia que cuenta con un plazo de validez determinado, por lo general es de 12 meses.

En cuanto al software que no incurre en gasto ninguno se encuentra la licencia de PCAN-View, utilizada por el adaptador del pc a bus de comunicación CAN, y de Sensor-Connect, relacionado con el sensor inercial. Los dos programas que sí que incurren en un coste son MATLAB en su versión académica R2022b y la licencia del Serial Port Monitor, ambos con 12 meses de validez. Por último, se deben computar los gastos derivados del uso e instalación del sistema operativo Windows 10 en el ordenador.

A la hora de contabilizar el coste de las herramientas software, así como se hará del equipo hardware, se deberá considerar el uso del mismo y se tomará como base de medida el tiempo en meses. Para ello se deberá calcular la depreciación durante el tiempo de uso. Esta depreciación vendrá dada por los tiempos de duración de las licencias y en caso de que los programas no tengan un vida útil corta se tomará un plazo máximo de 36 meses.

La ecuación 6.1 por la cual se obtendrá el coste de utilización de cada una de las herramientas viene dado por el cálculo de: el precio del equipo P , el tiempo de uso T_u y el tiempo de depreciación de la herramienta T_d .

$$C_T = P \cdot \frac{T_u}{T_d} \quad (6.1)$$

Por lo que los costes por el uso del software serán los mostrados en la tabla 6.2

CAPÍTULO 2: SOFTWARE

Descripción	Uds.	Precio	Dedicación	Depreciación	Importe total
Windows 10	1	80 €	9 meses	36,00 meses	20,00 €
Licencia de MATLAB Academic	1	262 €	9 meses	12,00 meses	196,50 €
Licencia PCAN-View Basic	1	0 €	-	-	0,00 €
Licencia Serial Port Monitor	1	90 €	9 meses	12,00 meses	67,50 €
Licencia Sensor-Connect	1	0 €	-	-	0,00 €
Coste total					284,00 €

TABLA 6.2. COSTES TOTALES DEL SOFTWARE.

En relación a los costes del hardware, presentado en la tabla 6.3 empleando el criterio de cálculo de costes es el mismo que para el caso del software. En este caso existe una diferenciación entre aquellos equipos que se pueden utilizar en otro proyectos (ordenador, ratón,...) y aquellos de uso exclusivo para este proyecto (PCAN-USB). Estos últimos imputarán su coste completo sobre el presupuesto del trabajo.

CAPÍTULO 3: HARDWARE

Descripción	Uds.	Precio	Dedicación	Depreciación	Importe total
ASUS X540UA	1	350 €	9 meses	60,00 meses	52,50 €
Ratón	1	9,99 €	9 meses	60,00 meses	1,50 €
PCAN-USB	1	280 €			280,00 €
Coste total					334,00 €

TABLA 6.3. COSTES TOTALES DEL HARDWARE.

Para finalizar con el capítulo enfocado en el presupuesto se realiza la recopilación en la tabla 6.4 del coste de cada uno de los aspectos antes tratados, reflejando el coste total del proyecto.

RESUMEN DE COSTES

Capítulo	Descripción	Importe total
1	Mano de obra	4500,00 €
2	Software	284,00 €
3	Hardware	334,00 €
	Coste total	5118,00 €

TABLA 6.4. RESUMEN DE COSTES POR CAPÍTULO.

Asciende el presupuesto general de este Trabajo de Fin de Grado a la expresada cantidad de CINCO MIL CIENTO DIECIOCHO EUROS.

7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1. Conclusiones

El objetivo principal de este Trabajo Fin de Grado ha sido desarrollar e implementar una librería que permita la comunicación efectiva con el cuello robótico blando empleando el protocolo CANopen utilizando como soporte del software la plataforma MATLAB.

Para lograr el objetivo planteado, se ha diseñado una librería según el protocolo OSI de CAN y CANopen, en la que se ha probado que las funciones creadas para la comunicación de los dispositivos de control de la articulación se comuniquen correctamente. Además, se ha adquirido un conocimiento sólido sobre el protocolo de comunicación CANopen.

Se ha realizado la adaptación de la librería del sensor inercial a la plataforma de MATLAB. Aunque los resultados obtenidos no son tan satisfactorios como se esperaban, este es un buen punto de partida para mejorar este aspecto en trabajos futuros, dotando al robot de un buen sistema de medición sobre esta plataforma de programación.

La librería desarrollada cuenta con funciones que emplean la cinemática inversa para el tratamiento de las consignas de los controladores, permitiendo así que los motores de las articulaciones se muevan de forma simultánea y coordinada.

Para validar la solución software propuesta fue necesario realizar ciertas pruebas con el cuello robótico blando y el brazo robótico blando para observar que el resultado del posicionamiento de los elementos era satisfactorio. Lo que demuestra su capacidad para que esta librería sea utilizada en distintas articulaciones y abre la posibilidad de su uso para futuros trabajos, que pueden ser extraídos de este trabajo fin de grado o ser de otra índole y que utilice el protocolo CANopen como base para sus comunicaciones.

La implementación de esta librería abre la posibilidad de utilizar toda la potencia que MATLAB posee para el desarrollo de proyectos de ingeniería de forma directa y eficiente. Además, este trabajo ha sentado las bases para futuros trabajos en los que se puedan utilizar la librería desarrollada, o incluso se puedan llevar a cabo otras líneas de trabajo en este campo.

En conclusión, este Trabajo Fin de Grado ha logrado alcanzar el objetivo principal planteado, que era el desarrollo e implementación de una librería de comunicaciones bajo el protocolo CANopen en MATLAB para un cuello robótico blando.

7.2. Trabajos futuros

En la presente sección se van a realizar una serie de propuestas que podrán ser desarrolladas en trabajos futuros, las cuales son una parte fundamental de cualquier proyecto, ya que permiten establecer una hoja de ruta para la continuidad del mismo. Existen numerosas posibilidades de mejora y ampliación en distintos aspectos.

Una primera línea de trabajo podría centrarse en aumentar las funcionalidades de la librería, como añadir modos de funcionamiento de velocidad y aceleración. Además, sería conveniente implementar una interfaz gráfica de usuario que permita una gestión intuitiva y sencilla de los comandos al cuello robótico, de manera que el usuario pueda obtener información en tiempo real con una disposición amigable y visualmente atractiva.

Otra posibilidad sería utilizar la librería de comunicaciones en otras articulaciones del robot, para desarrollar posteriormente aplicaciones de gestión específicas para cada una de ellas. Esto permitiría crear un sistema completo de control de los sistemas móviles del robot en MATLAB, con lo que se podrían abordar nuevas aplicaciones y técnicas de control de movimiento y posición para así mejorar su funcionalidad.

Asimismo, la librería podría ser utilizada como punto de partida para la integración de nuevos elementos hardware que utilicen los mismos protocolos de comunicación. De esta manera, se podría extender la funcionalidad del cuello robótico a otros sistemas, lo que permitiría crear robots más complejos y con mayores capacidades.

Finalmente, cabe destacar la posibilidad de integrar una IA desarrollada con los elementos de MATLAB. Esta podría estar enfocada en el desarrollo de redes neuronales utilizadas para el aprendizaje y entrenamiento del funcionamiento de todas y cada una de sus funciones, y podría ser extensible a programas más complejos que puedan gestionar los controles de equilibrio del robot humanoide y sistemas que le permitan moverse de forma más eficiente.

En resumen, existen muchas posibilidades de ampliación y mejora de este proyecto, y será necesario seguir trabajando en él para explorar todas sus capacidades y aplicaciones.

BIBLIOGRAFÍA

- [1] S. Monje C. A. y Martínez de la Casa, “SOFIA: Soft intelligent articulation with reconfiguration and modularity capabilities for robotic platforms,” *Ministerio de Economía, Industria y Competitividad*, 2021.
- [2] H. Zhang, W. Liu, M. Yu e Y. Hou, “Design, Fabrication, and Performance Test of a New Type of Soft-Robotic Gripper for Grasping,” *Sensors*, vol. 22, n.º 14, 2022. doi: [10.3390/s22145221](https://doi.org/10.3390/s22145221). [En línea]. Disponible en: <https://www.mdpi.com/1424-8220/22/14/5221>.
- [3] A. Toribio, C. Relaño, C. A. Monje, S. Martínez de la Casa y C. Balaguer, “Identificación de articulación blanda para brazo robótico mediante redes neuronales,” *XLIII Jornadas de Automática: libro de actas*, pp. 843-850, 2022. doi: <https://doi.org/10.17979/spudc.9788497498418.0843>.
- [4] T. Kim, S. J. Yoon e Y.-L. Park, “Soft Inflatable Sensing Modules for Safe and Interactive Robots,” *IEEE Robotics and Automation Letters*, vol. 3, n.º 4, pp. 3216-3223, 2018. doi: [10.1109/LRA.2018.2850971](https://doi.org/10.1109/LRA.2018.2850971).
- [5] C. Monje C. A. y Balaguer, “Diseño y Control de Eslabones Blandos para Robots Humanoides,” *Jornadas Nacionales de Robótica*, pp. 1-6, 2017.
- [6] L. Nagua, C. Monje, J. Muñoz Yañez-Barnuevo y C. Balaguer, “Design and performance validation of a cable-driven soft robotic neck,” 2018, pp. 1-5.
- [7] UC3M. “La UC3M desarrolla articulaciones blandas para robots.” () .
- [8] L. Nagua, *Diseño y simulación de un prototipo de cuello robótico*, 2018.
- [9] N. Puente, *Estudio y puesta en marcha de prototipo de cuello blando de robot humanoide*, 2018.
- [10] CiA. “CANopen – The standardized embedded network.” [Acceso: Marzo 2023]. (), [En línea]. Disponible en: <https://www.can-cia.org/canopen/>.
- [11] PEAKSystem, *PCAN-USB User manual*, 2022.
- [12] Technosoft, *iPOS4808 MX Technical Reference*. 2014.
- [13] Technosoft. “Controlador de motor CANopen iPOS4808 MY-CAN.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: <https://www.directindustry.es/prod/technosoft/product-28695-2380049.html>.
- [14] L. MicroStrain, *3DN-GX5-10 User manual*, 2017.
- [15] G. Metta et al., “The iCub humanoid robot: An open-systems platform for research in cognitive development,” *Neural Networks*, vol. 23, n.º 8, pp. 1125-1134, 2010, Social Cognition: From Babies to Robots. doi: <https://doi.org/10.1016/j.neunet.2010.08.010>. [En línea]. Disponible en: <https://www.sciencedirect.com/science/article/pii/S0893608010001619>.

- [16] P. Fitzpatrick, G. Metta y L. Natale, “Towards long-lived robot genes,” *Robotics and Autonomous Systems*, vol. 56, n.º 1, pp. 29-45, 2008, Human Technologies: “Know-how”. doi: <https://doi.org/10.1016/j.robot.2007.09.014>. [En línea]. Disponible en: <https://www.sciencedirect.com/science/article/pii/S0921889007001364>.
- [17] O. Stasse et al., “TALOS: A new humanoid research platform targeted for industrial applications,” en *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 2017, pp. 689-695. doi: [10.1109/HUMANIODS.2017.8246947](https://doi.org/10.1109/HUMANIODS.2017.8246947).
- [18] K. Kaneko et al., “Humanoid robot HRP-2,” en *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04*. 2004, vol. 2, 2004, 1083-1090 Vol.2. doi: [10.1109/ROBOT.2004.1307969](https://doi.org/10.1109/ROBOT.2004.1307969).
- [19] D. Greenfield. “Entrevista exclusiva de Automation World con Beckhoff.” [Acceso: Abril 2023]. (2021), [En línea]. Disponible en: <https://www.mundopmmi.com/automatizacion/article/21533029/directivos-de-beckhoff-hablan-de-la-evolucion-de-tecnologia-de-automatizacion-a-un-ao-de-la-pandemia>.
- [20] M. Rostan, J. E. Stubbs y D. Dzilno, “EtherCAT enabled advanced control architecture,” en *2010 IEEE/SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, 2010, pp. 39-44. doi: [10.1109/ASMC.2010.5551414](https://doi.org/10.1109/ASMC.2010.5551414).
- [21] B. Automation, *EtherCAT system description*, 2022.
- [22] R. Trask. “What is EtherCAT?” [Acceso: Marzo 2023]. (2022), [En línea]. Disponible en: <https://www.controleng.com/articles/what-is-ethercat/>.
- [23] G. Ribichini. “¿Qué es el protocolo EtherCAT y cómo funciona?” [Acceso: Abril 2023]. (2021), [En línea]. Disponible en: <https://dewesoft.com/es/blog/que-es-protocolo-ethercat#ethercat-history>.
- [24] A. motion controls. “Comunicación RS-232.” [Acceso: Marzo 2023]. (), [En línea]. Disponible en: <https://www.a-m-c.com/es/experiencia/technologies/peripheral-interface/rs-232/>.
- [25] N. G. Forero Saboya, “Normas de Comunicación en Serie: RS-232, RS-422 y RS-485,” *Ingenio Libre*, vol. 11, pp. 86-94, jun. de 2012.
- [26] Telefónica, “Manual del sistema - NETCOM neris 4/8/64 I5,” pp. 817-830,
- [27] O. Weis. “Interfaz de comunicación serie. Pinout RS232.” [Acceso: Marzo 2023]. (), [En línea]. Disponible en: <https://www.virtual-serial-port.org/es/article/what-is-serial-port/rs232-pinout/>.
- [28] D. S. Dawoud y P. Dawoud, “1 Serial Communication,” en *Serial Communication Protocols and Standards RS232/485, UART/USART, SPI, USB, INSTEON, Wi-Fi and WiMAX*. 2020, pp. 1-46.

BIBLIOGRAFÍA

- [29] O. Weis. “Diferencia entre RS232 y RS485 - casos de uso y tecnología.” [Acceso: Marzo 2023]. (2020), [En línea]. Disponible en: <https://www.virtual-serial-port.org/es/article/what-is-serial-port/rs232-vs-rs485.html>.
- [30] CiA. “CiA 402 series: CANopen device profile for drives and motion control.” [Acceso: Marzo 2023]. (), [En línea]. Disponible en: <https://www.can-cia.org/can-knowledge/canopen/cia402/>.
- [31] J. Vizárraga, *Interfaz de control sobre Python y C++ para cuello robótico blando*, 2020.
- [32] H. Zimmermann, “OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection,” *IEEE Transactions on Communications*, vol. 28, n.º 4, pp. 425-432, 1980.
- [33] R. Sánchez, *CANopen*.
- [34] “SLLA 279: Controller Area Network physical layer requirements,” *Texas Instruments Inc.*, p. 5, 2008.
- [35] “ISO 11898-2:2016 Road vehicles - Controller area network (CAN) - Part 2: High-speed medium access unit,” *INTERNATIONAL STANDARD ISO*, p. 1, 2016.
- [36] P. Marius, C. TOMA, C. Boja y Z. Alin, “Privacy and Security in Connected Vehicles Ecosystems,” *Informatica Economica*, vol. 21, pp. 29-40, dic. de 2017. doi: [10.12948/issn14531305/21.4.2017.03](https://doi.org/10.12948/issn14531305/21.4.2017.03).
- [37] “ISO 11898-3:2016 Road vehicles - Controller area network (CAN) - Part 3: Lowspeed, fault-tolerant, medium-dependent interface,” *INTERNATIONAL STANDARD ISO*, p. 1, 2016.
- [38] “ISO 11898-1:2015 Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signalling,” *INTERNATIONAL STANDARD ISO*, p. 1, 2016.
- [39] CiA. “Classical Controller Area Network (CAN).” [Acceso: Marzo 2023]. (), [En línea]. Disponible en: <https://www.can-cia.org/can-knowledge/can/classical-can/>.
- [40] CiA. “Standardized higher-layer protocols.” [Acceso: Marzo 2023]. (), [En línea]. Disponible en: <https://www.can-cia.org/can-knowledge/hlp/standardized-higher-layer-protocols/>.
- [41] CiA. “CANopen history.” [Acceso: Marzo 2023]. (), [En línea]. Disponible en: <https://www.can-cia.org/can-knowledge/canopen/canopen-history/>.
- [42] CORDIS. “Automation and Control Systems for Production Units Using an Installation Bus Concept.” [Acceso: Marzo 2023]. (), [En línea]. Disponible en: <https://cordis.europa.eu/project/id/7302>.
- [43] N. Instruments. “The Basics of CANopen.” [Acceso: Marzo 2023]. (), [En línea]. Disponible en: <https://www.ni.com/es-es/innovations/white-papers/13/the-basics-of-canopen.html>.

- [44] Technosoft, *iPOS CANopen Programming. User Manual*. 2019.
- [45] S. Automation, *Motion Drive.Digital drive for Brushless motors SMD Series*, 2019.
- [46] C. Electronics. “CANopen Explained - A Simple Intro [2022].” [Acceso: Marzo 2023]. (), [En línea]. Disponible en: <https://www.csselectronics.com/pages/canopen-tutorial-simple-intro>.
- [47] “canChannel.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: <https://es.mathworks.com/help/vnt/ug/canchannel.html>.
- [48] “canChannelList.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: <https://es.mathworks.com/help/vnt/ug/canchannellist.html>.
- [49] “configBusSpeed.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: <https://es.mathworks.com/help/vnt/ug/can.channel.configbusspeed.html>.
- [50] “canMessage.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: <https://es.mathworks.com/help/vnt/ug/canmessage.html>.
- [51] “can.Message Properties.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: <https://es.mathworks.com/help/vnt/ug/can.message-properties.html>.
- [52] “transmit.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: <https://es.mathworks.com/help/vnt/ug/can.channel.transmit.html>.
- [53] “receive.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: <https://es.mathworks.com/help/vnt/ug/can.channel.receive.html>.
- [54] “CiA 402 Power State Machine.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: https://en.nanotec.com/products/manual/NP5_CANopen_EN/general%5C%2Fds402_power_state_machine.html.
- [55] “classdef.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: <https://es.mathworks.com/help/matlab/ref/classdef.html>.
- [56] “handle class.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: <https://es.mathworks.com/help/matlab/ref/handle-class.html>.
- [57] “Property Syntax.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: https://es.mathworks.com/help/matlab/matlab_oop/defining-properties.html.
- [58] “Property Attributes.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: https://es.mathworks.com/help/matlab/matlab_oop/property-attributes.html.
- [59] “Initialize Property Values.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: https://es.mathworks.com/help/matlab/matlab_oop/initialize-property-values.html.
- [60] “methods.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: <https://es.mathworks.com/help/matlab/ref/methods.html>.

BIBLIOGRAFÍA

- [61] “Method Syntax.” [Acceso: Abril 2023]. (), [En línea]. Disponible en: https://es.mathworks.com/help/matlab/matlab_oop/specifying-methods-and-functions.html.
- [62] L. Nagua, C. Monje, J. Muñoz y C. Balaguer, “Design and performance validation of a cable-driven soft robotic neck,” 2018, pp. 1-6.
- [63] N. A. Continelli, L. F. N. Cuenca, C. A. Monje y C. Balaguer, “Modelado de un cuello robótico blando mediante aprendizaje automático,” *Revista Iberoamericana de Automática e Informática industrial*, 2023. doi: [10.4995/riai.2023.18752](https://doi.org/10.4995/riai.2023.18752).