UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

TRABAJO FIN DE GRADO:

ESTUDIO Y PUESTA EN MARCHA DE PROTOTIPO DE CUELLO BLANDO DE ROBOT HUMANOIDE

Autor: Natalia Puente Carreño
Directora: Concepción Alicia Monje Micharet

Agradecimientos

Al término de mi trabajo en este proyecto, me doy cuenta de que está cimentado en el esfuerzo, apoyo y ayuda de muchas personas. Por ello, me gustaría dedicarles unas palabras a todas ellas:

En primer lugar, me gustaría agradecer a mi tutora, Concha, que con presencia, dedicación y cercanía me ha orientado y enseñado durante todos estos meses de trabajo.

Quiero agradecer igualmente a todos los miembros del grupo HUMASoft, que me integraron y acogieron desde el primer momento, y con los que he trabajado codo con codo durante meses. No solo he obtenido de ellos conocimiento y colaboración, también apoyo, compañerismo y amistad.

Y en general querría mandar un saludo a todas las personas de RoboticsLab con las que he tenido el placer de compartir laboratorio, siempre dispuestas a tender un brazo cuando se les necesitaba.

También me gustaría mencionar mi gratitud por el apoyo incondicional de mi familia. Sus consejos, comprensión y cariño me han empujado a seguir adelante y luchar para alcanzar mis metas. Por ello, quiero agradecer a mis padres, Antonio y Elena, a mis hermanos, Alberto y Enrique, y por supuesto a David.

Y finalmente, a todos aquellos que han puesto su granito de arena para que haya llegado hasta aquí, gracias.

Resumen

El presente Trabajo Fin de Grado tiene como propósito la puesta en marcha de un prototipo de cuello blando para el proyecto HUMASoft de la Universidad Carlos III de Madrid. Para el desarrollo de dicho cuello, ha sido necesaria la realización de tareas de diseño, montaje y puesta en marcha del prototipo. La peculiaridad de este proyecto es el uso de la robótica blanda en un robot capaz de soportar cargas relativamente altas, logrando una cinemática de dos grados de libertad que incorpora como elemento mecánico un resorte.

Una tarea de gran relevancia en el proyecto es el montaje de la plataforma del cuello a partir de piezas impresas con impresora 3D. A lo largo del proyecto, varios diseños y modificaciones fueron propuestos atendiendo a las necesidades que se encontraron. Finalmente se obtuvo una plataforma desmontable capaz de albergar diversas variaciones, tales como la posibilidad de cambiar el cuello blando por otros eslabones de diversos materiales.

Para dirigir el prototipo, se han desarrollado unas librerías de comunicación entre el ordenador y los drivers que gobiernan el sistema. Estas librerías han sido escritas en C++ y fueron clave para la realización de la aplicación de usuario donde se implementa la cinemática inversa del sistema.

La fase final fue la puesta en marcha global del prototipo. Es decir, la plataforma con su configuración final es conexionada con los drivers, y estos con el ordenador. Posteriormente, se hace uso de las ecuaciones de cinemática inversa y las librerías de comunicación para, dados unos parámetros de inclinación y orientación del cuello, comandar a una posición determinada simultáneamente cada uno de los motores.

El trabajo de investigación realizado a lo largo de estos meses propiciará un desarrollo más profundo del prototipo, ya sea con modificaciones y mejoras en su diseño, como con el análisis de lazos de control para mejorar el rendimiento y el funcionamiento del cuello robótico.

Abstract

The purpose of this Final Bachelor Thesis is to perform the start-up of a soft neck prototype for the HUMASoft project of the Universidad Carlos III de Madrid. For the development of the aforesaid neck, it has been necessary to carry out tasks of design, assembly and start-up of the prototype. The peculiarity of this project is the use of soft robotics in a robot capable of supporting relatively high loads, achieving a kinematics of two degrees of freedom using a spring as mechanical element.

A task of great relevance in the project is the assembly of the neck's platform from printed pieces with 3D printer. Throughout the project, several designs and modifications were proposed according to the needs that were found. Finally, a removable platform was obtained, capable of admitting multiple variations, such as the possibility of changing the soft neck by other links made from various materials.

To run the prototype, a communication library has been developed to perform the communication between the computer and the drivers that govern the system. These libraries have been written in C ++ and were key for the realization of the user's program where the inverse kinematics of the system is implemented.

The final phase was the global start-up of the prototype. That is to say, the platform with its final configuration is connected to the drivers, and these to the computer. Subsequently, the inverse kinematics and the communication libraries are used to, given certain inclination and orientation parameters, command each of the three motors simultaneously to a determined position.

The research work carried out over these months will lead to a deeper development of the prototype with modifications and improvements in its design, as well as the analysis of control loops to improve the performance and functioning of the robotic neck.

Índice

Índice de figuras

Índice de tablas

1. Introducción

En este capítulo se introducen los conceptos básicos de la robótica blanda. Dicha descripción se ve apoyada por ejemplos de aplicaciones reales en proyectos de investigación de universidades europeas. Del mismo modo, se han añadido las bases del proyecto de robótica blanda HUMASoft para el que se ha realizado el presente Trabajo Fin de Grado y la influencia de su desarrollo.

Se pueden encontrar igualmente en este capítulo dos apartados destinados a ubicar el proyecto tanto en el entorno socio-económico, como en el marco regulador actual.

Existe además un apartado conteniendo la organización del trabajo realizado, determinada principalmente por las motivaciones y objetivos que han propiciado el desarrollo de este trabajo de investigación. Finalmente, se ha completado este capítulo con una descripción de la estructura del documento, añadiendo un pequeño resumen del contenido de cada capítulo que lo compone.

1.1. Robótica blanda

Los robots han adquirido con el tiempo más complejidad y precisión en sus movimientos, al igual que mayor inteligencia y exactitud en el control en su motricidad [1]. Sin embargo, el uso de eslabones rígidos en robots ha sido una constante fundamental desde los primeros avances en esta rama. Si bien, existen limitaciones en la realización de ciertas tareas para las cuales es necesario indagar en tecnologías revolucionarias.

Supone, por lo tanto, una innovación romper esa barrera e introducir eslabones blandos en la investigación. La motivación para investigar en esta tecnología está promovida principalmente por nuevos paradigmas científicos, pero también para cubrir nuevas necesidades y aplicaciones [2]. Los robots denominados dentro de "soft robotics" no solo suponen una nueva dirección de desarrollo tecnológico, sino un creciente potencial para una nueva generación de robots, asistiendo a humanos en nuestro propio entorno.

El término "soft robotics", o "robótica blanda" se utilizó inicialmente para designar aquellos robots con eslabones rígidos y, articulaciones y actuadores de rigidez variable. Actualmente, "soft robotics" engloba también los eslabones flexibles. De hecho, el concepto «blando» puede introducirse en la robótica de múltiples

formas: textura blanda, materiales blandos y deformables, movimientos blandos, materiales elásticos y actuadores de rigidez variable, e interacciones blandas entre humano y máquina.

Actualmente se puede tomar como ejemplo los siguientes proyectos internacionales:

- OCTOPUS (SSSA, Italia), inspirado en un invertebrado marino con capacidades motoras y comportamiento inteligente. Por el momento, OCTOPUS tan sólo posee un único brazo bioinspirado en el de un pulpo, pero en el futuro se espera tener un conjunto de ocho brazos. Las capacidades de este tentáculo consisten en alargase tanto como retraerse, tirar de un objeto ligero o ser capaz de moverse en cualquier dirección, lo que le confiere infinitos grados de libertad. El concepto blando de este prototipo consiste principalmente en una base de silicona dirigida con cables y diversas tecnologías SMA.[3] Esto permite evitar cualquier estructura rígida en su anatomía, pero consecuentemente tiene como desventaja la incapacidad de ejercer grandes esfuerzos mecánicos o soportar grandes cargas. En la Figura 1 se observa dicho brazo robótico implantado en dos de los brazos de una estructura similar a un pulpo.

Figura 1: Robot blando OCTOPUS de la Scuola Superiore Sant' Anna (SSSA).

- Proyecto Soft Robotics (TU DELFT, Países Bajos), consiste en un conjunto robótico de antebrazo y mano, compuesto de materiales blandos, que es capaz

de dar un apretón de manos. La funcionalidad de este robot está orientada a usos ortopédicos, protésicos, o como robot asistencial o exploratorio. Para el desarrollo de este robot, sus distintos componentes han sido impresos en una impresora 3D utilizando un determinado patrón de densidad y obteniendo, de esta forma, materiales blandos. Su actuación también es blanda, pues utiliza para este cometido aire a presión. Adicionalmente, el robot es capaz de controlar la fuerza que ejerce en el apretón de manos adaptándose al entorno y haciendo más seguras e intuitivas las tareas de interacción Humano-Robot [5]. En la Figura 2 se observa el apretón de manos entre este robot y una persona.

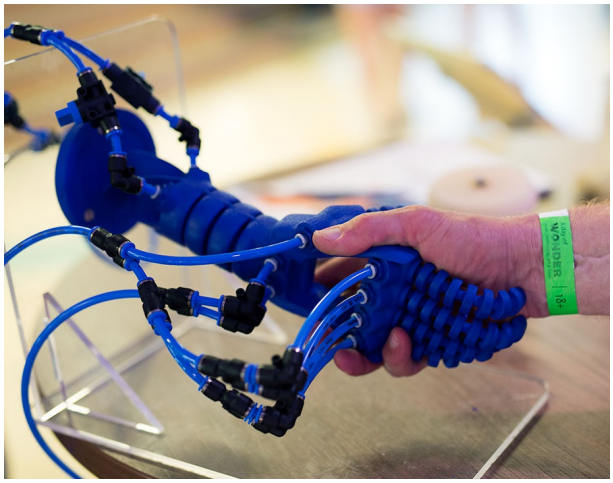

Figura 2: Proyecto Soft Robotics de la Universidad Técnica de Delft (TU DELFT).

1.1.1. La robótica blanda y los robots humanoides

Hoy en día la investigación en robótica está fuertemente orientada al uso de robots en tareas humanas. De ahí el surgimiento de las actuales líneas de investigación en robots humanoides con una fisionomía similar a la de una persona. El objetivo de este tipo de máquinas es el acercamiento a los humanos, ya sea realizando trabajos efectuados típicamente por humanos, o interactuando con estos por diversos motivos.

En los robots humanoides, la tecnología blanda se presenta en la mayoría de los casos como Compliant Manipulation (Manipulación Adaptable). Este concepto se aplica a robots que pueden controlar la fuerza que efectúan adaptándola a su entorno. Esto proporciona un entorno más seguro, así como la posibilidad de

efectuar movimientos o maniobras que resultaban imposibles anteriormente.

Esta tecnología está siendo ampliamente utilizada en la investigación en robótica humanoide. A continuación, se mencionarán dos robots humanoides desarrollados en universidades europeas que utilizan en gran medida la manipulación adaptable.

- Rollin' JUSTIN (DLR, Alemania), es un robot humanoide de hasta 51 grados de libertad que fue inicialmente diseñado como punto de partida para la investigación en tareas de manipulación complejas. Estaba principalmente orientado hacia el desarrollo de tecnología capaz de crear robots que ofrezcan asistencia en tareas domésticas. Con la integración de la Manipulación Adaptable de Cuerpo Entero, el Robot Justin es capaz de realizar diversas tareas donde imita la manipulación humana [4]. En la Figura 3 se muestra una imagen de cuerpo completo de este humanoide.

Figura 3: Robot Humanoide Rollin' Justin desarrollado por el Instituto de Robótica y Mecatrónica (DLR).

- COMANOID (CNRS, Francia), utilizado en operaciones de ensamblaje de aviones Airbus que resulten laboriosas o tediosas para trabajadores humanos, o para los cuales el acceso a dichas operaciones resulte imposible. Entre los objetivos de COMANOID se encuentra la de evaluar hasta qué punto la tecnología de robótica humanoide resulta eficiente en la operativa de la

fabricación de aviones. En este caso el robot ha de ser capaz de adaptarse a un entorno con sistemas rígidos y flexibles que han de ser tratados con cuidado. Igualmente, al tratarse de espacios no ergonómicos el robot deberá desarrollar estrategias para ser capaz de alcanzar la posición objetivo, manteniendo la estabilidad y su capacidad de trabajo y sin perjudicar a su entorno [6]. En la Figura 4 se muestra una simulación del humanoide COMANOID en un entorno de trabajo.

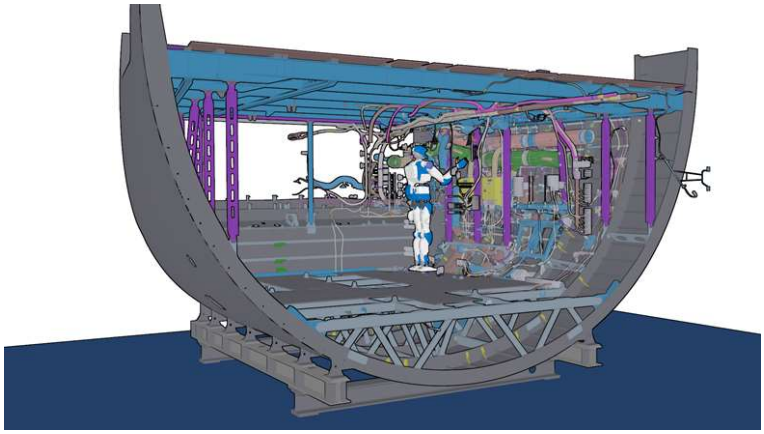

Figura 4: Simulación de una operación de ensamblaje o revisión del robot COMANOID del Centre National de la Recherche Scientifique (CNRS).

Como se ha dicho anteriormente los robots con tecnología de robótica blanda presentan diversas ventajas respecto a sus predecesores. Cierto es que igualmente también conllevan un conjunto de inconvenientes propios de este tipo de tecnología.

Por un lado, el control y comportamiento dinámico del sistema resulta de una complejidad mucho más alta frente a los robots tradicionales. Conjuntamente, es necesario un riguroso estudio de su modelo teórico que resulta ser más enrevesado e imprevisible a la hora de su implementación. Estas han sido algunas de las dificultades que se debieron solventar para presentar una solución funcional al prototipo.

1.2. Proyecto HUMASoft

El proyecto HUMASoft [8] hace uso esta tecnología blanda para su introducción en la robótica humanoide. El uso de eslabones blandos presenta diversas ventajas

frente a un diseño completamente rígido:

- **Representa un diseño más simple.** Se logra una arquitectura en la que se necesita una actuación menor, derivando en un número inferior de actuadores para un mayor número de grados de libertad de la plataforma.
- **Aumenta la capacidad de adaptación y flexibilidad en ciertos entornos.** Existen ciertos entornos donde la utilización de un robot rígido resultaría muy complicada o imposible, como podría ser una situación de rescate.
- **Resulta en una interacción más directa, segura y sencilla con humanos o con el medio.** Estos robots pueden adquirir la habilidad de adaptarse a su entorno y ser capaces de absorber impactos.

El proyecto HUMASoft propone abrir nuevas líneas de investigación para rediseñar las diferentes partes del humanoide TEO del grupo de investigación RoboticsLab del Departamento de Ingeniería de Sistemas y Automática [7]. Las partes de las que se pretende realizar modelos alternativos blandos son brazos, cuello o columna vertebral. De esta forma se consigue incorporar las ventajas que aporta la tecnología blanda de simplicidad, accesibilidad y seguridad, manteniendo igualmente la estabilidad y funcionalidad de dichas partes.

1.2.1. TEO y la incorporación del proyecto HUMASoft en forma de cuello blando

El humanoide TEO nació en el año 2012 y es pionero al ser uno de los primeros humanoides bípedos de Europa. TEO fue concebido como un robot ligero, con un peso de aproximadamente 60 kg y estatura muy similar a la humana, que fuera capaz de realizar tareas domésticas entre las que ya se encuentran planchar o ejercer de camarero. Esto es posible debido a que el humanoide consta de diversas extremidades, bien definidas y funcionales, inspiradas en las de un ser humano y para las que son necesarios un total de 24 grados de libertad. En la Figura 5 encontramos el robot humanoide TEO desarrollado por el grupo de investigación RoboticsLab. La siguiente imagen, la Figura 5, muestra el estado actual de TEO en el laboratorio donde se investiga también para el proyecto HUMASoft.

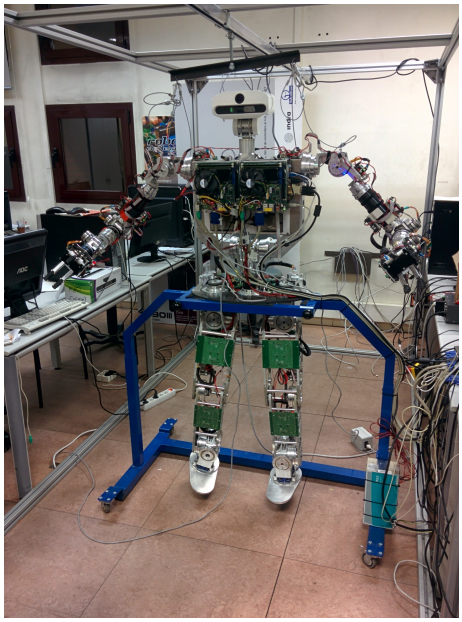

Figura 5: Robot humanoide TEO para el cual se está desarrollando la tecnología de HUMASoft (UC3M).

Dos de estos grados de libertad corresponden al cuello actual de TEO sobre el que descansa la cabeza del robot. Está compuesta de una carcasa de una cámara Kinect en la que se ha instalado la cámara monocular PointGrey FLEA y un sensor de RGB XTion Pro Live de ASUS.

El proyecto HUMASoft propone para TEO un prototipo de cuello alternativo que utilice robótica blanda. El objetivo es realizar una primera aproximación a un cuello blando e iniciar una vía de investigación en esta dirección. El desarrollo de la plataforma se realizará independientemente de TEO, sin embargo, la tecnología desarrollada deberá ser completamente compatible con la del humanoide, siendo este uno de los principios de la investigación de HUMASoft. Por esta razón, y como se describirá más adelante en el capítulo del diseño del cuello, se han utilizado motores, encoders y drivers similares a los usados en TEO; de forma que su mantenimiento sea más rápido y efectivo y, en caso de incorporar esta o una versión superior del cuello a TEO, el montaje y tareas de puesta en marcha se realicen con sencillez. Se ve ilustrado en la Figura 6 una representación de la incorporación del cuello de HUMASoft en la configuración actual del humanoide TEO.

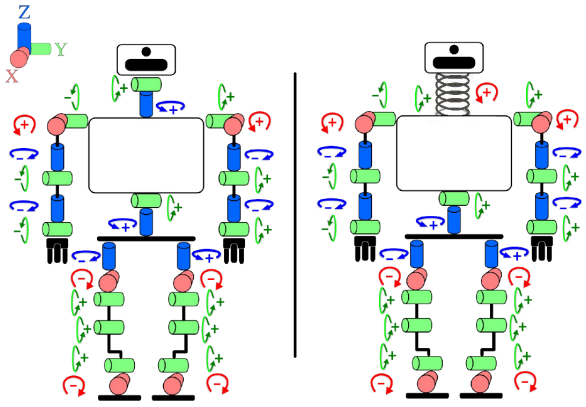

Figura 6: Incorporación del prototipo del cuello de HUMASoft en el humanoide TEO.

1.3. Impacto en el entorno socio-económico

Como se ha planteado anteriormente, la robótica blanda presenta diversas ventajas desde el punto de vista de ligereza, alta resistencia mecánica y sensibilidad. Sin embargo, las auténticas ventajas de este proyecto aparecen en la interacción humano-robot. Puesto que el objetivo del proyecto es la integración de partes blandas en un robot humanoide, el impacto generado por el proyecto está igualmente relacionado con este tipo de robots.

Un robot humanoide con tecnología blanda se presenta mucho más cercano a la hora de interactuar con humanos. Por un lado, realiza movimientos más naturales y similares a los de los seres humanos, lo que implica cercanía en múltiples aplicaciones. Por otro lado, se logra un mayor control de la sensibilidad y actuación del sistema, lo que lograría crear un entorno más seguro alrededor del robot, y adicionalmente este sería capaz de poseer habilidades complejas. Estos atributos, por lo tanto, suponen que se puedan usar este tipo de robots en tareas realizadas por humanos. Esto resulta de gran utilidad en el caso particular del humanoide TEO, que realiza tareas de planchado y doblado de ropa o de camarero, mejorando la realización de estas tareas y por lo tanto reemplazando al trabajador que necesite realizarlas.

Por otro lado, a escala menor, también se debe considerar el impacto que supone la investigación en la plataforma de cuello blando individualmente, dado que la implementación de esta tecnología en TEO no sería posible sin su desarrollo de

forma previa. El cuello blando permite realizar el primer acercamiento a los objetivos finales del proyecto HUMASoft, trasladando más adelante la investigación realizada a otros eslabones del humanoide. Adicionalmente, el prototipo supone una innovación en el terreno de la robótica blanda, que servirá de base para proyectos futuros en esa rama.

Estas características permiten identificar los diferentes tipos de impactos que puede ocasionar el desarrollo del prototipo:

- **Impacto académico/científico:** Desde una perspectiva académica/científica, este proyecto pone a disposición de la comunidad científica el desarrollo de la tecnología necesaria en el prototipo y su posterior documentación, de forma que se abren nuevas vías de investigación a partir de los adelantos que se lleven a cabo en este proyecto. De manera paralela, también incentivará la creación de documentos técnicos tales como publicaciones, así como participaciones en congresos y otros eventos relacionados con la investigación científica en robótica. De igual manera, implica un impacto en la comunidad universitaria, pues diversos proyectos académicos están siendo desarrollados acerca de la investigación en este proyecto.

- **Impacto económico:** Dado el grado de madurez del proyecto, los mayores impactos económicos se esperaran a largo plazo. En un estado más avanzado, el prototipo puede llegar a evaluarse y comercializarse, lo que repercutiría en las empresas industriales dedicadas al desarrollo de tecnología. Siendo un robot destinado principalmente al sector de servicios o incluso con aplicaciones como robot social, también se apreciaría un efecto en estos dos sectores, al realizar ciertas tareas humanas: entiéndase el robot planchador o interacción humano-robot como robot social. En el corto plazo, el uso de la robótica blanda en el proyecto presenta una cierta ventaja económica pues los materiales blandos suelen tener coste reducido, y piezas de este material pueden ser fácilmente fabricadas o mecanizadas, frente a otro tipo de materiales.

- **Impacto social:** En un contexto social, la incorporación de partes blandas a un robot humanoide tiene un efecto inmediato. Por un lado, incrementa la aceptabilidad de los robots a la hora de introducirlos como elemento cotidiano. Sin embargo, es igual de importante el incremento en la seguridad al operar con este robot, tanto en aplicaciones del sector servicios como sociales.

1.4. Marco regulador

Dado el grado de madurez de este prototipo y su fin principalmente académico, su implementación no se ve afectada en esta etapa por ninguna legislación aplica-

ble. De igual manera tampoco necesita de un estudio en cuestiones relacionadas con propiedad intelectual. Si bien, podría formar parte del marco regulador el uso de los estándares CiA402 y CiA301 de CANopen en la organización de las librerías de comunicación. Estos estándares forman parte del Modelo OSI (Open System Interconnection) para los protocolos de la red de arquitectura en capas, registrado como ISO/IEC 7498-1.

1.5. Objetivos y cronograma

En el punto de partida de este Trabajo Fin Grado, el proyecto ya contaba con un diseño preliminar del prototipo de eslabón blando, así como con un estudio de su modelo teórico. Por ello, los objetivos del presente Trabajo se plantearon para continuar con la investigación en dicho prototipo. Es entonces, el objetivo principal, la colaboración y apoyo a lo largo de todas las etapas del proyecto hasta su puesta en marcha.

Para facilitar la descripción de los objetivos del presente Trabajo Fin de Grado, en la Figura 7 se muestra el esquema hardware para el funcionamiento de la plataforma. En este montaje se utilizan tres motores Maxon, con sus respectivos encoders y reductoras. De igual manera, cada motor está conectado a un driver iPOS. Adicionalmente, el conjunto necesitará un PC, una placa de conexión, y una tarjeta Peak CAN.

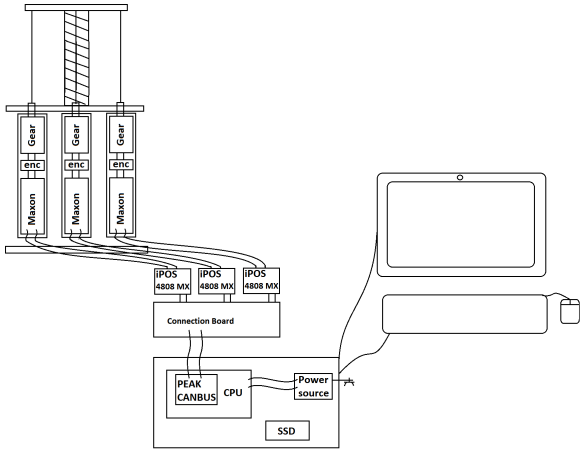

Figura 7: Esquema del conexionado Hardware de la plataforma.

Este trabajo puede ubicarse en la metodología propuesta en la memoria científico-técnica del proyecto HUMASoft. En ella se proponen diversos Paquetes de Trabajo (PT) que engloban tareas según su propósito. En la Figura 8 mostrada a continuación, se describe la funcionalidad principal de cada paquete de trabajo, así como su orden de ejecución.

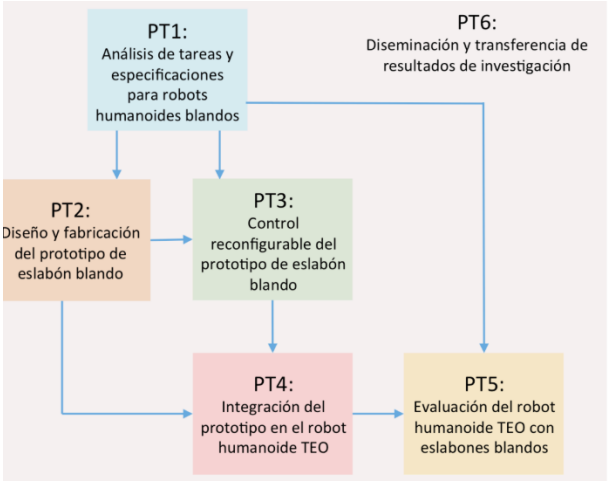

Figura 8: Paquetes de trabajo del proyecto y su interrelación

Los objetivos principales que se describirán a continuación forman parte principalmente del Paquete de Trabajo 2: Diseño y fabricación del prototipo de eslabón blando. Consisten fundamentalmente en trabajo práctico de desarrollo y montaje partiendo de un diseño ya realizado.

- **Montaje del prototipo.** Montaje de las distintas piezas para la obtención de una plataforma funcional en la que poder realizar pruebas de funcionamiento para validar el diseño. Estas piezas han de encargarse o fabricarse en las instalaciones de la universidad.
- **Desarrollo y ensayo de librerías de comunicación.** Desarrollo de funciones de comunicación complejas para el sistema de control del prototipo. Posteriormente, realización de pruebas y ensayos de distintos ejemplos del manual de los drivers para evaluar el correcto funcionamiento de las funciones desarrolladas. Para ello se montó un sencillo banco de pruebas.
- **Realización de la aplicación de usuario.** Desarrollo de un código fuente que, introducidas una posición y orientación de la parte superior del cuello, comande los tres motores de forma simultánea para llegar a ese punto.

- **Puesta en marcha del prototipo.** Montaje del sistema completo para el funcionamiento de la plataforma. Realización de las primeras pruebas y adquisición de resultados y conclusiones para la validación del prototipo.

El cronograma temporal de las tareas a realizar en este proyecto es el que se presenta en la Figura 9.

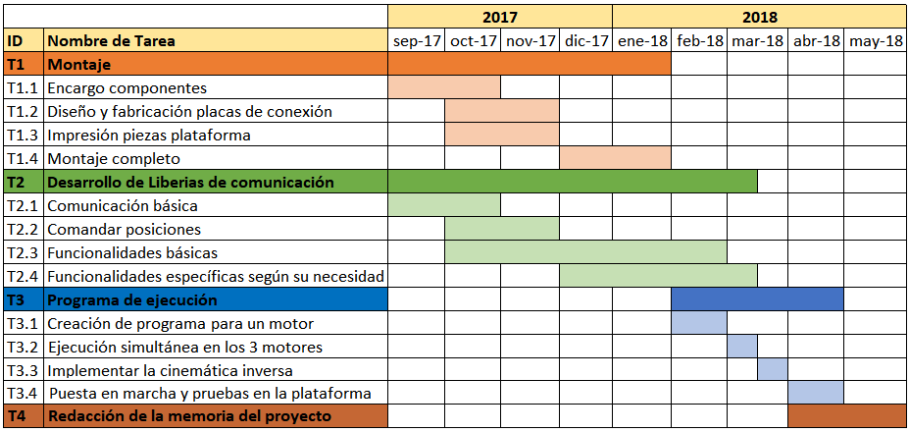

Figura 9: Cronograma de los objetivos iniciales.

1.6. Estructura del documento

El presente documento consta de siete capítulos diferenciados entre los que se encuentra el capítulo de *Introducción*. La estructura utilizada para los seis capítulos restantes que conforman el documento es la siguiente:

- En primer lugar, se encuentra el capítulo de *Diseño y montaje*. En este capítulo se presentan los diversos diseños propuestos para la plataforma. Primero se presentará el diseño inicial tal y como se concibió al inicio de este trabajo; pero también se describirán las diferentes versiones que se prototiparon posteriormente. Adicionalmente, encontramos un apartado dedicado al montaje del prototipo completo, incluyendo el ensamblaje de la plataforma y el conexionado hardware entre los distintos elementos del sistema.

- El siguiente capítulo está orientado a las librerías de comunicación. Dada la gran importancia de los protocolos CAN y CANopen en este proyecto, se comienza el capítulo con una recopilación de sus aspectos más relevantes

en términos de comunicación. Siguiendo este hilo, el apartado siguiente trata la comunicación específica para los drivers adquiridos para el proyecto. Finalmente, el capítulo termina con una descripción de las librerías de comunicación y de la organización de la estructura del código desarrollado.

- El capítulo *Aplicación de usuario para el control del cuello* describe el código desarrollado para el movimiento de la plataforma. Encontramos en la aplicación de usuario dos módulos distintos: El módulo de control de posición, que permite mover la plataforma a una posición determinada en lazo abierto; mientras que el Módulo de control por realimentación hace uso de control fraccionario en lazo realimentado.

- A continuación, se encuentra el capítulo que describe la puesta en marcha del prototipo. En este capítulo se encuentran descritas las diferentes pruebas efectuadas en el prototipo para su validación. Estas pruebas vienen acompañadas de los resultados y conclusiones obtenidos tras su realización.

- El capítulo 6, *Calificación del proyecto y presupuesto*, describe el desglose de los costes que genera la fabricación del prototipo. Se han identificado cuatro fuentes de gastos principales: la mano de obra, el software utilizado, el hardware utilizado, y material y herramientas. De igual manera, al final de este capítulo se ha efectuado el cómputo de los costes totales de la fabricación del prototipo.

- El último capítulo contiene las conclusiones alcanzadas tras el trabajo realizado en el proyecto. Adicionalmente, este capítulo consta de un apartado en el que se han propuesto trabajos futuros a realizar en el prototipo partiendo del estado actual del proyecto.

Adicionalmente, este documento contiene cuatro anexos. Tres de estos anexos corresponden a una descripción de los métodos de las principales clases de las librerías de comunicación desarrolladas.

2. Diseño y montaje

En este capítulo se presentarán los distintos modelos que se han evaluado hasta el momento para el prototipo de cuello robótico blando de HUMASoft.

En primer lugar, se detallará el diseño inicial y su funcionamiento tal y como se planteó al inicio de este Trabajo Fin de Grado. De forma adicional, se hará una descripción más detallada de las diferentes partes que conforman esta plataforma. Finalmente, este apartado concluye con una tabla de las características estimadas de la plataforma.

Adicionalmente, en este capítulo se describirán otros dos diseños. En primer lugar, se hablará de un diseño alternativo cuya diferencia principal es su método de transmisión. Para ello, la geometría de la plataforma entera ha sido modificada y las disimilitudes resultantes han sido igualmente identificadas. Se describirá de igual manera esta transmisión alternativa.

El último modelo descrito es el diseño final, ideado a partir de las dos versiones anteriores y el que finalmente ha sido ensamblado y utilizado físicamente en el proyecto.

Finalmente, existe un último apartado en el que se describirá el montaje de la plataforma siguiendo el diseño más actualizado, así como las modificaciones que se efectuaron respecto de este. De igual manera, se describe el montaje del prototipo completo, incluyendo el conexionado hardware necesario.

2.1. Diseño inicial del cuello

El proyecto HUMASoft de la Universidad Carlos III de Madrid consta previamente de un prototipo de cuello que es capaz de flectar emulando a un cuello humano. Este diseño se recoge en el Trabajo Fin de Máster de Luis Fernando Nagua Cuenca «Diseño y simulación de un prototipo de cuello robótico» [9]. En dicho trabajo, se detallan las consideraciones de la biomecánica que se tuvieron en cuenta para el diseño y funcionalidad del cuello. Igualmente, se hizo uso de investigaciones tales como «Development of a Low Motion-Noise Humanoid Neck: Statics Analysis and Experimental Validation»[10] para diseñar un modelo matemático lo más fiel posible al modelo real, de modo que se posibilite la cinemática directa e inversa en el cuello.

El prototipo de cuello se basa en la anteriormente mencionada robótica blanda. En este caso este concepto aparece en el eslabón central del mecanismo: un muelle metálico dirigido por tres cables. Esta configuración en manipuladores lleva el nombre de Mecanismo Paralelo Conducido por Cable (CDPM), donde típicamente se actúa sobre una columna central de forma que esta sea capaz de flectar en una o más direcciones. La dirección y fuerza del movimiento en un CDPM es siempre dependiente de la combinación de tensiones de los cables adheridos a la columna, pudiendo contraerse o relajarse para dirigirla.

El diseño se inspiró parcialmente en un proyecto en colaboración con el Centro Aeroespacial Alemán (DLR) [11], mostrado en la Figura 10, en el que se lograba una tarea de control mediante control fraccionario en la flexión de un bloque de silicona. El accionamiento se realiza mediante dos actuadores que recogen o liberan dos cables situados en extremos opuestos respecto del bloque central, y que permite una inclinación de dicho bloque en dos direcciones. Sin embargo, este modelo presenta diversas desventajas: En primer lugar, el diseño ideado por el DLR resultaba inconveniente debido a su gran envergadura, e igualmente sólo permitía la flexión del elemento blando en dos direcciones. El nuevo diseño propuesto para el proyecto HUMASoft solventa estos problemas proponiendo un diseño compacto, ligero, relativamente móvil y con una gran innovación respecto a los grados de libertad del dispositivo: añadiendo un tercer motor a una distancia equidistante de los otros dos, permite una flexión en los 360º alrededor del eje central del cuello. Este diseño queda reflejado en la Figura 11.

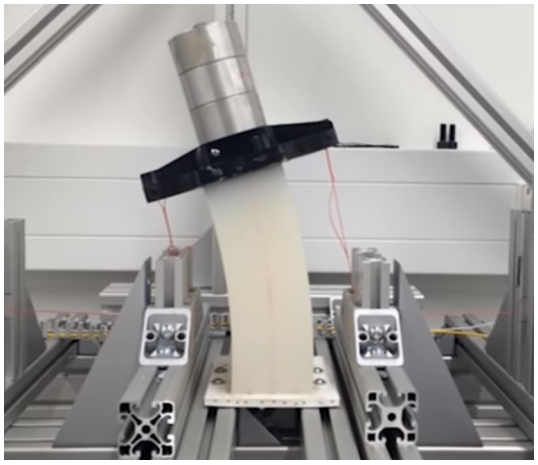

Figura 10: Sistema de robótica blanda con control fraccionario en el que se inspira el diseño inicial de la plataforma de HUMASoft.

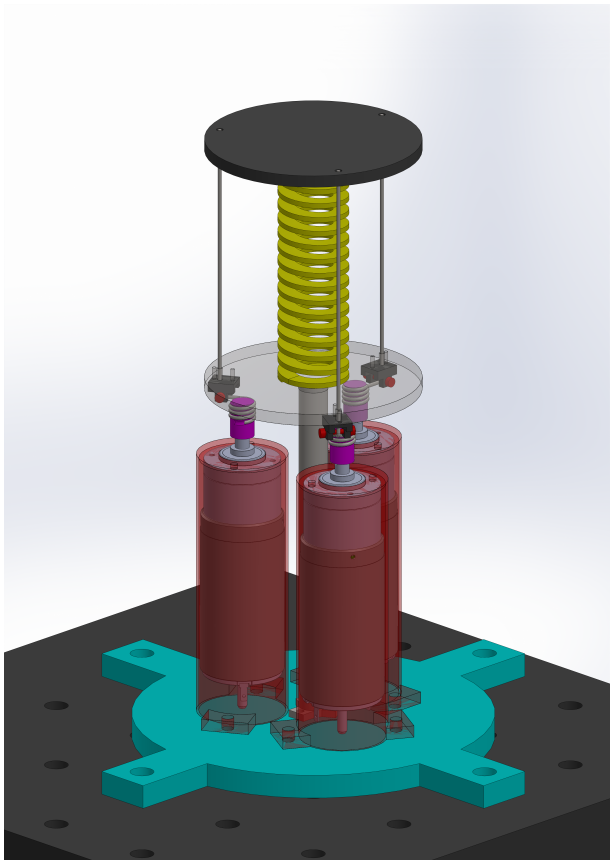

Figura 11: Diseño inicial del prototipo.

Partiendo de este diseño se pueden definir los parámetros que se utilizarán para comandar el movimiento del cuello: inclinación y orientación. Tal como se muestra en la Figura 12, la inclinación del cuello corresponde a la variación del ángulo de la normal de la parte superior del cuello respecto de su posición de reposo; mientras que la orientación es la dirección angular en la que se produce esta inclinación. Los rangos de estos parámetros vienen definidos a continuación, en la descripción del diseño del prototipo.

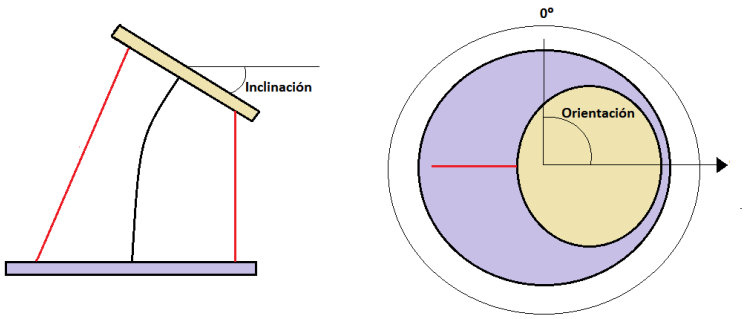

Figura 12: Inclinación y orientación del cuello robótico en una posición determinada.

Comparada con el diseño del DLR, en la propuesta inicial, mostrada en la Figura 11, el bloque de silicona ha sido reemplazado por un muelle como columna central. Este muelle ha de tener la resistencia necesaria como para soportar el peso de la placa y la cabeza del robot. Se estableció como requisito que el muelle fuera capaz de soportar el peso de 1 kg con una compresión mínima. Por esta razón, y recurriendo a las ecuaciones del teorema de Castigliano, se determinó que el diámetro del resorte (hélice) debía ser de 30 mm.

Como diseño CDPM, los cables son un elemento fundamental que también necesitaba ser objeto de estudio. Dado que el cuello está inspirado en un cuello humano, otra de sus características es un ángulo de flexión máximo de 40º, muy próximo a los 45º que flexiona un cuello humano. El estudio de fuerza se realizó bajo estas características, usando las ecuaciones que se recogen en el artículo de Stephen Timoshenko «Theory of elastic stability» [12]. Finalmente, los resultados indican que para un ángulo máximo de inclinación es necesaria una tensión de 60.392 N. Esta información es de gran relevancia a la hora de elegir los motores y reductoras que actuarán sobre el dispositivo.

A cada extremo del muelle, encontramos dos placas circulares de geometría muy similar. La placa que se sitúa sobre el muelle soportará la cabeza del robot y podrá llegar a alcanzar la inclinación máxima de flexión del muelle. Esta placa se llamará también base móvil. En la parte inferior del muelle se sitúa la base fija. Sobre ella está colocado el muelle y, sobre este, la placa móvil. Estas placas conforman un mecanismo en paralelo, unido por medio de tres cables atados a la base superior móvil y que atravesarán sin fijación la base fija. Dicha placa, está a su vez sostenida sobre un eje central anclado a la base de la plataforma.

Por otro lado, el funcionamiento de la plataforma consiste en la actuación mediante tres motores colocados de forma equidistante en la base de la plataforma. Estos motores, contenidos en una carcasa del mismo material que las placas, tensan o relajan los tres cables sujetos a la placa superior. De esta forma se logra una flexión posible en los 360º alrededor del eje del resorte, es decir, una flexión en tres dimensiones, obteniendo un manipulador con tres pseudogrados de libertad: inclinación en dos dimensiones y una ligera compresión. Esta compresión no ha llegado a ser explorada en su totalidad, pues se consideró que en calidad de cuello no era completamente relevante. Sin embargo, se mantiene abierta la posibilidad de utilizar ese grado de libertad para poder utilizar la investigación realizada en este proyecto en futuras utilidades.

En la placa fija, sobre la que está situada el muelle, se ha diseñado un sistema que permite la correcta recogida del cable conduciéndolo mecánicamente a un acople de 20 mm de diámetro donde terminará almacenado. Esto permite convertir el movimiento angular que efectúa el motor en tensión o relajación del cable. Esta transmisión por cabestrante resultó ser la más adecuada para obtener la fuerza necesaria para una inclinación de 40º. Adicionalmente, permitió el cumplimiento de los requisitos de durabilidad de la plataforma minimizando su desgaste.

2.1.1. Materiales y componentes

El mecanismo que se ha descrito anteriormente consta de distintos elementos. Estos se han seleccionado para cumplir con la correcta funcionalidad de la plataforma. A continuación, en la Figura 13 se mostrará un esquema de la plataforma con la lista de las diferentes partes que la conforman:
1. Base móvil
2. Base fija
3. Resorte
4. Polea del cabestrante
5. Cable
6. Acople del eje
7. Carcasa del motor
8. Reductora del motor
9. Motor DC
10. Superficie para la sujeción de la plataforma
11. Base de la plataforma

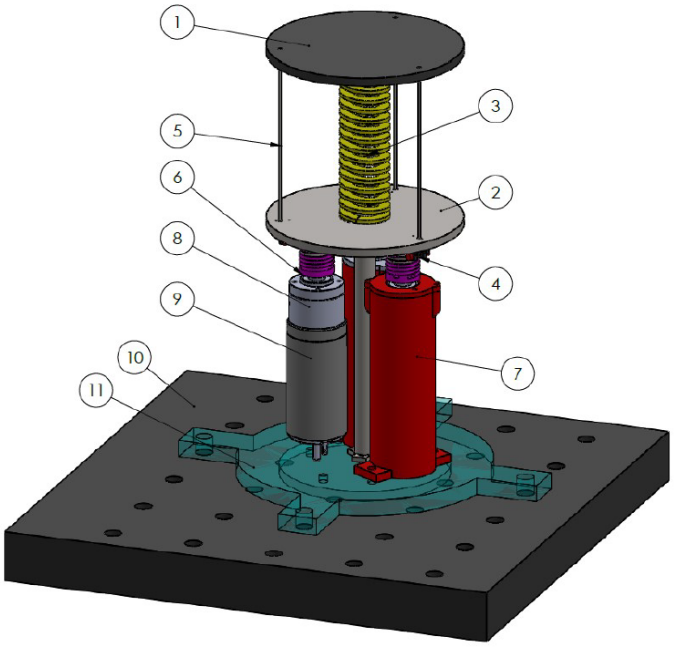

Figura 13: Esquema de las diferentes partes de la plataforma inicial.

Por lo tanto, para cumplir con las especificaciones requeridas siguiendo el diseño propuesto, se determinó la elección de las siguientes piezas y materiales.

En primer lugar, el resorte está compuesto a partir de Alambre de piano ASTM A228, habiendo dimensionado el diámetro de su hélice en 30 mm. Se han tenido en cuenta, para un alambre de diámetro de 3 mm de dicho material, las siguientes propiedades:

D(mm)	**d(mm)**	**Lo(m)**	δ**(m)**	**Na**	**G (GPa)**	**E (GPa)**
30	3	0.1	0.001	15	80	200

Tabla 1: Propiedades del Alambre de piano ASTM A228.

Siendo:

- D: El diámetro de la hélice del resorte.
- d: El diámetro del alambre del resorte.
- Lo: La longitud inicial del resorte.

- δ: La deformación por compresión del resorte.
- Na: El número de espiras del resorte.
- G: El módulo de elasticidad transversal del material del resorte.
- E: El módulo de elasticidad longitudinal del material del resorte.

Para los cables era necesario un material no elástico y de gran resistencia, pero tampoco podía ser muy grueso. Se eligió utilizar un hilo similar al hilo de pescar, dado que cumplía estas características. Sin embargo, esta decisión no es inamovible: diversos tipos de hilo o cable pueden ser utilizados en la plataforma con resultados muy similares.

Para la actuación se decidió utilizar motores que fueran compatibles con TEO, de forma que la implementación del cuello blando en el humanoide fuera lo más sencilla posible. El actuador seleccionado fue, por lo tanto, el Maxon DC 273762 RE 35 con escobillas de grafito y de 90 vatios. A continuación, se muestran las características básicas del motor (Tabla 2) así como una imagen (Figura 14) del mismo:

Figura 14: Motor Maxon DC 273762 RE 35 Graphite Brushes, 90 Watt

Motor Data			
	Values at nominal voltage		
1	Nominal voltage	V	48
2	No load speed	rpm	2110
3	No load current	mA	17.2
4	Nominal speed	rpm	1230
5	Nominal torque (max. continuous torque)	mNm	105
6	Nominal current (max. continuous current)	A	0.507
7	Stall torque	mNm	255
8	Stall current	A	1.19
9	Max. efficiency	%	77
	Characteristics		
10	Terminal resistance	Ω	40.2
11	Terminal inductance	mH	10.3
12	Torque constant	mNm/A	214
13	Speed constant	rpm/V	44.7
14	Speed / torque gradient	rpm/mNm	8.4
15	Mechanical time constant	ms	5.39
16	Rotor inertia	gcm²	61.2

Tabla 2: Extracto de la hoja de características del Motor Maxon DC 273762 RE 35 Graphite Brushes, 90 Watt.

Por otro lado, se calculó previamente la fuerza necesaria para contraer o expandir la longitud de los cables. Dado que se optó por utilizar los motores anteriores, la elección de la reductora se decidió con cautela, ya que era necesaria una reducción muy específica. Finalmente, se escogió la reductora GP 32 166155 de 32 mm diámetro cuya hoja de características se muestra a continuación en la Tabla 3.

Gearhead Data			
1	Reduction		3.7 : 1
2	Reduction absolute		$^{26}/_{7}$
3	Max. motor shaft diameter	mm	6
4	Number of stages		1
5	Max. continuous torque	Nm	0.75
6	Intermittently permissible torque at gear output	Nm	1.1
7	Max. efficiency	%	80
8	Weight	g	118
9	Average backlash no load	°	0.7
10	Mass inertia	gcm²	1.5
11	Gearhead length L1	mm	26.5

Tabla 3: Extracto de la hoja de características de la reductora GP32 166155.

Los encoders también se seleccionaron pensando en la compatibilidad con TEO. De hecho, se eligió el mismo modelo que se ha estado utilizando en el humanoide: Encoders CUI Inc AMT 203-V.

Por otro lado, las partes rígidas de la estructura del prototipo se han diseñado mediante un diseño CAD para ser posteriormente impresas en una impresora 3D. Estas partes corresponden a las tres placas, la base móvil, la base fija y la base la plataforma; la polea; y el acople del eje del motor.

Finalmente, habiendo finalizado la elección de estos componentes, se dio por completado el diseño inicial del primer prototipo de la plataforma. En las especificaciones de la plataforma, que se indican a continuación en la Tabla 4, se reflejan los requerimientos iniciales de la plataforma y las restricciones surgidas para su correcto funcionamiento.

Especificación técnica	**Valor**
Carga soportada	1 Kg
Grados de libertad	2
Voltaje	48 V DC
Peso aproximado	12,5 Kg (cálculo por programa CAD)
Dimensiones aproximadas	0,11 x 0,11 x 0,262 m
Rango de inclinación	0 - 40 º
Sensores	3x Encoders
Actuadores	3x Motor DC: Maxon RE 273762

Tabla 4: Especificaciones del prototipo inicial.

2.2. Diseño alternativo del cuello

Tras la realización del diseño inicial, se encargaron aquellos componentes que no se fabricaban por impresión 3D a fabricantes externos. Sin embargo, debido al largo periodo sin tener acceso a las reductoras encargadas, el grupo de HUMASoft se dispuso a buscar alternativas de reducción para el prototipo, de forma que estuviera operativo en los plazos estipulados. Se comenzó entonces el desarrollo de un segundo prototipo, modificado para reducir la longitud de los cables que orientan la plataforma entrelazándolos entre sí.

2.2.1. Sistema de reducción con entrelazamiento de cuerdas

Este sistema de transmisión consiste en dos cuerdas que se entrelazan y desenredan con el movimiento rotativo de un actuador atado a un extremo de estas cuerdas. El segundo extremo de las cuerdas está conectado a aquello de lo que se desea tirar. Con el giro del motor y el consecuente entrelazamiento de las cuerdas,

la distancia entre sus extremos se reduce, creando una tensión en el objeto móvil y provocando un movimiento. Con las cuerdas tensadas, un movimiento contrario al del entrelazamiento previo causaría que el lazo entre ambas cuerdas se desenrede, liberando la tensión de tracción en la cuerda.

Adicionalmente, el objeto debe estar en equilibrio soportando dos fuerzas contrarias. Por el extremo adherido a las cuerdas deberá sufrir constantemente un nivel de tensión mínimo, mientras que por el otro extremo deberá recibir igualmente una fuerza en sentido contrario a la tracción de las cuerdas. Esto permite que en el momento en el que las cuerdas estén desenrollándose y ejerzan menos tensión, el objeto sufra una fuerza que lo devuelva a su posición inicial. En nuestro prototipo, no sería necesario añadir una tensión adicional contraria a los motores, pues el carácter elástico del resorte cumple esta función.

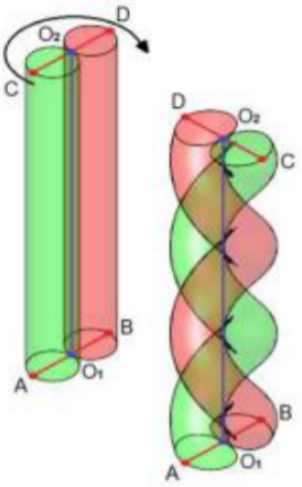

Figura 15: Transmisión por entrelazamiento de dos cuerdas [13].

2.2.2. Modificaciones realizadas en el prototipo y funcionamiento

Para poder implementar la transmisión por el entrelazamiento de cuerdas, se tuvieron que realizar diversas modificaciones en el prototipo de la plataforma. Durante el diseño de esta alternativa, siempre se mantuvo presente que no era parte del objetivo final al que quería llegar el proyecto HUMASoft, por lo que su diseño se ideó de forma que fuera fácilmente transformable a otro que incluyera las reductoras. El diseño alternativo planteado para usar este tipo de transmisión está representado en la Figura 16.

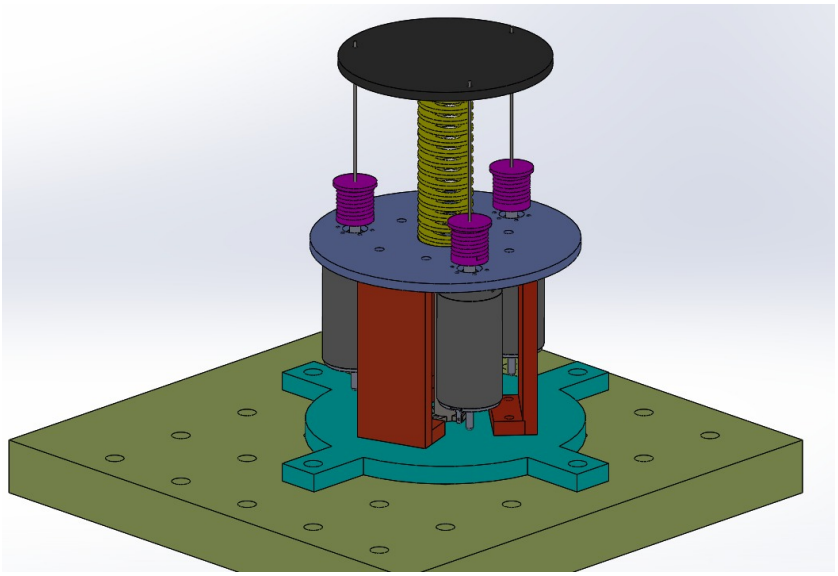

Figura 16: Diseño CAD del prototipo alternativo con transmisión por entrelazamiento de cuerdas.

Se puede observar que el cambio más importante en el modelo fue la fijación de los motores. En el modelo inicial estos están posados sobre la base de la plataforma, en el interior de una carcasa. Sin embargo, tanto en este diseño alternativo como en el diseño final que se describirá a continuación, los motores están suspendidos de la placa fija. Esto permite que los acoples que almacenan el cable se encuentren por encima de la base fija y directamente enfrentados con la placa móvil. Dos cuerdas por motor han de estar fijadas en cada uno de sus extremos a este acople y a la placa móvil respectivamente.

Un elemento que desaparece, tanto en este diseño como en el nuevo diseño con transmisión por medio de reductoras respecto del diseño inicial, es la carcasa de plástico que recubre los motores debido a que su introducción en las nuevas configuraciones resultaba inviable y su funcionalidad era meramente estética. En su lugar, aparecen tres soportes de plástico que unen directamente la base del montaje con la placa fija, en vez de estar conectadas a través de los motores, como se planteó en el prototipo inicial. Estos soportes aparecen en la Figura 16 representados en color rojo.

El funcionamiento de esta plataforma se basa en la transmisión por entrelazamiento de cuerdas que se ha mencionado anteriormente. Para alcanzar una posición

determinada de la base móvil, los motores han de moverse un cierto ángulo en un sentido u otro. Según el sentido de giro, los tres pares de cuerdas se entrelazarán o se desenredarán, y será la variación de la longitud de las dos cuerdas entrelazadas lo que incrementará o relajará la tensión que sufrirá la placa móvil en cada uno de los puntos de adherencia.

2.3. Diseño final del cuello

El diseño final que finalmente se llegó a construir, partió del diseño con transmisión por cuerdas entrelazadas. Como se ha comentado anteriormente, para el diseño de ambas alternativas se intentó que tuvieran muchos elementos comunes, de modo que montar uno a partir del otro resultara relativamente sencillo. Para ello, fue necesario modificar ciertas partes del modelo inicial. En la siguiente imagen, la Figura 17, se observa el modelo del prototipo final.

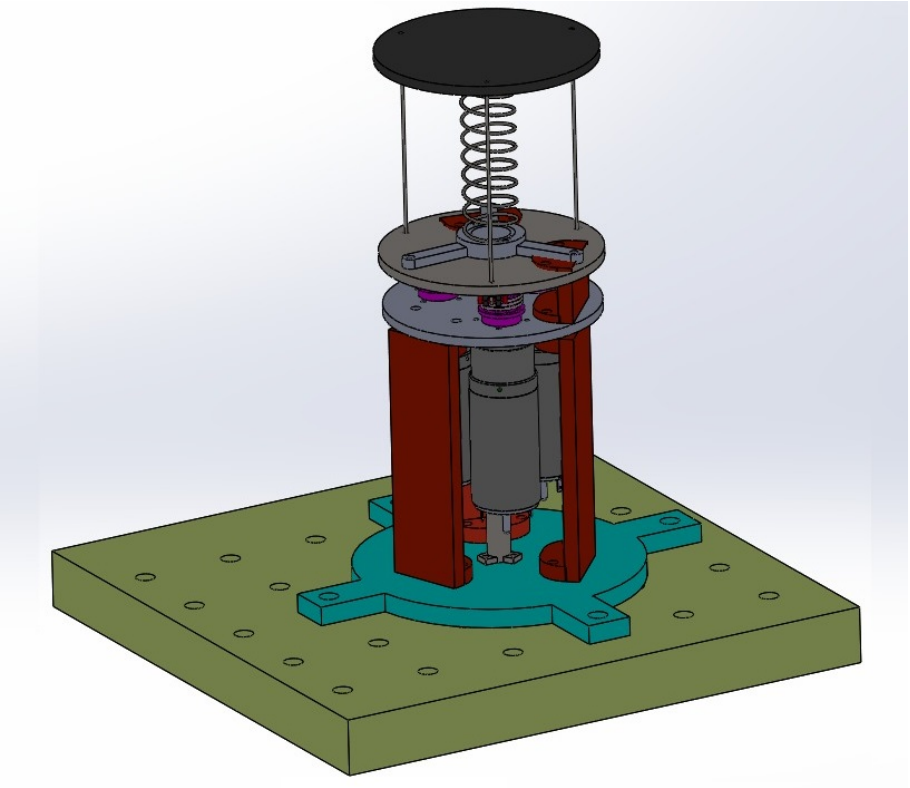

Figura 17: Diseño CAD del prototipo final utilizando las reductoras elegidas.

Este diseño hereda la placa del diseño anterior, siendo una pieza idéntica en ambas opciones. También pueden ser idénticos los soportes que sujetan dicha placa, aunque en las Figuras 16 y 17 aparezcan con una longitud distinta. De igual manera, tampoco se utilizan las carcasas que recubren los motores en el modelo inicial.

Sin embargo, partiendo del diseño inicial y el alternativo, fue necesario la introducción de ciertas modificaciones. En este caso, debido al uso de las reductoras situadas en el eje superior del motor, fueron estas las que se atornillaron a la base fija de la plataforma. Esto provocó que no hubiera suficiente espacio en el eje superior para situar el encoder de posición, que terminó siendo posicionado en el eje inferior del motor de menor diámetro. Finalmente, para poder fijar las reductoras con un orificio de 32 mm en el eje del motor de 35 mm, fue necesario el uso de un torno para reducir el tamaño del eje de los motores.

Tal y como sucede en el diseño alternativo, los acoples se encuentran en el extremo del superior del eje del motor. Esto los sitúa a una altura mayor respecto de la placa fija de la plataforma, ya que estos ejes atraviesan un orificio en esta. Si embargo, en este caso el acople sí cumple su función de recogida y liberación de cable, por lo que será necesario una placa adicional que contenga el sistema de recogida del cable por medio de la transmisión por cabestrante. Esta placa recibe el nombre de base intermedia y es donde está situado el resorte de la plataforma y el anteriormente mencionado sistema de transmisión, mientras que los tres cables que unen los motores con la base móvil la atraviesan por unos orificios. Para sostener esta placa sobre la placa fija se utilizan unos alzadores cuya geometría, exceptuando la altura, es idéntica a la de los soportes.

Dado que se pretende probar el prototipo, no solo con el resorte a modo de cuello blando, sino también con otros elementos de diferentes formas y materiales. En el prototipo final se introdujo una pieza que simplifica el cambio entre las distintas alternativas. Esta pieza consiste en un adaptador que hace de intermediario entre el cuello y las placas de la plataforma. De esta manera, se evita adherir el resorte, u otra pieza equivalente, directamente a la plataforma, teniendo que recurrir a una tediosa tarea de montaje y desmontaje para intercambiarlas. Dos adaptadores, adheridos a cada uno de los extremos del elemento blando permiten un fácil intercambio, pues simplemente habría que atornillar los adaptadores del eslabón deseado a la placa intermedia y a la placa móvil de la plataforma. A continuación, se muestra una imagen de los adaptadores que se han usado en la plataforma:

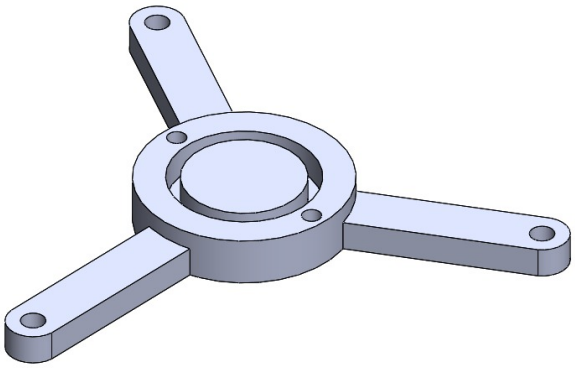

Figura 18: Diseño CAD del adaptador para los distintos cuellos.

Las distintas partes descritas se muestran a continuación, en la Figura 19, que representa un esquema de la plataforma:
1. Base móvil
2. Base fija
3. Resorte
4. Polea del cabestrante
5. Cable
6. Acople del eje
7. Soporte
8. Reductora del motor
9. Motor DC
10. Superficie para la sujeción de la plataforma
11. Base de la plataforma
12. Base intermedia
13. Alzador
14. Adaptador

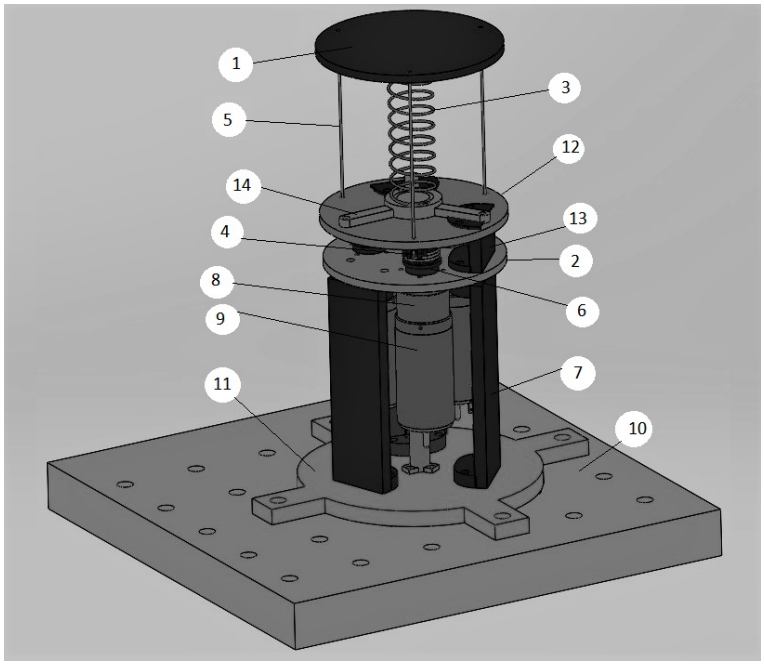

Figura 19: Esquema de las diferentes partes de la plataforma final.

2.4. Montaje del prototipo completo

Durante el montaje de la plataforma se utilizaron dos métodos para fijar las distintas piezas. El atornillado de los diferentes componentes resultó ser el más extendido. Sin embargo, existen tres elementos que no se fijaron de esta manera: Por un lado, al elemento blando se le adhirió un adaptador a cada extremo aplicando silicona con una pistola de material termo-fusible. El segundo conjunto de elementos que se aseguró sin utilizar tornillos son los acoples, que se introdujeron a presión en el eje del motor. Por último, los tres cables que tiran de la la base móvil se encuentran anudados a esta y sus respectivos acoples.

Finalmente, tras el ensamblaje de la plataforma completa, siguiendo los esquemas del diseño final mostrado en la Figura 19, finaliza la etapa principal de diseño y montaje. Sin embargo, tras las primeras pruebas con la plataforma, fueron necesarias unas ligeras modificaciones para optimizar su funcionamiento.

En primer lugar, se suprimió el eje central pues, con la presencia de los nuevos soportes, resultaba redundante y sin embargo, dificultaba el acceso para el atornillado de los motores y hacía más complejo el montaje y desmontaje de la

plataforma. También se hicieron más profundas las hendiduras de los acoples de los ejes de los motores, ya que se probó de forma experimental que los cables no se recogían correctamente con la consecuencia de que estos podían quedar destensados en pleno funcionamiento. La Figura 20 muestra el estado actual de la plataforma tras el largo proceso de prototipado y montaje. De igual manera, las especificaciones de la plataforma inicial han sido modificadas a las mostradas en la Tabla 5.

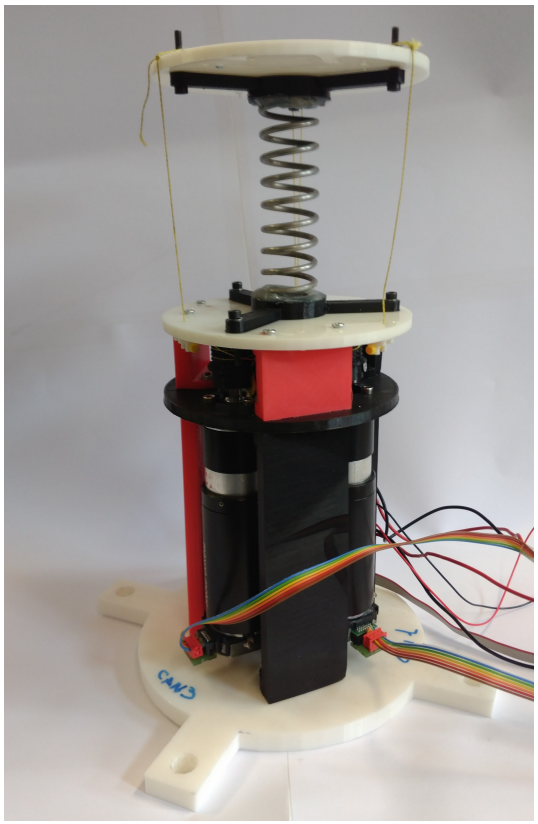

Figura 20: Prototipo ensamblado con el diseño final.

Especificación técnica	Valor
Carga soportada	1 Kg
Grados de libertad	2
Voltaje	30 - 48 V DC
Peso aproximado	1,9 Kg
Dimensiones aproximadas	0,22 x 0,22 x 0,29 m (incluyendo la base) 0,12 x 0,12 x 0,29 m (tronco)
Rango de inclinación	0 - 40 º
Sensores	3x Encoders: CUI Inc AMT 203-V
Actuadores	3x Motor DC: Maxon RE 273762
Reducción	3x Reductora: GP 32 166155

Tabla 5: Especificaciones del prototipo final.

Una vez ensamblada la plataforma, fue necesario el montaje del prototipo completo. Para ello, se realizó el conexionado hardware mostrado en la Figura 21. Este conexionado es equivalente al del esquema mostrado en el apartado de *Objetivos y cronograma*, en la Figura 7.

Como se observa en la figura, los elementos necesarios para el conexionado de todo el sistema son:

- **Plataforma.** Montada con los tres motores y sus respectivos encoders y reductoras.
- **Tres drivers iPOS4808 MX.** Conectados a la placa de conexionado que permitirá la conexión del driver con el resto de elementos. En cada placa se pueden conectar dos iPOS.
- **Dos placas de conexión.** Ambas placas están conectadas mediante la alimentación y el bus CAN. Son utilizadas para unir, por un lado la señal proveniente del encoder y la alimentación de su motor, y por otro recibe la alimentación de la fuente y la señal CANbus.
- **Fuente de alimentación.** Proveyendo con un voltaje que ha de ser superior a 30 V e inferior a 48 V, alimenta el sistema completo partiendo de las placas de conexión.
- **Tarjeta PeakCAN PCAN-minipci.** Situada en el interior de la caja del ordenador, permite la conexión del ordenador a la red CAN.
- **PC.** Envía los mensajes CAN para que cada driver regule el movimiento de su motor.

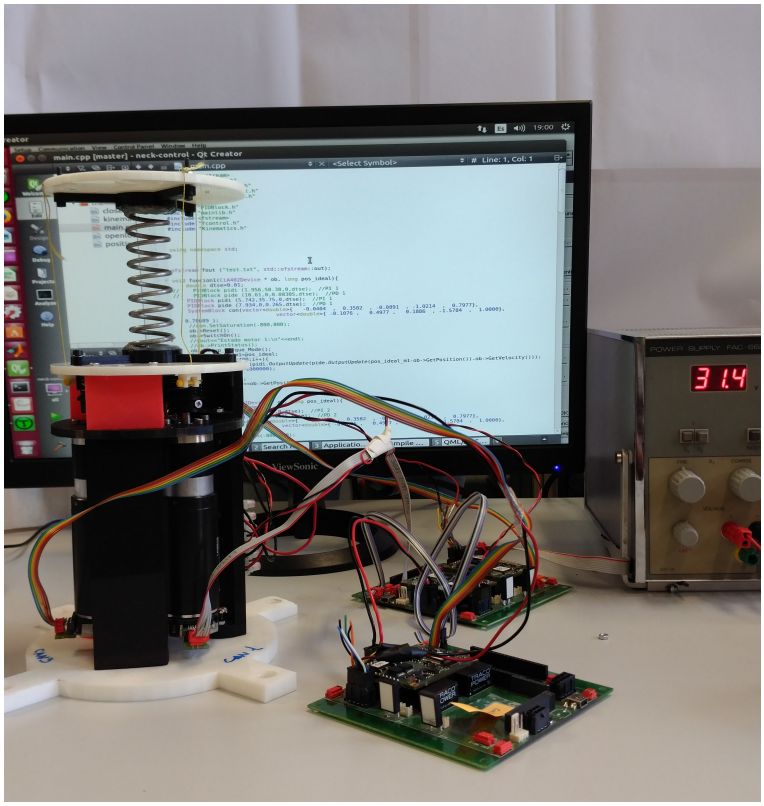

Figura 21: Conexionado Hardware de la plataforma.

Finalmente, con la plataforma ensamblada y correctamente conectada al hardware necesario para su funcionamiento, se da por finalizado el montaje del prototipo robótico de cuello blando.

3. Librerías de comunicación

En este capítulo se describirán las características de los protocolos de comunicación utilizados en las librerías desarrolladas. Dado que se han diseñado para un modelo de driver determinado, se indica cómo se realiza el control de dicho dispositivo y los modos de operación disponibles. Finalmente, se describe la organización general de las librerías de comunicación apoyándose en diagramas de clases.

3.1. La tecnología CAN y CANopen

Según el modelo OSI existen diferentes capas de CAN y CANopen [14]. Para cada protocolo, se dará su descripción detallada, así como la composición de sus diferentes tramas de comunicación. También se especificarán los distintos mensajes que pueden ser enviados y recibidos.

3.1.1. Protocolo CAN

El protocolo CAN fue introducido por Bosch en 1986 en la conferencia de la Society of Automotive Engineers en Detroit. Fue rápidamente traslado al sector de la automoción en el que el uso de buses de datos permitieron eliminar gran parte del cableado que hasta ese momento se realizaba de punto a punto. Bosch publicó nuevas versiones de las especificaciones CAN hasta que en 1993 se estandarizó con la publicación de la ISO 11898 de estándar CAN.

En la actualidad, un vehículo moderno está compuesto de diversos subsistemas con unidades electrónicas de control. Muchos de estos sistemas necesitan controlar actuadores o recibir respuesta de los sensores. La tecnología CAN no solo permite esto, sino que además es capaz de proporcionar funciones adicionales de seguridad o comodidad a un precio más reducido implementando únicamente software. No obstante, el uso protocolo CAN no se limita únicamente a los vehículos; en el campo de la robótica es de gran utilidad para realizar la comunicación entre motores y drivers o microcontroladores. [15]

3.1.1.1. Capa física: Arquitectura

CAN se comunica con un bus serie de múltiples maestros. La comunicación se efectúa entre dos o más Unidades de Control Electrónico o nodos que pueden ser

desde dispositivos muy sencillos hasta un ordenador.

Estos nodos se conectan con un bus de dos líneas trenzadas con una impedancia de 120 Ω denominados CAN high y CAN low. La especificación de CAN se divide en tres capas diferenciadas: la de aplicación, la de enlace de datos y la capa física. Esta última está fuera de la especificación ISO, ya que de esta forma cada diseñador puede modificar los niveles de voltaje o el medio de transmisión. No obstante, existe una diferenciación según el uso de alto o bajo voltaje, o el cableado. [19]

High speed CAN ISO 118982 Usa líneas de bus cerradas en cada extremo. El CAN de alta velocidad, implementado con dos cables, ofrece velocidad de transmisión desde los 40 Kbit/s hasta 1 Mbit/s. Adicionalmente, este tipo de especificación permite una desconexión simple de dispositivos, lo que la convierte en una especificación muy popular que se utiliza con frecuencia en automóviles o en otras aplicaciones industriales donde el bus transmite de un extremo a otro. Se puede observar la arquitectura de esta especificación en la Figura 22. [21]

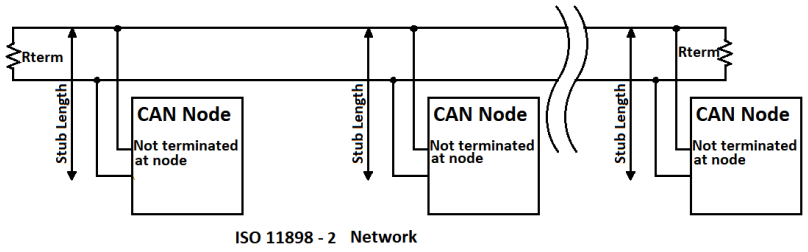

Figura 22: Red de High Speed CAN ISO 11898-2. [17]

Low speed CAN ISO 118983 También llamado tolerante a fallas debido a que posee transceptores que son capaces de detectar ciertos tipos de fallos. De forma similar al tipo anterior, también está implementado con dos cables, aunque en este caso la terminación de cada dispositivo es propia. Esta diferencia impide la utilización de dispositivos High Speed CAN y Low Speed Can simultáneamente en la misma red. Como su nombre indica, existe otra desemejanza respecto a la velocidad de transmisión: en esta especificación esta se encuentra entre los 40 Kbit/s y los 125 Kbit/s. El Low Speed CAN es comúnmente utilizado cuando hay un grupo de nodos que deben de estar interconectados. Esta especificación se muestra en la Figura 23.[22]

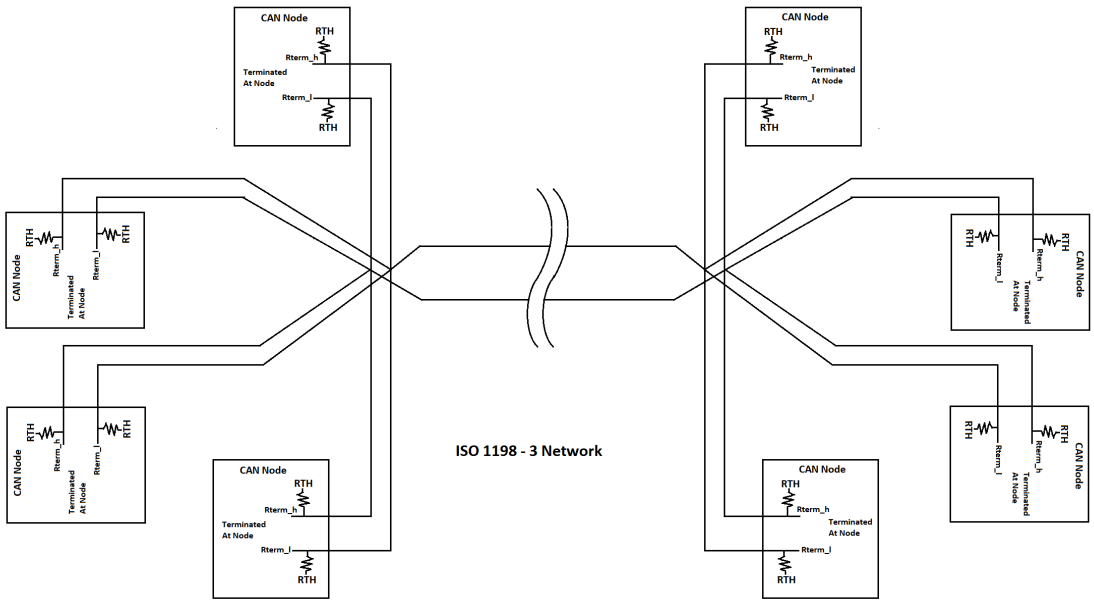

Figura 23: Red de Low Speed CAN ISO 11898-3. [17]

Es posible igualmente implementar un hardware CAN con únicamente un cable, aunque en este caso la velocidad de transmisión se reduce considerablemente, con una tasa máxima de 33KBit/s.[23]

La comunicación se realiza entre nodos CAN, que son capaces de enviar y recibir mensajes siempre y cuando no realicen las dos tareas simultáneamente. Existen distintos elementos que son fundamentales para la implementación de un nodo CAN.

Una Unidad Central de Procesamiento o un microprocesador capaz de interpretar los mensajes CAN que se reciben. Este procesador puede estar conectado a sensores, actuadores o dispositivos de control.

Adicionalmente, es necesario un controlador CAN que puede estar integrado

dentro del microcontrolador mencionado anteriormente. Recibe bit a bit la información serie hasta que el mensaje se haya transmitido completamente; en ese momento el controlador alerta al microprocesador de que hay un nuevo mensaje para procesar. De forma similar se realiza la tarea de envío de mensajes en el momento en el que el bus no está transmitiendo otra información y se encuentra libre.

Finalmente, es necesario un transceptor que sea capaz de convertir el flujo de información que proviene del bus CAN a un nivel con el que pueda operar el controlador CAN. En el apartado siguiente se explicará con más detalle en qué consisten estos niveles. [24]

3.1.1.2. Capa de enlace de datos

La Capa CAN de enlace de datos comprende dos protocolos: El CAN Clásico introducido originalmente en 1986, frente al CAN FD que no salió hasta el año 2012. Estos protocolos comparten ciertas características comunes. La estructura de trama de datos de ambos en CAN se puede observar en la Figura 24.

Dado que en ambos protocolos cada nodo puede pedir derechos de transmisión en cualquier momento, es necesario un método de arbitraje para evitar conflictos de transmisión. Para ello se usa el término «dominante» y «recesivo», equivalente respectivamente a un valor lógico de 0 o de 1. El estado natural del bus en espera es «recesivo», aunque si dos nodos desean enviar un mensaje o trama simultáneamente, se producirá lo que se denomina colisión. En este caso el bit dominante prevalece frente al recesivo y el nodo que pretendía transmitir este último pasa a modo pasivo para cederle el paso al nodo con bit dominante. Los nodos por lo tanto han de tener una transmisión de las tramas sincronizada.

El arbitraje se encuentra al principio de la trama en lo que se llama identificador del mensaje. Tras el proceso de arbitraje, finalmente sólo un nodo ha de quedar activo y con la capacidad de mandar mensajes. Los demás nodos, con menos prioridad, han de esperar a que el bus este libre para poder utilizarlo. De esta manera, todos los nodos que desean transmitir se ordenarán estableciendo un orden, en el que aquellos nodos con menor número de bits dominantes quedarán a la espera. [25]

Los mensajes que se transmiten entre nodos se llaman tramas existiendo hasta cuatro tipos de tramas:

- **Trama de datos.** Es la trama que se utiliza para la transmisión de información. Existen dos tipos de mensajes en esta trama. Por un lado, se encuentra el formato básico con 11 bits de arbitraje, frente al formato extendido que

cuenta hasta 29 de estos bits. Está compuesta de las siguientes partes:

Start of Frame (SOF): indica el comienzo de la transmisión de una trama.

Arbitration Field: identificador único que también contiene la prioridad del mensaje.

Remote Transmission Request (RTR): diferencia entre una trama remota de una trama de datos. Debe ser falso (0) para la primera, y verdadero (1) para la segunda.

Identifier Extension Bit (IDE): Ha de ser falso (0) para tramas con arbitraje estándar de 11 bits y verdadero para aquellas con arbitraje extendido.

Data Length Code (DLC): indica el número de bytes del dato transmitido, que puede ser de 0 a 8.

Data field: El mensaje de datos que se desea transmitir con una longitud entre 0 y 8 bytes.

Cyclic redundancy Check (CRC): código de revisión cíclica redundante de 15 bits; se delimita con otro bit adicional verdadero (1). Este conjunto se utiliza para la detección de errores en la transmisión.

ACK: Contiene un bit, denominado «hueco» (slot) utilizado como acuse de recibo, seguido de otro bit delimitador que ha de ser siempre verdadero (1). El hueco ACK es verdadero si es emitido por el transmisor, y falso si ha sido emitido por cualquier receptor.

En la Figura 24 se muestra la estructura de la Trama de Datos CAN en su formato básico y extendido.

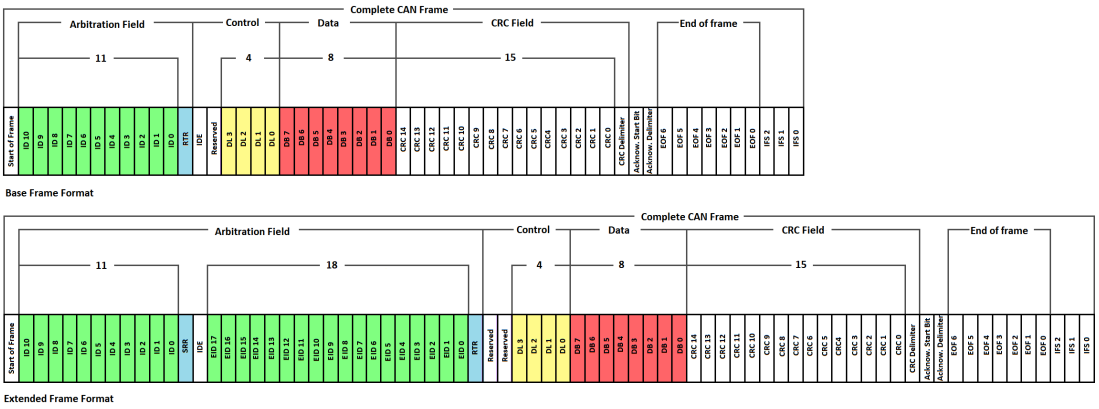

Figura 24: Estructura de la Trama de Datos en CAN.

- **Trama remota.** Puede ocurrir que un nodo requiera datos alojados en otro nodo. Para ello es necesario la transmisión de una trama remota que solicite dicha información. Se utiliza entonces el conjunto RTR descrito anteriormente que existe para este propósito: en una trama remota, el RTR es verdadero (1). Adicionalmente, las tramas de remotas se diferencian de las de datos en que las primeras carecen de campo de datos.

- **Trama de error.** Es transmitida cuando un nodo detecta que ha habido un error en la transmisión debido a un mensaje incorrecto. El error se propagará al resto de nodos de la red con el objetivo de eliminar el mensaje erróneo, algo que va en contra del estándar de CAN. La saturación del bus con continuas tramas de error se evita con un sistema de contadores de error gestionado por el controlador.

- **Trama de sobrecarga.** Se transmite cuando es necesario retardos adicionales entre tramas en el bus debido a la saturación de un nodo. Contrariamente a las tramas de datos y remotas, que se separan con tres bits verdaderos tras los cuales se espera una nueva trama, no existe separación entre las tramas de error o las tramas de sobrecarga. [25]

3.1.2. CANopen

CANopen es un sistema de comunicaciones superior y basado en el protocolo CAN. Se desarrolló originalmente para el control de sistemas de máquinas en movimiento, aunque hoy en día se utiliza en múltiples aplicaciones, tales como en equipo médico, en vehículos todoterreno, en electrónica marítima, en el campo ferroviario o en la inmótica. Fue en 1994 cuando se publicó la primera especificación CANopen, la CiA301, siendo parte del programa de investigación la Comunidad Europea ESPRIT. De ahí surgió el proyecto ASPIC cuyo objetivo fue el desarrollo una capa de aplicación de sencilla implementación, dedicada al control de sistemas embebidos. [26]

CANopen ofrece ciertas características ventajosas. En primer lugar, permite la transmisión de información en procesos de tiempo crítico. Por otro lado, proporciona estandarización en diversos factores: en la descripción del dispositivo a través del Diccionario de Objetos; en la monitorización del dispositivo, señalización de errores o coordinación de la red; en los servicios del sistema para la sincronización de operaciones; en las funciones de ayuda para configurar la identificación del dispositivo o la velocidad de transmisión; y en la asignación de patrones para identificadores de mensaje en configuraciones simples del sistema. [27]

Respecto a su estructura, consiste en una gestión del bus maestro/esclavo en la que existe un único maestro y uno o varios esclavos. El maestro ha de inicializar los esclavos, supervisar su funcionamiento y proporcionar información de su estado.

Como se ha indicado anteriormente, CANopen está por encima de las capas CAN y consiste en la implementación simultánea de diversos niveles de protocolo OSI que pueden estar cubiertos por CAN o por CANopen, como se puede observar en la Figura 25. Existen dos capas cubiertas por CAN: la capa física, que define las líneas, la tensión y velocidad utilizadas; y la capa de enlace de datos, que define la estructura de mensajes entre los nodos CAN. Ambas se han descrito ya en el apartado 3.1.1. Sin embargo, existen cinco capas superiores a las capas CAN, siendo estas: la capa de red (Network), que realiza el direccionamiento y enrutamiento; la capa de transporte (Transport), encargada de asegurar la fiabilidad extremo a extremo; la capa de sincronización (Session), que realiza precisamente tareas de sincronización; la capa de representación (Presentation), que representa los distintos datos y su estructura; y finalmente la capa de aplicación, que permite la configuración, transferencia y sincronización de dispositivos CANopen. [28]

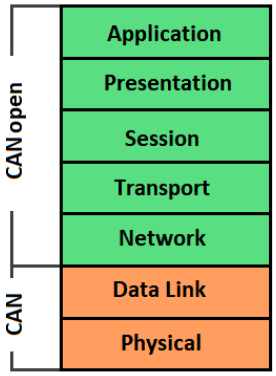

Figura 25: Las diferentes capas OSI de CAN y CANopen.

3.1.2.1. Diccionario de objetos

Uno de los elementos principales de la capa de Aplicación de CANopen es el diccionario de objetos. El diccionario de objetos es esencialmente una tabla que almacena la configuración y la información del proceso. Una entrada en el diccionario de objetos está definida por:

- **Índice:** la dirección de 16 bits del objeto en el diccionario, por lo tanto, es posible tener hasta 65536 índices.
- **Subíndice:** indica el número de elementos de la estructura de datos con un valor de 8 bits, puede haber hasta 256 subíndices.
- **Tipo/Tamaño del objeto:** indica el tipo simbólico del objeto en la entrada, como por ejemplo un array o simplemente una variable.
- **Nombre:** un string describiendo la entrada.
- **Tipo:** determina el tipo de dato de la variable o variables en caso de tratarse de un array.
- **Atributo:** da información acerca de los derechos de acceso de la entrada que puede ser read/write, read-only o write-only.

El estándar define ciertas direcciones o rangos de direcciones que han de contener determinados parámetros, como es el caso del índice 1008h con subíndice 00h que debe contener el nombre del dispositivo. De esta manera el maestro puede identificar con facilidad los dispositivos esclavos. [28]

Los diferentes dispositivos de una red CANopen pueden comunicarse a través del diccionario de objetos. La comunicación se realiza mediante dos tipos de mensajes que pueden acceder a este: Service Data Objects (SDOs) y Process Data Objects (PDOs).

3.1.2.2. Comunicaciones

Tal y como se muestra en la Figura 26, CANopen transmite únicamente mensajes con un estructura determinada: un ID de 11 bits, un bit de Remote Transmission Request (RTR), 4 bits del campo Data Length, y de 0 a 8 bytes de datos. El ID de 11 bits a su vez se divide en 4 bit de Function Code y 7 bits de CANopen node ID. Al usarse 7 bits para la identificación de los nodos, se permite la conexión de hasta un máximo de 127 dispositivos a una red CANopen. Debido a que existe una extensión en la trama de datos de CAN, se puede también extender el COB - ID del mensaje CANopen hasta un máximo de 29 bits [28]. Esta estructura se ve reflejada en la Figura 26.

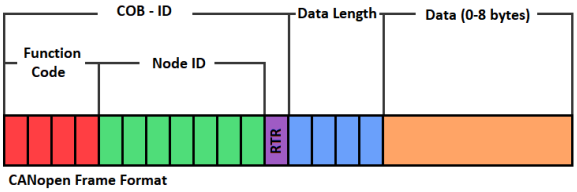

Figura 26: Estructura de la Trama de Datos en CAN.

El COB-ID sirve también para determinar el arbitraje en caso de ocurra una colisión, donde tendrá preferencia el mensaje con el ID más pequeño. Precisamente en la parte de Function Code, se determina el tipo de mensaje que se envía, y puede influir igualmente en la prioridad del mensaje completo.

Estos mensajes CANopen entre nodos pueden ser transmitidos en distintos modos de comunicación:

- **Maestro/Esclavo:** Un nodo CAN es considerado el maestro, mientras que el resto de nodos son los esclavos. El maestro ha de enviar o solicitar información de los nodos esclavos. Los mensajes NMT utilizan un modelo de comunicación maestro/esclavo.

- **Cliente/Servidor:** El servidor ha de dar soporte a los clientes. Este modo de comunicación se utiliza en los mensajes SDO. El cliente envía información (el índice y subíndice del diccionario de objectos) al servidor. El servidor ha de responder posteriormente con uno o varios mensajes SDOs con la información solicitada.

- **Productor/Consumidor:** El productor envía información al consumidor sin necesidad de que esta haya sido requerida en el modo push, mientras que en el modo pull el consumidor sí ha de solicitar la información del servidor. Este es el modelo utilizado en los protocolos de transmisión Heartbeat y Node Guarding. [29]

Existen distintos tipos de objetos que permiten la comunicación en CANopen que pueden tener un diferente tipo de prioridad, forma de comunicación y propósito. Estos mensajes se describirán a continuación:

- **Service Data Object (SDO):** Este tipo de objetos son utilizados para configurar y leer gran cantidad información no relevante del diccionario de objetos de un dispositivo remoto. Frente a los PDOs, son mensajes de poca prioridad. En los objetos SDO se utiliza una comunicación cliente/servidor en la que cada dispositivo cliente ha de implementar un servidor, denominado servidor SDO, que maneja las solicitudes de escritura y lectura, empezando siempre la transferencia en el cliente. Si un cliente SDO requiere información del servidor, enviará una solicitud SDO utilizando un CAN-ID de 600h + Node ID, mientras que la respuesta utilizará un CAN-ID de 580h + Node ID. El COB-ID de los distintos mensajes SDOs puede ser almacenado en el diccionario de objetos, pudiendo guardar hasta 128 en las direcciones 0x1200 a la 0x127F; de forma similar las conexiones del cliente SDO se puede configurar con las variables almacenadas entre 0x1280 a 0x12FF. El COB-ID de un mensaje SDO tiene la estructura de la Figura 27. Adicionalmente, existen dos modos de transferencias SDO: segmentada, cuando una información tiene una capacidad superior a un solo mensaje y ha de ser enviada en distinto segmentos; o expédita, cuando toda la información se ha transmitido en un único mensaje.

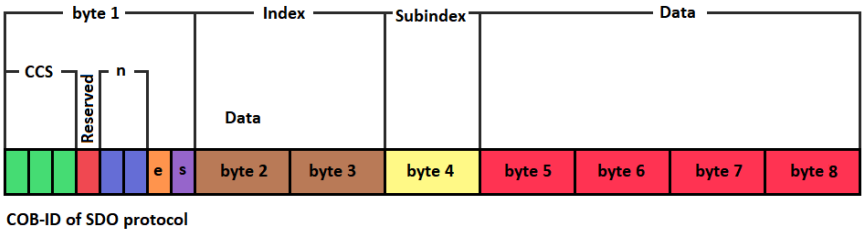

Figura 27: Estructura del COB-ID de un SDO

Client Command Specifier (CCS): Está compuesto de tres bits que indican el tipo de mensaje que se ha transferido (0 para leer, 1 para iniciar la lectura, 2 para iniciar la escritura, 3 para la escritura, 4 para abortar la transferencia, 5 para bloquear la lectura y 6 para bloquear la escritura).

n: Está compuesto de dos bits que indican el número de bytes de la parte de datos que no contienen información.

e: Si está a nivel alto indica una transferencia expédita mientras que a nivel bajo indica una transferencia en segmentos.

s: Si está a nivel alto indica que el tamaño del mensaje está indicado en la parte n del COB-ID, mientras que este bit a nivel bajo indica que este dato se encuentra en la parte de datos.

Índice: Corresponde al índice del diccionario de objetos de la información a la que se accede.

Subíndice: Corresponde al subíndice de la variable diccionario de objetos.

Data: Contiene la información que ha de ser escrita en caso de transferencia expédita (e=1), o el tamaño del dato que ha de ser escrito (si s=1 y e=0).

- **Process Data Object (PDO):** Estos objetos se utilizan para representar información del proceso en tiempo real. Las variables del proceso se almacenan en el diccionario de objetos donde pueden ser consultadas o modificadas. La comunicación para PDO es productor/consumidor donde los datos se

transmiten de un productor a varios consumidores. Debido a su naturaleza de mensajes de alta prioridad para tráfico en tiempo real, no se permite su fragmentación.
Existen dos tipos de mensajes PDOs: los TPDOs, que contienen información del proceso del nodo transmisor (productor); y los RPDOs, mensajes con información del proceso del nodo receptor (consumidor). La sección del diccionario de objetos reservada para los PDOs son los índices 1400h-1BFFh, permitiendo hasta 512 PDOs posibles.
Existen dos métodos de transmisión PDO: transmisiones síncronas y asíncronas. Así mismo, la transmisión asíncrona tiene tres formas de iniciar una comunicación PDO. En primer lugar, puede iniciarse con un evento, pero también puede ocurrir con un temporizador iniciándose después de un periodo de tiempo determinado, o con una solicitud remota con transmitida con una trama RTR enviada por otro dispositivo. Por otro lado, la transmisión síncrona se inicia usando una señal de protocolo SYNC (1005h en el diccionario de objetos). Esta señal se usa como temporizador global, y el tiempo de transmisión está determinado por la periodicidad del objeto SYNC.

- **Network Management Protocol (NMT):** Se utilizan para cambiar el estado de un esclavo y la administración y monitorización de los dispositivos de la red. El estado «Initialising» es el estado de inicio tras encender un dispositivo. En este estado se transmite un mensaje boot-up (700h + Node-ID) y posteriormente se pasa al estado «Pre-Operational». El dispositivo puede configurarse en este estado mediante SDOs, mensajes de emergencia, SYNC, time stamp y NMTs, pero no con PDOs. El siguiente estado, «Operational», el dispositivo ya se encuentra configurado y en funcionamiento. En este estado se pueden enviar y recibir PDOs. Finalmente, el último estado es Stopped en el que los objetos de comunicación están inactivos y no se pueden enviar ni PDOs ni SDOs, tan solo NMTs. En la siguiente figura, la Figura 28 se puede observar como es el cambio de estado que se produce en el nodo tras enviar los correspondientes mensjaes NMT. Sin embargo, respecto al esquema es importante aclarar que existe una diferencia entre el mensaje Reset_Node y Reset_Communication: mientras que en el primero se fijan los parámetros específicos del nodo a los de fábrica y se regresa al estado inicial (Application Reset y Communication Reset), con el segundo mensaje tan solo se realiza lo segundo (Communication Reset).[30] [29]

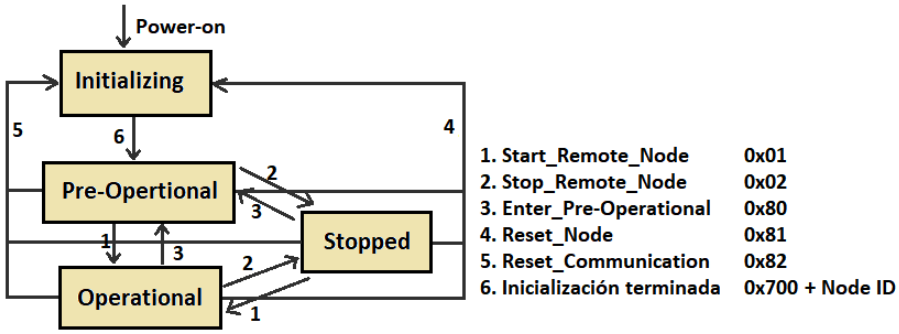

Figura 28: Máquina de estados de un nodo y los mensajes NMT de cambio de estado.

- **Mensajes Node Guarding y Heartbeat:** Los mensajes node/life guarding se utilizan para conocer si un nodo ha dejado de funcionar. Este tipo de mensajes son supervisados por el maestro. Si un nodo no responde, se asume que el nodo ha dejado de estar operativo y se actuará en consecuencia. Existen dos formas de implementación: Node Guarding y Heartbeat.

  Protocolo Heartbeat: el nodo esclavo envía mensajes periódicos con el function code 1110b y el node ID. La parte data contiene un byte que indica el estado. Si el mensaje tarda en llegar, el maestro puede tomar acciones.

  Protocolo Node Guarding: en este caso es el maestro el que envía mensajes a los nodos solicitando una respuesta con una trama RTR con el COB-ID 700h+Node ID. Los nodos han de responder con el mismo COB-ID seguido de un bit de toggle que se va alternando cada vez que se envía un mensaje de este tipo, seguido de otros 7 bits que indican el estado. El maestro espera respuesta, y si no la recibe o es incorrecta, actúa en consecuencia. [31] [32]

- **Mensajes de emergencia:** Se utilizan cuando ocurre un fallo en algún nodo. Se transmiten desde el nodo en cuestión a otros dispositivos de mayor prioridad. Un mensaje de emergencia tiene la estructura descrita en la Figura 29. En el Apéndice A se muestra la Tabla 14 con los posibles códigos de emergencia.

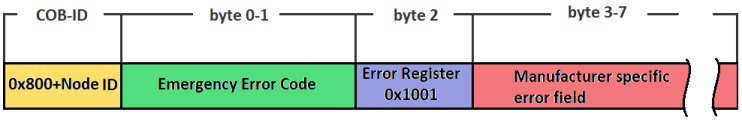

Figura 29: Estructura de un mensaje de emergencia.

- **Mensaje TimeStamp:** Este tipo de objetos representan el tiempo transcurrido desde el 1 de Enero de 1984 utilizando un campo de 6 bytes: 32 bits para representar los milisegundos transcurridos en un día y 16 bits para representar el número de días, sumando un total de 48 bits. Los objetos TimeStamp dan una referencia temporal común a los distintos dispositivos y son utilizados cuando es necesaria una sincronización de estos muy precisa.[30]

Descritos ya los objetos más generales que puede utilizarse en las comunicaciones CANopen, resulta de gran relevancia para el desarrollo de las librerías de comunicación conocer como ha de configurarse el COB-ID para cada tipo de mensaje. La siguiente imagen, la Figura 30, y en la Tabla 6, se muestra como está hecha la repartición de los distintos valores de COB-ID entre los objetos de comunicación de CANopen y el significado de cada uno de ellos. [33]

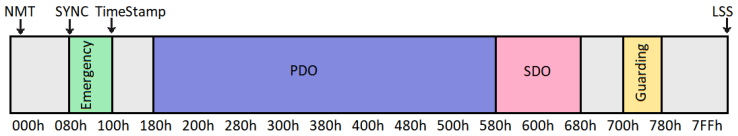

Figura 30: Espacio de identificación de CANopen.

Objeto de comunicación	**COB-ID**	**Nodos esclavos**
Control de nodo NMT	000h	Recepción solo
Sync	080h	Recepción solo
Emergencia	080h + NodeID	Transmisión
TimeStamp	100h	Recepción solo
PDO	180h + NodeID	1.Transmisión PDO
PDO	200h + NodeID	1.Recepción PDO
PDO	280h + NodeID	2.Transmisión PDO
PDO	300h + NodeID	2.Recepción PDO
PDO	380h + NodeID	3.Transmisión PDO
PDO	400h + NodeID	3.Recepción PDO
PDO	480h + NodeID	4.Transmisión PDO
PDO	500h + NodeID	4.Recepción PDO
SDO	580h + NodeID	Transmisión
SDO	600h + NodeID	Recepción
Monitorización de nodo NMT (node guarding/heartbeat)	700h + NodeID	Transmisión
LSS	7E4h	Transmisión
LSS	7E5h	Recepción

Tabla 6: Descripción de los diferentes COB-IDs en CANopen. [33]

Conocidos los protocolos de comunicación CAN y CANopen, se puede explicar el funcionamiento específico de los drivers que se han utilizado en la plataforma. Los diferentes tipos de mensajes descritos anteriormente forman parte de una comunicación más compleja que tan solo este dispositivo, o dispositivos muy similares, conocen. A continuación, se describe cómo se efectúa dicha comunicación.

3.2. Comunicación con los drivers iPOS

Para el controlar el prototipo de cuello se decidió utilizar tres drivers iPOS4808 MX de Technosoft, uno para cada motor. La documentación acerca de este driver se recoge en el Manual de Uso del fabricante [36].

A continuación se describirá como se pueden controlar los diferentes estados del driver, así como los modos de movimiento que se utilizarán en las librerías de comunicación.

3.2.1. Control del driver y de su estado

El bloque de control del dispositivo controla todas las funciones de este utilizando la máquina de estados Device Profile Drives and Motion Control. Esta máquina de estados es capaz de describir el estado del dispositivo y sus posibles secuencias de control. Los estados se pueden cambiar utilizando la Control Word o por medio de eventos internos, y estos, se pueden ver reflejados en la Status Word. En la Figura 31 se puede observar la descripción de dicha máquina de estados.

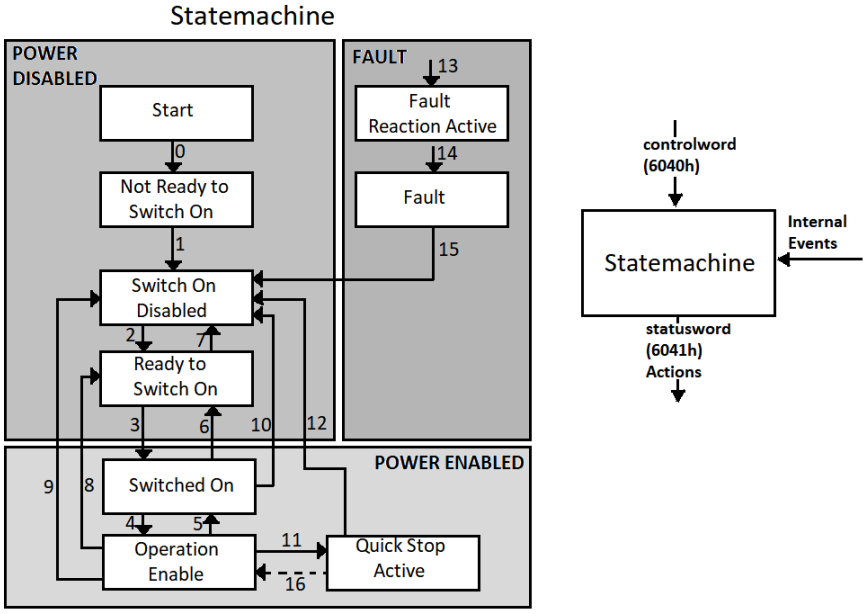

Figura 31: Máquina de estados de los dispositivos iPOS4808 MX.

De forma complementaria a la Figura 31, se han añadido la Tabla 7 que describe las acciones llevadas a cabo en las transiciones.

Transición	Evento	Acción
0	Transición automática tras el encendido o reseteo de la aplicación	Inicialización
1	Inicialización completada con éxito. El driver está operativo.	Ninguna

Continúa en la siguiente página

Tabla 7 – *Continuación de la página anterior*

Transición	Evento	Acción
2	Bit 1 Disable Voltage y Bit 2 Quick Stop, están activos en la Control Word (Shotdown command). Puede haber voltaje en el motor.	Ninguna
3	El bit 0 está también activo en la Control Word (Comando Switch On)	El voltaje está encendido (habilitado). El motor se mantendrá en la referencia de fuerza cero.
4	El bit 3 está también activo en la Control Word (Comando Enable Operation)	El movimiento está habilitado, dependiendo del modo activo, el motor aplicará fuerza y conservará su posición y velocidad actual como cero
5	El bit 0 se ha cancelado en la Control Word (Comando Disable Operation)	La función de movimiento está deshabilitada. El driver se ha detenido, utilizando la aceleración seleccionada en los perfiles de posición y velocidad. Dependiendo del modo de operación seleccionado antes del comando, el motor se mantendrá en la referencia de fuerza cero.
6	El bit 0 se ha cancelado en la Control Word (Comando Shutdown)	El voltaje se ha deshabilitado. El driver no controla en fuerza. El motor se puede girar libremente.
7	El bit 1 y el bit 2 se han cancelado en la Control Word (Comando Quick Stop o Disable Voltage)	Ninguna
8	El bit 0 se ha cancelado en la Control Word (Comando Shutdown)	El voltaje se ha deshabilitado. El driver no controla en fuerza. El motor se puede girar libremente.
9	El bit 0 se ha cancelado en la Control Word (Comando Disable Voltage)	El voltaje se ha deshabilitado. El driver no controla en fuerza. El motor se puede girar libremente.
10	El bit 1 y el bit 2 se han cancelado en la Control Word (Comando Quick Stop o Disable Voltage)	El voltaje se ha deshabilitado. El driver no controla en fuerza. El motor se puede girar libremente.

Continúa en la siguiente página

Tabla 7 – *Continuación de la página anterior*

Transición	Evento	Acción
11	El bit 2 se han cancelado en la Control Word (Comando Quick Stop)	EL driver se ha detenido con la deceleración quick-stop. El voltage continúa habilitado. Dependiendo del modo de operación seleccionado antes del comando Quick Stop, el motor puede conservar su posición, a velocidad cero o fuerza cero.
12	Quick Stop se ha completado o el bit 1 se ha cancelado en la Control Word (Comando Disable Voltage)	La salida se ha deshabilitado, el driver no tiene fuerza.
13	Señal de fallo	Ejecuta la rutina específica de tratamiento del fallo
14	El bit 7 activo en la Control Word	Algunos bits del Motion Error Register se resetean. Si todas las condiciones de error se han reseteado, el driver vuelve a Switch On Disable. Después de abandonar el estado de Fault, el bit Fault Reset de la Control Word ha de ser cancelado
15	El bit 2 está activo en la Control Word (Comando Enable Operation). Esta transición es posible si Quick-Stop-Option-Code es 5,6,7 o 8.	El driver abandona Quick Stop. La funcionalidad del driver se habilita

Tabla 7: Transición de estados del dispositivo

Las acciones que se efectúan en cada uno de los diferentes estados se describirán a continuación seguidos de las respectivas Status Word que plasman dicho estado.

- **Not ready to switch on** (xxxx xxxx x0xx 0000b) El driver realiza la inicialización básica tras el encendido. El funcionamiento del driver está deshabilitado. La transición a este estado es automática.
- **Switch on disabled** (xxxx xxxx x1xx 0000b) Las inicializaciones básicas del driver ya se han efectuado, la luz verde ha de estar encendida si no se

han detectado errores. Los parámetros del driver pueden modificarse, incluyendo la información del EEPROM completo. El voltaje del motor puede encenderse, pero no se pueden realizar funciones motrices aún. La transición a este estado es automática.

- **Ready to switch on** (xxxx xxxx x01x 0001b) El voltaje del motor puede encenderse, la mayor parte de parámetros pueden modificarse. No se pueden realizar funciones motrices aún.
- **Switched on** (xxxx xxxx x01x 0011b) Se debe aplicar voltaje al motor. El motor ha de mantenerse con una referencia de fuerza cero.
- **Operation enable** (xxxx xxxx x01x 0111b) Si no hay fallos presentes, las funciones motrices se habilitan. Si el operation mode estaba en control de posición cuando se comandó quick stop, el motor se mantiene en la posición. Si estaba en control de velocidad, el motor se mantiene a velocidad cero. Si estaba en modo torque externo, el motor se mantiene a fuerza cero. En este modo se pueden ejecutar comandos de movimiento.
- **Quick stop active** (xxxx xxxx x00x 0111b) El driver se ha detenido con la deceleración quick stop. El voltaje está habilitado. Si el operation mode está en control de posición, el motor se mantiene en la posición. Si está en control de velocidad, el motor se mantiene a velocidad cero. Si está en modo torque externo, el motor se mantiene a fuerza cero.
- **Fault reaction active** (xxxx xxxx x0xx 1111b) El driver realiza una reacción a falla en función de la condición de error.
- **Fault** (xxxx xxxx x0xx 1000b) El voltaje del motor se ha apagado. El driver se mantiene en condición de falla hasta que reciba un comando Reset Fault, tras el cual todos los bits del Motion Error Register se resetean y el driver abandona el estado.

3.2.2. Control del movimiento

El driver contiene diferentes modos a la hora de controlar un movimiento en un motor. En las librerías de comunicación desarrolladas se han explotado principalmente estos modos de funcionamiento: Position Profile Mode, donde el driver controla la posición del motor; o Velocity Profile Mode donde el driver realiza un control de velocidad. Estos serán descritos a continuación.

Position Profile Mode

En el perfil de posición además existen diferentes modos alternativos que pueden

cambiarse utilizando la control word. El primero, el perfil trapezoidal, tiene tres periodos. En el primero acelera de forma constante para luego mantenerse a velocidad constante. Luego desacelera de forma igualmente constante dibujando de esta forma un trapezoide en el perfil velocidad-tiempo. En función de cuál sea la posición a alcanzar (absoluta o relativa), el driver calcula como debe ser este perfil. En este modo se puede alterar la posición objetivo, el perfil de velocidad y el perfil de aceleración.

El segundo modo es el perfil de la s-curva. Su perfil de velocidad es muy similar al trapezoidal, excepto que este no tiene geometría angular, solo curva. El incremento de velocidad se produce debido a un periodo de incremento constante de la aceleración hasta alcanzar un máximo durante un cierto tiempo, para luego llegar a un decremento también constante. En este momento el motor se mueve a velocidad constante. Para parar el motor ocurre la secuencia inversa: se incrementa la deceleración, se mantiene durante un periodo de tiempo, para finalmente volver a llevar está deceleración a cero. La curva se puede parametrizar indicando la posición objetivo, el perfil de velocidad, el perfil de aceleración y el tiempo máximo para alcanzar la aceleración máxima.

En ambos modos, existen dos alternativas a la hora de actuar cuando se recibe una nueva posición antes haber alcanzado la anterior. Si se utiliza el perfil de movimiento discreto, el motor primero se detendrá y, transcurrido un pequeño periodo de tiempo, el motor volverá a moverse hacia la nueva posición. Sin embargo, utilizando el perfil de movimiento continuo, el motor cambiará inmediatamente su perfil de velocidad para alcanzar la nueva posición, aunque no haya completado la anterior.

Velocity Profile Mode
En este modo también se traza un perfil de velocidad con forma trapezoidal, aunque en este caso el descenso de velocidad no viene dado por una posición determinada, si no por la orden de Halt (Alto) en la Control Word. Si no se da esta orden una vez alcanzada la velocidad especificada, el motor se mantendrá girando constantemente a esta velocidad. Este modo se puede configurar indicando la velocidad objetivo y el perfil de aceleración.

Adicionalmente, el driver es capaz de efectuar tareas de control utilizando referencias de dispositivos externos, pudiendo controlar en posición, en velocidad y en fuerza, utilizando los modos External Position Mode, External Speed Mode, y External Torque Mode. Estos modos serán de gran utilidad a la hora de implementar acciones de control en la aplicación de usuario.

Las funciones necesarias para cambiar de modo y gestionar los perfiles de velocidad se han implantado en las librerías de comunicación diseñadas.

3.3. Organización de funciones de comunicación

Uno de los objetivos principales del proyecto es lograr la comunicación con el dispositivo y conseguir controlar la máquina de estados mencionada en el apartado anterior para alcanzar aquellos estados que permiten motricidad y control de fuerza. De ahí la importancia del desarrollo de una librería de comunicación que envíe los mensajes pertinentes para que el manejo por el usuario se efectúe de forma sencilla y eficiente.

Las librerías de comunicación están en constante desarrollo, aunque existe actualmente una versión sólida para realizar la correcta comunicación entre el usuario y la plataforma. La versión descrita a continuación es la correspondiente a abril de 2018.

Para el desarrollo de las librerías de comunicación se ha utilizado una organización basada en la norma CANopen, más concretamente en los estándares CiA402 y CiA301, de forma que la librería fuera compatible para cualquier dispositivo CANopen que posea un motor. Por esta razón, existe una organización jerárquica de las funciones realizadas. Es decir, hay funciones de alto nivel con gran complejidad que han de utilizar funciones más sencillas y menos complejas. Es un sistema en el que las clases más complejas son dependientes de otras funciones más básicas. Podemos distinguir cuatro niveles de abstracción en la librería según se muestra en la Figura 32:

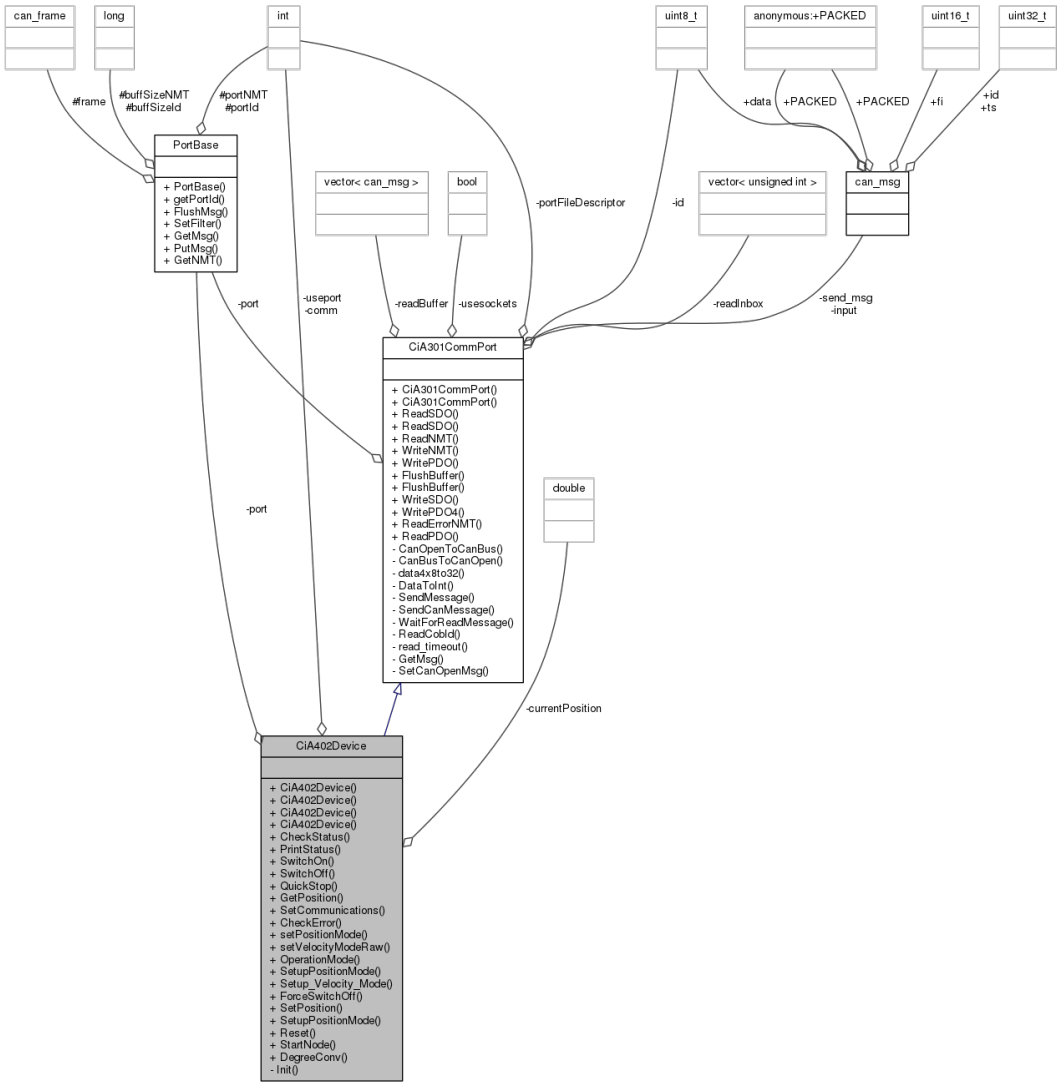

Figura 32: Diagrama de clases de la librería bajo los estándares CiA402 y CiA301 para el control del dispositivo.

En realidad, la figura anterior está resumida para mostrar principalmente las relaciones de la clase CiA402Device. Esta es la clase en la que se encuentran las funciones basadas en el estándar CiA402 que se utilizan directamente para comandar el movimiento de la plataforma. El uso del CiA402 en la librería permite estandarizar el comportamiento funcional de los drivers, así como introducir diversos modos de operación sujetos a parámetros de configuración. Consecuentemente, en los métodos de esta clase encontramos funciones con capacidad de realizar las

tareas seguidas en este estándar, como que el dispositivo avance por la máquina de estados por medio de la Control Word, o llevar a cabo las acciones permitidas en su estado actual. Es igualmente importante mencionar, que la clase CiA402Device tiene múltiples relaciones de uso con otras clases, sin embargo, la relación más importante es la de heredar de la clase CiA301CommPort.

Los métodos de la clase CiA301CommPort pueden utilizarse individualmente, pero por lo general sirven de apoyo para el desarrollo de métodos en su clase hija: la CiA402Device. Para la realización de muchas de las funcionalidades descritas en su clase hija es necesario, en otras cosas, métodos que permitan el envío y recepción de los distintos tipos de mensajes CANopen, implementados en la clase CiA301CommPort. Esta clase, al igual que la clase que CiA402Device, también sigue un estándar de CANopen; más concretamente el estándar CiA301. En él están descritos los tipos de datos que se pueden utilizar, las normas de codificación, al igual que el diccionario de objetos CANopen y los protocolos de comunicación.

Finalmente, existe un último conjunto de clases pertenecientes a librería que se podrían considerar como las funciones de más bajo nivel. En este conjunto de clases encontramos métodos que permiten operaciones con el puerto y el bus de datos. En primer lugar, tenemos la clase PortBase que aparece en el diagrama de la Figura 32 y permite funcionalidades como escribir en un puerto, limpiar mensajes residuales u obtener un mensaje determinado que se encuentre en el bus en ese momento. De ella derivan, 3 clases más que no se muestran en la figura del Diagrama de Clases del CiA402Device, ya que no tienen relaciones de uso directamente con esta. En la Figura 33 se describirá la relación de herencia existente entre la clase madre PortBase y las clases hijas CanBusPort, SocketCanPort y Test Port.

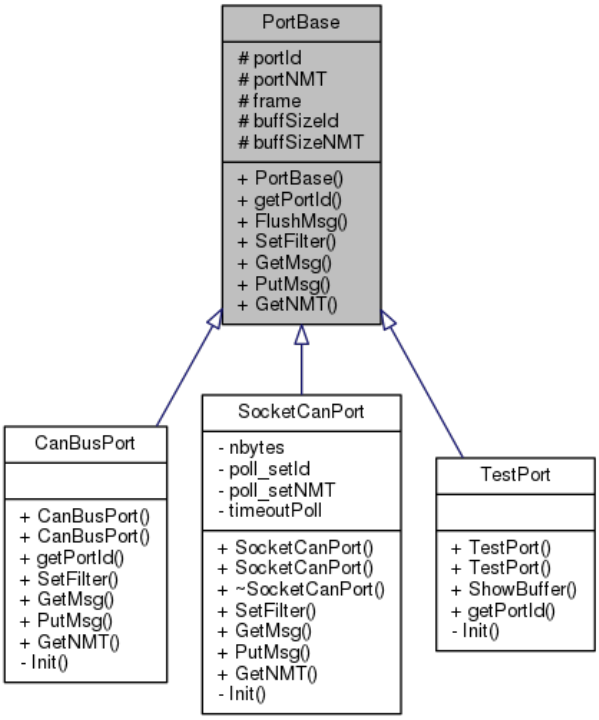

Figura 33: Diagrama de herencia desde la clase PortBase

La principal clase hija de PortBase, utilizada en la versión actual de las librerías de comunicación del iPOS, es SocketCanPort, ya que sus métodos serán los utilizados para inicializar el puerto en la aplicación de usuario. Las comunicaciones por medio de las funciones de esta clase no realizan una comunicación byte a byte, sino que logran una comunicación a través de una red por la que se envían y reciben datos. CambusPort se usa en menor medida pero, a diferencia de la clase anterior, esta clase está pensada para una comunicación chardev, es decir, byte a byte. Finalmente, la clase TestPort fue concebida como una clase para realizar pruebas referentes al puerto y no se utilizará en la aplicación de usuario ni en los métodos de otras clases relacionadas.

3.3.0.1. Descripción de los diversos métodos desarrollados en la librería de comunicación

En primer lugar, fue necesario manejar con facilidad los distintos objetos posibles que CANopen puede interpretar. Para ello se diseñó un archivo equivalente al diccionario de objetos, localizado en el archivo de cabecera ObjectDictionary.h.

De esta forma, se mantienen almacenados con un nombre propio cada uno de los objetos del diccionario de objetos de CANopen que se han necesitado. Cada objeto se escribe en un vector o en una cadena de vectores enteros sin signo dependiendo del número de bytes que forman el mensaje, formando un elemento de la cadena cada dos bytes. Los nombres con los que se han bautizado estos mensajes se han seleccionado intentando que coincidieran con aquel que tienen en el diccionario de objetos proporcionado por el manual de programación de CANopen del driver. En la Tabla 8 se muestra la relación entre los nombres y los objetos a los que corresponden:

Nombre	**Mensaje**	**Nombre**	**Mensaje**
controlword	6040h	velocityactvalue	6069h
statusword	6041h	velocityaddress	6069h
commreset	81h	target_position	00607Ah
fullreset	82h	position_demand	006062h
start	01h	torquemode	FBh
goreadytoswitchon	0006h	torque_type_extern	00201Dh
goswitchon	0007h	torque_online	0001h
goenable	000Fh	torque_target	00201Ch
goswitchondisabled	0000h	torque_max	26071h
run	001Fh	profile_acceleration	006083h
expedite	003Fh	quick_stop_deceleration	006085h
quickstop	0002h	motion_profile_type	6086h
OperationMode	006060h	profile_velocity	006081h
OperationModeDisplay	006061h	linear_ramp_trapezoidal	00h
positionmode	01h	target_velocity	0060FFh
velocitymode	03h	velocity_encoder_resolution_num	016094h
quick_stop_mode	605Ah		
stop_option_code	605Dh	velocity_encoder_resolution_den	026094h
checkerror	1002h		
positionaddress	006064h	gear_ratio	006091h

Tabla 8: Diccionario de objetos en el archivo «ObjectDictionary.h».

Este contenido es enviado en distintos mensajes de comunicación entre driver y ordenador. También fue de gran importancia desarrollar un entorno para cada tipo de mensaje, ya que, como se ha mencionado anteriormente, cada uno tiene características y funcionamiento distinto. Este entorno se ha realizado utilizando espacios de nombres, uno para cada tipo de mensaje. Los espacios de nombres han

sido los siguientes:

namespace sdo: donde se da el nombre de tx0 al número 580h, y rx0 al número 600h. Estos valores representan parte del COB-ID de un mensaje SDO de forma que se pueda identificar si se trata de un mensaje de solicitud del servidor, o de una respuesta.

namespace pdo: de forma similar al anterior, se han utilizado nombres para los COB-IDs de transmisión y recepción PDO. Estos nombres, tx0, rx0, tx1, rx1, tx4, rx4, corresponden respectivamente a los valores 180h, 200h, 280h, 300h, 380h, 400h.

namespace nmt: el último espacio de nombres utiliza el nombre de started para representar 01h. Tal y como se muestra en la máquina de estados de la Figura 28, este mensaje permite la transición del nodo del estado pre-operacional a operacional.

Estos espacios de nombres junto con la definición de las funciones más básicas para la transmisión de objetos CANopen están descritas en los archivos CiA301CommPort.h y CiA301CommPort.c. Una lista de estos métodos con su funcionalidad se recoge en el Anexo B de este documento.

Las funciones de más alto nivel son aquellas que son usadas directamente en el programa. Están declaradas en el archivo CiA402Device.h e implementadas en el archivo CiA402Device.cpp. Si bien es cierto que son poco numerosas, tienen la capacidad de poner en modo de operación el prototipo y provocar movimiento en el motor. Estas funciones se recogen en el Anexo C de este documento.

Finalmente, estas funciones se basan en las clases que permiten la escritura en el puerto PortBase y SocketCanPort. Se recogen en los archivos PortBase.cpp, PortBase.h, SocketCanPort.cpp y SocketCanPort.h de las librerías de comunicación. Su descripción se encuentra en el Anexo D de este documento.

Puede accederse a la versión más actualizada de las librerías de comunicación en los repositorios del usuario HUMASoft de la página web GitHub de alojamiento de proyectos.

4. Aplicación de usuario para el control del cuello

La aplicación de usuario del prototipo se desarrolló una vez que la plataforma final estaba ensamblada, y cuando las librerías de comunicación contaban ya con una base sólida y varias funciones complejas se habían programado. La aplicación final consiste en dos módulos cuyo propósito es utilizar la cinemática inversa de la plataforma para comandar una posición determinada.

La cinemática inversa fue investigada por Luis Fernando Nagua Cuenca en su Trabajo Fin de Máster «Diseño y simulación de un prototipo de cuello robótico» [9], cuyas ecuaciones se trasladaron a un programa desarrollado en Matlab. Este programa Matlab es capaz de calcular la longitud necesaria de la cuerda de cada motor en función del valor de inclinación y orientación proporcionados dentro de los valores permitidos: 0 - 40º para la inclinación y 0 - 360º para la orientación.

No obstante, la implementación de estas ecuaciones en un código fuente de C++ con las librerías de comunicación resultó problemática. Ciertas funciones utilizadas eran propias del entorno de Matlab y para su traslado a C++ habría que añadir dependencias a librerías, lo que reduciría la eficacia del código. Esto también repercutiría en la velocidad de ejecución y, dado que se desea efectuar control en tiempo real, no resulta conveniente. Por ello, se optó por utilizar una tabla de equivalencias para incluir la cinemática inversa en la aplicación. En esta tabla aparecen todas las combinaciones de orientación e inclinación posibles frente a la longitud que han de tener las tres cuerdas en esa posición. De esta forma, el programa tan solo deberá buscar los datos de orientación e inclinación y leer los datos de longitud que correspondan.

Sin embargo, para dirigir el motor a una posición determinada es necesario introducir el ángulo a alcanzar en grados. Para ello, se realizó el cálculo de la relación de transmisión del cabestrante, que consiste en lo siguiente. Tras obtener los tres valores de longitud, se calcula el incremento o decremento de esta respecto a los 10 cm de cable originales. Posteriormente, se utiliza el valor del radio del acople (1 cm), situado en el eje del motor donde se enrolla el cable, para obtener el ángulo de giro de este en radianes, que finalmente es convertido a grados. Este valor es el que se introduce en las funciones para generar movimiento en la plataforma y llevarla a la posición deseada.

4.1. Módulo de control de posición

El módulo de control de posición permite llevar a la plataforma a una posición determinada introduciendo los valores de inclinación y orientación deseados sin un controlador adicional al que poseen los propios drivers. La ejecución en dicho módulo se produce de la siguiente manera:

En primer lugar, se habilita el puerto socket para los tres motores a través del can0. Los valores de inclinación y orientación se almacenan en variables, con los que se obtienen los diferentes valores de longitud a través de la función de cinemática inversa. Tras calcular la relación de transmisión del cabestrante y obtener los grados de giro del motor, se procede a realizar un reseteo de los tres drivers y su posterior encendido hasta que estén operativos. Finalmente, en modo posición, se utilizan las funciones desarrolladas en la librería para comandar la posición a cada motor.

En este código, debido a que no se hace uso de un controlador, no es estrictamente necesario el uso de hilos de ejecución para que las operaciones realizadas en los distintos drivers se hagan de forma simultánea. En su lugar, las operaciones, como la del reseteo, se realiza de forma secuencial, es decir, se resetea el driver 1, posteriormente el 2 y finalmente el 3, para luego pasar a la siguiente operación.

A la hora de utilizar este módulo, tan solo hay que introducir los valores de inclinación y orientación deseados en las variables y ejecutar el programa.

4.2. Módulo de control por realimentación

El módulo de control por realimentación permite utilizar los valores de inclinación y orientación para mover la plataforma a una posición determinada utilizando control PID fraccionario. Tanto el código como el controlador han sido diseñado por miembros del grupo de investigación HUMASoft. Tal y como se muestra en la Figura 34, el controlador consiste en un bloque PI y PD fraccionarios que controlan el sistema en velocidad y fuerza.

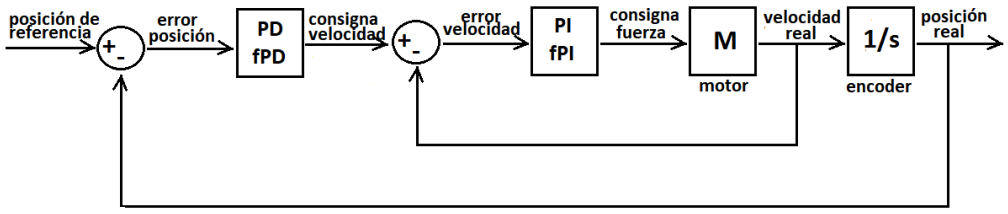

Figura 34: Diagrama de bloques del sistema de control fraccionario de la plataforma.

Las medidas de posición y velocidad real, obtenidas mediante las funciones GetPosition() y GetVelocity(), son calculadas por el driver en función de la información que recibe del encoder, de esta manera obtenemos un sistema retroalimentado que es capaz de regular el movimiento del robot.

En este caso, debido a la acción del controlador, es muy importante la ejecución simultánea del código para los tres motores. Por lo que la diferencia principal en la estructura de este código, respecto a la de control en lazo abierto, es el uso de tres hilos de ejecución simultáneos. Cada hilo de ejecución corresponde a un driver y su respectivo motor.

El programa, al igual que en el módulo anterior, abre el puerto socket y calcula, mediante la cinemática inversa y la relación de transmisión del cabestrante, el giro que ha de realizar cada motor a partir de las variables de orientación e inclinación. A partir de este momento la ejecución está separada en tres hilos de ejecución, uno para cada motor. En cada hilo, los procesos siguen la siguiente estructura: se resetea y enciende el driver; se calcula, mediante operaciones matemáticas y bloques para el PI y PD fraccionarios, la consigna de fuerza para ese motor; se controla en fuerza, en modo External Torque Mode, el resultado de la operación anterior; y finalmente se controla igualmente en fuerza una consigna de cero, tras una espera.

5. Puesta en marcha del prototipo

La puesta en marcha del prototipo comenzó en el momento en el que el control de la posición y de la velocidad estaban completamente operativos, la plataforma estaba ensamblada y la aplicación de usuario programada. Bien es cierto que diversas pruebas se llevaron a cabo antes de que todos los elementos anteriores estuvieran disponibles.

Las primeras pruebas se realizaron con un solo motor. El motor llevaba su encoder y su reductora atornillados, y tanto motor como encoder estaban conectados al driver mediante una placa de conexión. Los objetivos para este banco de pruebas se fueron marcando a medida que se desarrollaban las diferentes funciones de las librerías de comunicación. En primer lugar, se logró encender el driver correctamente y recibir mensajes indicando el estado en el que se encontraba. Posteriormente se hicieron pruebas para probar las funcionalidades del perfil de posición y el perfil de velocidad o, en otras palabras, comandar el robot a una posición o comandarle una velocidad de giro.

Una vez que esto que fue posible y todas las funcionalidades estaban terminadas y validadas, se probó el prototipo completo. A continuación, se procederá a relatar los resultados de los primeros experimentos.

5.1. Pruebas para validar el sistema

La primera prueba con la plataforma completa se llevó a cabo en lazo abierto. Consistió principalmente en comandar diversas posiciones consecutivas y ver cómo se comportaba el sistema. En las primeras posiciones de cada ciclo, la plataforma efectuaba el movimiento correctamente, sin embargo, acumulaba un error en las posiciones siguientes debido a que los cables perdían tensión. Se observó que este fenómeno se debía a que el hilo no se enrollaba correctamente en los acoples, por lo que se procedió a hacer ranuras más profundas en estos para evitar este problema. A continuación, en la Figura 35, se muestra una captura del momento en el que se realizó la primera prueba.

Figura 35: Captura del vídeo efectuado durante la primera prueba de la plataforma.

Las siguientes pruebas se efectuaron utilizando ambos módulos de la aplicación de usuario (el módulo de control de posición y el módulo de control por realimentación). En cada ciclo, se comandaron posiciones de orientación consecutivas con inclinación constante, de forma que el cuello diera una vuelta entera alrededor del eje. Se observa que en aquellas pruebas en las que se contaba con un controlador, los motores reajustaban la posición hasta encontrar el lugar adecuado, de lo que se concluyó que los controladores estaban actuando correctamente. Estas pruebas se efectuaron a 10 y 20 grados de inclinación. En la Figura 36 se muestra una escena del vídeo de esta prueba con una inclinación de 20 grados.

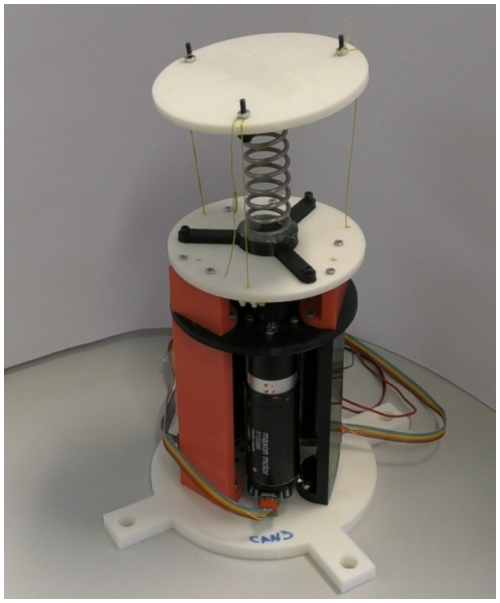

Figura 36: Captura del vídeo efectuado durante la segunda prueba de la plataforma con una inclinación de 20 grados.

A partir de este punto, se sacan nuevas conclusiones. En general se observa un comportamiento satisfactorio: Las posiciones alcanzadas sí corresponden con las previstas, aunque esta medida está basada en la observación y en los datos recogidos por el encoder. Para una medida más acertada, sería necesario introducir instrumentos de medida sobre la placa superior de la plataforma, como podría ser un acelerómetro, de forma que se pueda comprobar fielmente si la posición alcanzada es la correcta. Por otro lado, se observa que la plataforma en ocasiones efectúa el movimiento a la siguiente posición de forma no uniforme; es decir, realiza dos movimientos separados para alcanzar una posición como si los motores se estuvieran movimiento de forma no sincronizada. Serán necesarios reajustes en el código de forma que el tiempo de movimiento de los tres motores sea el mismo, aunque lleven velocidades distintas.

Adicionalmente se aprovechó esta segunda prueba para probar un nuevo eslabón blando para el cuello. Consiste en una pieza impresa con la impresora 3D, cuya geometría se ha inspirado en las vértebras humanas. Para su fabricación se han realizado pruebas de densidad y de patrones internos, obteniendo finalmente un eslabón hueco que es capaz de flectar lo requerido sin que sus paredes cedan. El eslabón se muestra en la Figura 37.

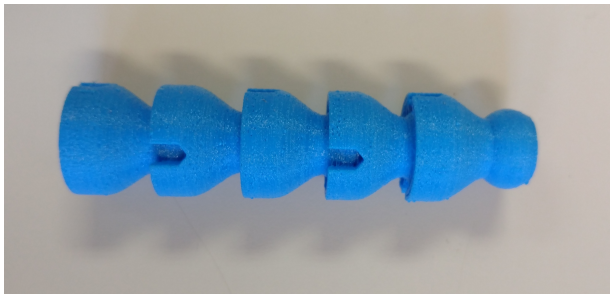

Figura 37: Eslabón alternativo de material de flexible

La prueba con este material se realizó de la misma manera que con el resorte. Se utilizó el código con control fraccionario, y se comandaron varias posiciones, con inclinación constante, incrementando la orientación hasta efectuar una vuelta entera. Hay que tener en cuenta que las ecuaciones de cinemática inversa no están adaptadas para este tipo de eslabón, pero el objetivo de la prueba era observar su comportamiento durante el funcionamiento de la plataforma. En la siguiente imagen, la Figura 38, se muestra una captura de esta prueba.

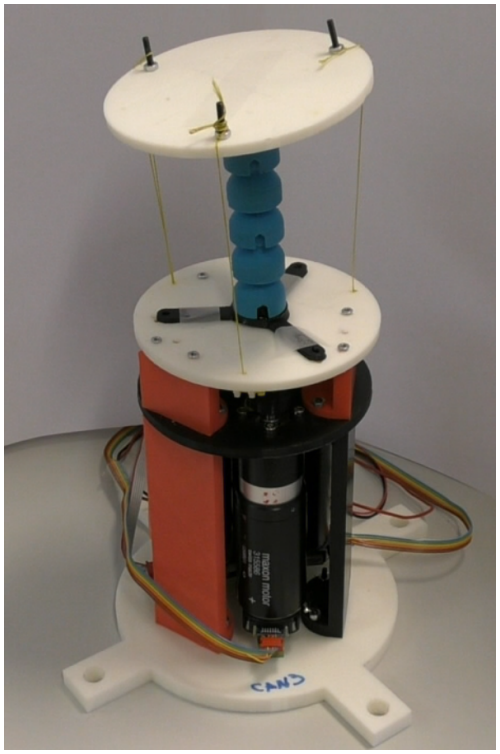

Figura 38: Prueba con eslabón alternativo de material de flexible con inclinación de 20 grados

En este caso, se obtienen resultados muy satisfactorios. La parte superior del cuello logró realizar la vuelta completa, y se podía observar que el material llegaba a flexionarse sin que sus paredes cedieran o se doblaran. No obstante, por falta de captura de datos no se puede asegurar que las posiciones fueran correctas, aunque se sospecha, debido al uso de una cinemática inversa inadecuada, que existe un error entre las posiciones esperadas y las reales. En conclusión, estos resultados dan el visto bueno a futuras investigaciones partiendo de este eslabón.

6. Calificación del proyecto y presupuesto

El desarrollo del presente proyecto ha incurrido en ciertos costes. En el capítulo actual se ha realizado su recapitulación y se ha calculado finalmente el importe total del proyecto. Para ello, se han tenido en cuenta principalmente tres fuentes de gastos: En primer lugar, la mano de obra de un ingeniero junior; también se han tenido en cuenta los gastos de software y hardware; y finalmente, los costes relativos a los materiales y herramientas utilizados.

Para el cálculo del coste de la mano de obra se parte de que para el actual proyecto tan solo es necesario un trabajador. Este es catalogado como un ingeniero junior pudiendo llegar a cobrar aproximadamente 15 euros por hora. También es importante determinar el número total de horas que este trabajador ha empleado en el proyecto, llegando a la conclusión de que este valor ronda las 300 horas. Con estos datos se pueden calcular los coste totales de la mano de obra tal y como se muestra en la Tabla 9.

Mano de obra			
Concepto	Nº Horas	Coste hora	Coste imputable
Ingeniero Junior	300h	15 €/h	4.500 €
Coste Total		Coste hora	4.500 €

Tabla 9: Costes totales de la mano de obra del proyecto

A continuación, se describirán el coste relacionado con el software utilizado. En este caso solamente se incurrirá en costes de depreciación (Cdep) del software, que dependerán del precio del producto (P), de tiempo de uso (Tu) y del tiempo de mantenimiento (Tm) de esa versión por el fabricante. Se utilizará para ello la siguiente ecuación:

$$Cdep = P * Tu/Tm$$

Los costes relacionados con el software se muestran en la Tabla 10

Software					
Concepto	Cantidad	Precio	Dedicación	Tiempo depreciación	Coste imputable
Ubuntu	1	0 €	9 meses	36 meses	0 €
Qt creator	1	0 €	9 meses	24 meses	0 €
EasyMotion Studio (Technosoft)	1	295 €	1 mes	36 meses	8,19 €
Coste Total					8,19 €

Tabla 10: Costes totales del software utilizado en el proyecto

Los costes de hardware también deben computarse en el cálculo total. Por un lado, tenemos maquinaria que tan solo incurre en coste de depreciación, ya que se podrá usar en otras aplicaciones al terminar el proyecto; mientras que también hay elementos que forman parte del prototipo en sí y su coste equivale a su precio. Para la depreciación del hardware y herramientas se utilizará una ecuación muy similar a la de los costes de depreciación del software: en este caso, en vez de hablar de tiempo de mantenimiento, se utilizará el concepto de tiempo de vida útil (Tvu):

$$Cdep = P * Tu/Tvu$$

Los costes relacionados con el hardware se muestran en la Tabla 11.

Hardware					
Concepto	Cantidad	Precio	Dedicación	Tiempo depreciación	Coste imputable
Ordenador	1	279 €	9 meses	60 meses	41,85 €
Ratón	1	9,99 €	9 meses	60 meses	1,50 €
Teclado	1	14,99 €	9 meses	60 meses	2,25 €
iPOS4808 MX	3	270 €			810 €
PCAN-miniPCI	1	220 €			220 €
Placa de conexión	2	60 €			120 €
Encoder CUI Inc AMT 203-V	3	39,38 €			118,14 €
Motres Maxon DC 273762 RE35	3	268,91 €			806,73 €
Reductoras	3	124,04 €			372,12 €
Coste Total					2.492,59 €

Tabla 11: Costes totales del hardware utilizado en el proyecto

Se incluirán también los costes de material y herramientas, que consisten en el resorte y el material plástico utilizado en la impresora 3D. Se ha calculado que, para la impresión de todas las piezas necesarias, no se ha necesitado más de una bobina de material, de 1 kg de peso. También se podrían incluir en este apartado

el coste de depreciación del torno o los costes de tornillos y tuercas, pero frente al resto de componentes resultan despreciables. Por lo tanto, el importe total de los gastos en material es el que se refleja en la Tabla 12.

Material y herramientas					
Concepto	Cantidad	Precio	Dedicación	Tiempo depreciación	Coste imputable
Resorte	1	2,79 €			2,79 €
Bobina para impresora 3D - 1kg	1	12,12 €			12,12 €
Impresora 3D : ZMorph 2.0 SX	1	2.219,01 €	2 meses	60 meses	73,97 €
Coste Total					88,88 €

Tabla 12: Costes totales de material y herramientas del proyecto

Finalmente, se hace uso de todos los cálculos anteriores para hallar el importe total del proyecto, que aparece reflejado en la Tabla 13.

Coste Total	
Conceto	Coste Total
Mano de obra	4.500 €
Software	8,19 €
Hardware	2.492,59 €
Material	88,88 €
Presupuesto proyecto	7.089,66 €

Tabla 13: Coste total del proyecto

7. Conclusiones

El objetivo principal de este Trabajo Fin de Grado consiste en la puesta en marcha de una plataforma robótica con un eslabón blando a modo de cuello. Se partía de un diseño basado en un mecanismo CDPM cuya columna central es un resorte con unas características determinadas. De igual manera, también se disponía del modelo teórico del accionamiento de este mecanismo CDPM por medio de tres cables equidistantes. Partiendo de esta base, los puntos requeridos y logrados en este proyecto han sido los siguientes:

- En primer lugar se efectuó el montaje de la plataforma siguiendo los últimos diseños. Para ello, fue necesario realizar la impresión de ciertas piezas de la plataforma utilizando el equipo del Departamento de Ingeniería de Sistemas y Automática de la universidad; mientras que otros componentes se adquirieron a proveedores externos. Se hizo uso de tornillos o silicona para la fijación de cada una de las piezas.

- En la actualidad las librerías de comunicación continúan en desarrollo. No obstante, existe ya una versión sólida que atiende a todas las necesidades que han surgido para la puesta en marcha del prototipo. Debido a que se realizaron en el entorno de C++ y su función es la de facilitar la comunicación entre dispositivos CANopen, para su construcción fue necesario un cierto nivel de conocimiento del protocolo CANopen y de programación C++. Finalmente, el resultado es que diversas funciones de comunicación, que se han organizado según el protocolo OSI de CAN y CANopen, fueron codificadas en estas librerías.

- Partiendo de las librerías de comunicación ya diseñadas, se ha desarrollado una aplicación cuyo propósito es poner en funcionamiento el prototipo. Es posible mover la plataforma a la posición deseada introduciendo los datos devueltos por las ecuaciones de cinemática inversa, que también han sido implementadas en el código. Esta aplicación consta de dos módulos: uno de ellos realiza el movimiento de la plataforma en lazo abierto; mientras que el segundo tiene adicionalmente un controlador fraccionario. Esta aplicación está diseñada en C++ y para el funcionamiento de cualquiera de los dos módulos tan solo hay que introducir en él el valor de la orientación e inclinación de la posición deseada y ejecutarlo.

- Para validar el prototipo fue necesaria la realización de ciertas pruebas para observar si los resultados eran satisfactorios. Para ello se efectuó el montaje de todo el sistema hardware. También debía de encontrarse la plataforma

completamente ensamblada y la aplicación de usuario operativa. Estas pruebas finales corresponden a la puesta en marcha del prototipo.

Adicionalmente, se han realizado las siguientes tareas que no se encontraban entre los objetivos principales del Trabajo:

- Debido a que las reductoras que se encargaron tardaron en llegar más de lo previsto, se idearon alternativas de reducción para la plataforma: Un sistema de reducción por entrelazamiento de cuerdas. Sin embargo, esto conllevó cambios en la configuración del prototipo de los que surgieron dos nuevos diseños. Si bien el estudio CAD de los diseños es externo a este Trabajo, sí que se colaboró identificando las modificaciones necesarias, tanto en la nueva configuración del conjunto, como en la geometría de las partes que se debían añadir.

- Se han aprovechado las pruebas realizadas en el prototipo del resorte para estudiar el comportamiento de un material blando distinto a modo de cuello. De esta manera, los resultados positivos obtenidos han permitido validar la geometría y composición del material de la pieza para su uso en futuras investigaciones de HUMASoft.

En conclusión, se ha logrado ensamblar y poner en funcionamiento un prototipo robótico de cuello blando. Tras las pruebas realizadas en la fase de puesta en marcha, se han obtenido resultados satisfactorios que han validado el diseño de una versión mejorada de la plataforma inicialmente propuesta; el modelo teórico de la cinemática del resorte; las librerías de comunicación para los drivers; y la aplicación de usuario utilizada para comandar movimientos en la plataforma.

7.1. Trabajos futuros

Dado el grado actual del avance del proyecto, surgen de éste diversas líneas de investigación.

- Para completar el proceso de validación de la plataforma sería necesario demostrar que es capaz de soportar el peso de 1 kg, como indican sus especificaciones. La forma más sencilla sería diseñar una pieza capaz de albergar pesas de forma que estas se mantengan inmóviles sobre la base móvil de la plataforma, y realizar las pruebas pertinentes entonces.

- Tras el éxito de las pruebas con el eslabón de material flexible, se debería continuar por este camino con el propósito de obtener un modelo teórico adecuado y presentar este elemento como una alternativa genuinamente blanda

igual de válida para la plataforma que el resorte. De la misma manera, también se propone iniciar un estudio de nuevas geometrías y materiales para la fabricación y evaluación de distintos eslabones blandos.

- Dado que se podría considerar que existe en el prototipo un tercer pseudo-grado de libertad que no ha sido explotado, resultaría interesante realizar modificaciones en la plataforma actual con el objetivo de incrementar sus grados libertad sin modificar el nivel de actuación existente.

- Adentrándonos en trabajos futuros a más largo plazo y siguiendo la línea de investigación del proyecto HUMASoft, sería necesario cumplir dos objetivos: en primer lugar, trasladar la tecnología desarrollada para el cuello robótico a nuevos prototipos para introducir eslabones blandos a modo de brazos o columna vertebral en TEO; en segundo lugar, se debería evaluar el funcionamiento de la plataforma actual, o de una versión superior a esta, ensamblada en el humanoide y sujetando el peso de su cabeza sobre la plataforma móvil.

Anexos

A. Emergency Messages CANopen: Emergency Error Code

Emergency Error Code	Meaning
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	- current device input side
22xx	- current inside the device
23xx	- current device output side
30xx	Voltage
31xx	- mains voltage
32xx	- voltage inside the device
33xx	- output voltage
40xx	Temperature
41xx	- ambient temperature
42xx	- device temperature
50xx	Device hardware
60xx	Device software
61xx	- internal software
62xx	- user software
63xx	- data set
70xx	Additional modules
80xx	Monitoring
81xx	- Communication
8110	- - CAN overrun
8120	- - Error Passive
8130	- - Life Guard/Heartbeat Error
8140	- - Recovered from Bus-Off
82xx	- Protocol Error
8210	- - PDO not processed due to length error
8220	- - Length exceeded
90xx	External error
F0xx	Additional functions
FFxx	Device specific

Tabla 14: Emergency Error Codes de mensajes de emergencia de CANopen. [30]

B. Métodos de la clase CiA301CommPort

- **long CanOpenToCanBus(const co_msg &input, can_msg &output** Convierte un objeto o mensaje CANopen en un objeto o mensaje CANbus. Para ello, recibe dos parámetros pasados por referencia. Uno de estos será un objeto CANopen, que la función convertirá en el otro, un objeto CANbus. Esto se hace copiando el id, el dlc, el rtr, el ts y los datos del objeto CANbus en el objeto CANopen.

- **long CanBusToCanOpen(const can_msg &input, co_msg &output)** Efectúa la tarea inversa a la función anterior. Esta función convierte un objeto o mensaje CANbus en otro objeto o mensaje CANopen, ambos pasados por referencia. Para ello copia en el mensaje CANopen la información del mensaje CANbus: el id, el dlc, el rtr, el ts y los datos del mensaje.

- **int SendMessage(co_msg input)** Esta función envía un mensaje CANopen. Primero, transforma un objeto CANopen a mensaje CAN utilizando la función CanOpenToCanBus y, si no se ha producido ningún error, escribe el mensaje CAN generado en el puerto con la función PutMsg.

- **int SendCanMessage(can_msg &input)** Tiene una funcionalidad muy similar a la función anterior: enviar un objeto CAN por el puerto utilizando PutMsg. Sin embargo, en este caso la función recibe por parámetro un objeto CAN en vez de CANopen por lo que la conversión no será necesaria.

- **co_msg SetCanOpenMsg(unsigned short id_co, unsigned short rtr, vector <uint_8t> coDataFrame)** Construye un objeto CANopen a partir de los parámetros proporcionados. Primero crea un objeto CANopen para posteriormente almacenar en él el valor de los parámetros proporcionados: id, rtr, y el data frame de CANopen. La función devuelve el objeto CANopen ya generado.

- **long GetMsg(can_msg &msg)** Esta función lee un mensaje que se encuentre actualmente en el puerto y lo copia en el objeto CAN pasado por referencia. Esta función devuelve cero si se ha efectuado correctamente la lectura, y -1 si ha habido un error en la lectura o se ha sobrepasado el tiempo máximo de espera.

- **long DataToInt(const uint_8t in, unsigned short size)** Esta función convierte los datos almacenados en un vector con un tamaño determinado, tanto el vector como el tamaño pasados por parámetro, en un número entero de tipo long.

- **uint_32t data4x8to32(const vector uint_8t in, int dlc)** Esta función convierte información dada en un vector de 4 enteros de 8 bits en un entero de 32 bits, devolviendo este dato. En sí, tiene una funcionalidad muy similar a la función DataToInt.

- **int ReadCobId(uint16_t expected_cobid, co_msg &output)** Esta función lee el puerto hasta recibir la respuesta CANopen esperada para posteriormente leerla. Para ello se le pasa por parámetro el COB-ID esperado y el objeto CANopen donde se desea escribir la respuesta. La función comprueba repetidamente si coincide el COB-ID esperado con el del mensaje que se encuentra en el puerto; y si no lo hace, comprueba simplemente el ID. En caso de que no coincida, se almacena el mensaje que posteriormente se mostrará por pantalla. Si se han superado el número de repeticiones de búsqueda del COB-ID se muestra un mensaje y se detiene la función devolviendo -1. Si por el contrario se ha encontrado una coincidencia de COB-ID, se almacena en el parámetro output y se devuelve.

- **long WritePDO(const vector <uint_8t> &command)** Escribe un objeto PDO con el contenido que se pasa por parámetro. Para ello se utilizan las funciones SetCanOpenMsg para formar el objeto CANopen con el COB-ID de transmisión y el comando proporcionado, para luego enviarlo utilizando la función SendMessage. De la misma manera, envía una petición de respuesta y se obtienen los datos usando la función ReadCobId.

- **long WritePDO4(const vector <uint_8t> &command)** Escribe un objeto PDO con el contenido que se pasa por parámetro. Funciona de forma muy similar a WritePDO, aunque en este caso el COB-ID enviado es el rx4 en vez del rx0.

- **long ReadPDO(long number)** Esta función permite recibir un objeto PDO. Para ello utiliza la función ReadCobId, en la que el número que se pasa por parámetro se utiliza para indicar el COB-ID del objeto PDO que se desea recibir pudiendo ser 0, 1, 2, 3 para COB-ID de 180h, 280h, 380h y 480h.

- **long WriteSDO(const vector <uint_8t> &address, const vector <uint_8t> &value)** Genera y transmite un objeto SDO con la dirección que se ha pasado por parámetro. El primer valor de la cadena se escribe en función de la longitud del mensaje a enviar. Se genera el dato introduciendo, tras la información de longitud, la dirección proporcionada y tras esta, el valor del comando que se ha pasado igualmente como parámetro. Después, se introduce en la función SetCanOpenMessage para formar el objeto CANopen

y luego enviarlo con SenCanOpenMsg. Finalmente, se espera una respuesta para comprobar que el objeto se ha enviado correctamente.

- **long ReadSDO(const vector <uint_8t> address)** Lee el objeto SDO que se encuentre en la dirección que se pasa como parámetro. Para ello se escribe un objeto SDO con el valor del mensaje vacío. El retorno de la función es a su vez el retorno de WriteSDO siendo esta la información del objeto SDO al que corresponde a la dirección indicada.

- **long WriteNMT(const vector <uint_8t> &nmtCommand)** Esta función transmite un objeto NMT con el comando introducido por parámetro. Genera primero la parte de datos del objeto para posteriormente hacer uso de las funciones SetCanOpenMessage y SendMessage para generar el objeto y enviarlo.

- **long ReadNMT(const vector <uint_8t> &nmtCode)** Esta función se utiliza para recibir un objeto NMT. Se toma el código NMT previsto pasado por parámetro y se le añade al final el id del nodo, para posteriormente pasar el conjunto por la función DataToInt que lo convertirá en un número entero, siendo este el NMT esperado. Posteriormente se toma el objeto NMT que se encuentre en el puerto manteniéndose luego a la espera de recibir respuesta. Si esta no coincide con la prevista o se han superado los ciclos de espera, se avisa por pantalla al usuario.

- **long FlushBuffer()** Esta función limpia el buffer de comunicación para evitar el solapamiento de mensajes y obtener una comunicación limpia y precisa.

- **long FlushBuffer(int msgs)** Funciona de forma similar a la función FlushBuffer() con la diferencia de que en este caso se eliminan del buffer un número determinado de mensajes. Este valor se introduce por parámetro.

- **long ReadErrorNMT()** Retorna la respuesta del COB-ID de un nodo tras enviarle un objeto NMT con un mensaje RTR Node Guarding utilizando la función ReadCobId.

- **int WaitForReadMessage(co_msg &output, unsigned int canIndex)** Esta función consiste en un bucle de espera hasta la lectura de un objeto CAN. Se imprime por pantalla su id y los datos que contiene. Este mensaje es transformado a un objeto CANopen que se copia en el objeto CANopen pasado por referencia, imprimiendo también por pantalla su COB-ID y sus datos. Si todo se ha efectuado correctamente, la función devuelve 0; si por

el contrario se ha esperado más del tiempo máximo o la conversión ha sido errónea, esta función devuelve -1.

C. Métodos de la clase CiA402Device

- **long Reset()**Esta función resetea el nodo del driver. Primero escribe un mensaje NMT fullreset(82h) que resetea dicho nodo, y se espera que haya pasado a estado Pre-operacional leyendo los mensajes del puerto. Entonces se envía un mensaje de start(01h), para que el nodo vaya al estado Operacional y vuelva a estar en funcionamiento.

- **long StartNode()** Es una función de inicialización que inicia el nodo enviando un mensaje NMT de start(01h).

- **int CheckError()** Esta función comprueba si existen mensajes de error. Para ello se leen los mensajes de error SDO que además se imprimen por pantalla.

- **long Init(vector<uint8_t> new_id)** Función de inicialización. Está función cambia el id del objeto que controla el dispositivo. Es importante que este id sea único de forma que se puedan diferenciar los distintos dispositivos.

- **uint16_t CheckStatus()** Devuelve el estado (de la máquina de estados) en el que se encuentra actualmente el dispositivo en un entero de 16 bits. Para ello, se lee el estado de un mensaje SDO tras haber pedido este dato mandando previamente un vector conteniendo el statusword.

- **void PrintSatus()** Al igual que la función anterior, también se accede al estado del dispositivo, solo que en este caso no se devuelve ningún dato. El estado se imprime por pantalla. Al igual que en la función CheckStatus(), se lee el estado de un mensaje SDO tras pedir este dato utilizando el statusword.

- **long Switchon()** Enciende el dispositivo y lo deja operativo para que los motores controlen su eje y se pueda comandar algún movimiento. La función comprueba el estado en el que se encuentra el dispositivo y en función de ese dato, envía los mensajes PDO necesarios para pasar de estado en estado hasta finalizar en Enable Operation. Devuelve 0 si el encendido se ha realizado correctamente y no existen errores.

- **long SwitchOff()** Realiza un apagado rápido del dispositivo. Para ello, sin importar el estado en el que se encuentre el dispositivo en ese momento, la función escribe un mensaje PDO que contiene el dato de quickstop. Tras utilizar esta función, el estado del dispositivo es Quik Stop. El retorno de la función es 0 si se ha ejecutado correctamente. Esta función está inacabada pues la funcionalidad de enviar QuickStop corresponde a la función con su mismo orden. Una vez correctamente implementada, Switch Off detectará el

estado en el que se encuentra el dispositivo en ese momento y, en función de cuál sea este último, pasará de estado en estado para realizar un correcto apagado hasta que el estado del dispositivo sea Switch On Disable.

- **long QuickStop()** Esta función tiene un resultado muy similar a la anterior, ya que es la que genuínamente ha de realizar esta tarea. QuickStop se utiliza como parada de emergencia, ya que directamente lleva al dispositivo al estado de Quick Stop. Para ello se da la orden de QuickStop en un mensaje PDO. El retorno de la función es un 0 si se ha realizado correctamente y no se han producido errores.

- **long ForceSwitchOff()** Esta función tiene un efecto similar a las anteriores excepto que no termina en el estado de Quick Stop si no que manda un objeto PDO con el mensaje goswitchondesable(0060h) del diccionario de objetos para que el dispositivo cambie directamente al estado Switch On Disable, sin pasar por estados intermedios. Si se ha efectuado correctamente, el retorno de la función es 0.

- **long GetPosition()** La función retorna el objeto Position Actual Value tras haberse convertido a grados. Position Actual Value(6064h) es la posición medida por el encoder relativo utilizado. Para obtener este objeto, se lee un mensaje SDO tras mandar el dato positionaddress. Finalmente, la función GetPosition retorna este valor en grados tras serle aplicado el factor de conversión.

- **long GetVelocity()** Funciona de forma muy similar a GetPosition() a excepción de que en este caso el objeto retornado es Velocity sensor actual value (6069h) que corresponde a la velocidad leída por el por encoder relativo conectado al driver. Es necesario mandar el vector velocityaddress y, tras obtener el objeto, deberá ser aplicado el factor de conversión para retornar el resultado en grados por segundo.

- **long SetCommunications(int fdPort)** Esta función establece el puerto que se le pasa por parámetro como el puerto en el que se realizarán las comunicaciones.

- **long OperationMode(const vector<uint8_t> new_mode)** Esta función se utiliza para cambiar el modo de operación del driver. Es decir, para que el driver cambie el control del motor a control en posición, velocidad o fuerza. Para ello hay que pasar por parámetro el modo al que se quiere cambiar. La función primero lee el modo actual y lo muestra por pantalla, para luego cambiar el modo de operación enviando un objeto SDO con el nuevo modo. Nuevamente vuelve a leer el modo actual y lo muestra por pantalla.

- **long SetPosition(long target)** Si el dispositivo está en Position Mode esta función permite realizar un control en posición en un motor y permite realizar un movimiento hacia la posición seleccionada. Esta posición, pasada por parámetro, ha de ser introducida en grados ya que se realizará su posterior conversión en la función. Hay que tener en cuenta que el motor identifica la posición de encendido como la posición de inicio, retornando a esta cada vez que se introduce como objetivo una posición de 0 grados. De igual forma si se introduce un valor superior a 360 el motor realizará más de una vuelta.
  Para la ejecución de esta función, en primer lugar se convierte el parámetro introducido a líneas del encoder (4096 para un vuelta en nuestro caso), teniendo en cuenta la reducción de 3,7 de las reductoras. Posteriormente, se manda un mensaje SDO indicando que la target-position(000607Ah) se modifique al valor de nuevo dato. Esta es comprobada, de la misma forma que también se comprueba el status word(0641h). Finalmente, se escribe un último mensaje SDO con la orden run(001Fh) para que el motor realice el movimiento.

- **long SetupPositionMode(const vector<uint32_t> velocity, const vector<uint32_t> acceleration)** Esta función configura el Modo Posición. Para ello primero se cambia el modo de operación a modo posición. Posteriormente se convierte la velocidad dada en grados/segundo a líneas del encoder/muestra(0,001s) y se forma el dato de velocidad trasladando esta información dos bytes a la izquierda. Se realiza el mismo proceso para convertir la aceleración en líneas/muestras^2 y formar el dato de aceleración. Ambos son posteriormente utilizados para cambiar la configuración del perfil de de velocidad y aceleración del modo posición enviándolos mediante mensajes SDO.

- **long SetVelocity(double target)** Si el dispositivo está en Speed Mode, esta función permite realizar un control en velocidad en un motor y permite realizar un movimiento a la velocidad seleccionada. Esta velocidad, pasada por parámetro, ha de ser introducida en grados/segundo ya que se realizará su posterior conversión. Posteriormente, se manda un mensaje SDO indicando que la target-velocity(00060FFh) se modifique al valor del nuevo dato.

- **long Setup_Velocity_Mode(const vector<uint32_t> target, const vector<uint32_t> acceleration)** Esta función configura el Modo Velocidad. Para ello primero se cambia el modo de operación a modo velocidad. Utiliza los datos pasados por parámetro para generar el nuevo valor de la velocidad objetivo y del perfil de aceleración. Estos son luego escritos en sus respectivos objetos (target_velocity y profile_acceleration) utilizando mensajes SDO.

- **long SetTorque(double target)** Si el dispositivo está en Torque Mode, esta función permite realizar un control en fuerza y cambiar la fuerza aplicada al motor a la fuerza seleccionada que se pasa por parámetro. Este valor es en realidad el tanto por mil de la intensidad máxima aplicada al motor. Por lo que si la fuerza (target) está dentro de los límites de -1000 a 1000, se cambia la fuerza objetivo a este valor; si no, se utiliza el valor del límite sobrepasado. El dato de la fuerza objetivo (torque_target) se cambia utilizando un objeto SDO, para posteriormente poner el dispositivo en movimiento con un objeto SDO conteniendo el dato de run(001Fh).

- **long Setup_Torque_Mode()** Esta función se utiliza para cambiar el Modo de Operación al Modo Fuerza utilizando una referencia externa digital. De esta manera se puede realizar control de fuerza en el motor.

D. Métodos de la clase PortBase y SocketCanPort

Métodos de la clase PortBase

- **int getPortId()** Esta función devuelve el Id del puerto.
- **long FlushMsg()** Esta función limpia los mensajes residuales que se encuentran el puerto.

Métodos de la clase SocketCanPort

- **long SetFilter(uint32_t canId, uint32_t mask)** Esta función filtra los mensajes CAN con un Id y máscara determinados, de forma que únicamente se reciban los mensajes de interés para un id concreto.
- **long GetNMT(uint_8t *data, uint_8t &size)** Esta función recibe mensajes NMT del puerto, y almacena los datos y el tamaño de los datos en las variables pasadas por parámetro. Si ha ocurrido un error en la lectura o en el tiempo de espera, la función lo comunicará por pantalla y devolverá -1.
- **long GetMsg(uint_32t &canId, uint_8t *data, uint_8t &size)** Esta función recibe mensajes del puerto, y almacena su Id, los datos y el tamaño de los datos en las variables pasadas por parámetro. Si ha ocurrido un error en la lectura o en el tiempo de espera, la función lo comunicará por pantalla y devolverá -1.
- **long PutMsg(const uint_32t &canId, uint_8t * const data, uint_8t const data_size)** Esta función escribe en el puerto un objeto CAN con el Id, el dato y el tamaño del dato pasados por parámetro.
- **long Init(string canPort)** Esta función inicializa un puerto socket.

Referencias

[1] C.Laschi, B. Mazzolai, M. Cianchetti, *Technologies and systems pushing the boundaries of robot abilities*, Science Robotics, Vol.1, Issue 1, dec 2016. [En línea]. Disponible en: http://robotics.sciencemag.org/content/1/1/eaah3690, Acceso: Abril 2018.

[2] C.Laschi, M. Cianchetti, *Soft robotics: new perspectives for robot bodyware and control*, Frontiers, 30 enero 2014. [En línea]. Disponible en: https://www.frontiersin.org/articles/10.3389/fbioe.2014.00003/full

[3] Proyecto *OCTOPUS: Novel Design Principles and Technologies for a New Generation of High Dexterity Soft-bodied Robots Inspired by the Morphology and Behaviour of the Octopus*, The Biorobotics Institute, Scuola Superiore Sant'Anna, Italia, Comunidad Europea (7 Programa Marco), 2009-2013.

[4] Proyecto *Rollin' JUSTIN*, Instituo de Robótica y Mecatrónica del Centro Aeroespacial Alemán (DLR), Alemania, Comunidad Europea (H2020), 2015-2019.

[5] Proyecto *Soft Robotics*, Foundational Labs, Universidad Técnica de Delft (TU DELFT), Países Bajos, 2015.

[6] Proyecto *COMANOID: Multi-Contact Collaborative Humanoids in Aircraft Manufacturing*, Centre National de a Recherche Scientifique (CNRS), Francia, Comunidad Europea (H2020), 2015-2019.

[7] S.Martínez, C. A. Monje, A. Jardón, P. Pierro, C. Balaguer, D. Muñoz, *TEO: Full-Size Humanoid Robot Design Powered by a Fuel Cell System*, Cybernetics and Systems, Vol. 43, N.3, pp 163-180, Taylor & Francis, 2012.

[8] Proyecto *HUMASoft: Diseño y Control de Eslabones Blandos para Robots Humanoides*, RoboticsLab, Universidad Carlos III de Madrid, España, Ministerio de Economía y Competitividad, 2016-2019.

[9] L. Nagua, C.A. Monje, *Tesis de Máster: Diseño y simulación de un prototipo de cuello robótico*, Departamento de Ingeniería de Sistemas y Automática, Universidad Carlos III de Madrid, España, 2018.

[10] B. Gao, N. Xi, Y. Shen, J. Zhao, R. Yang, *Development of a Low Motion-Noise Humanoid Neck: Statics Analysis and Experimental Validation*, In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 1203-1208, IEEE, 2010.

[11] B. Deutschmann, C. Ott, C.A. Monje, C. Balaguer *Robust Motion Control of a Soft Robotic System Using Fractional Order Control*, Transactions on Industrial Electronics, Aceptado para publicación en 2016.

[12] S. Timoshenko *Theory of elastic stability*, New York McGraw-Hill Book Company, inc, 1st ed edition, 1936.

[13] J. Gunsing, F. Gijselhart, N. Hagemans, H. Jonkers, E. Kivits, P. Klijn, B. Kapteijns, D. Kroeske, H. Langen, B. Oerlemans, J. Oostindieand Joost van Stuijvenberg, *Adaptive Robotic Systems Design in University of Applied Science*, Research Group for Robotics & Mechatronics, Avans University of Applied Sciences, Países Bajos, 2016.

[14] H. Zimmermann, *OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection*, IEEE Transactions on Communications, vol. 28, no. 4, april 1980, pp. 425 - 432.

[15] *CAN History*, CiA, [En línea]. Disponible en: http://www.can-cia.org/can-knowledge/can/can-history/. Acceso: Abril 2018

[16] *Introducción a CAN*, National Instruments, [En línea]. Disponible en: http://www.ni.com/white-paper/2732/es/. Acceso: Abril 2018

[17] *Can Bus Wiring Diagram*, Eidetec.com, [En línea]. Disponible en: http://eidetec.com/can-bus-wiring-diagram. Acceso: Abril 2018

[18] *CAN in Automation-Mercedes W140: First car with CAN*, CAN newsletter, [En línea]. Disponible en: http://can-newsletter.org/engineering/applications/160322_25th-anniversary-mercedes-w140-first-car-with-can/. Acceso: Abril 2018

[19] *Normas de la Capa Física de CAN: Alta-Velocidad vs. Baja-Velocidad/Tolerante a Fallas CAN*, National Instruments, [En línea]. Disponible en: http://digital.ni.com/public.nsf/allkb/9E97DC2B8C320544862579 7B00571E04/. Acceso: Abril 2018

[20] *CAN high-speed transmission*, CiA, [En línea]. Disponible en: https://www.can-cia.org/can-knowledge/can/high-speed-transmission/ Acceso: Abril 2018

[21] *ISO 11898-2:2016 Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit*,INTERNATIONAL STANDARD ISO, pp. 1. ISO 11898-2:2016

[22] *ISO 11898-3 Road vehicles — Controller area network (CAN) — Part 3: Low-speed, fault-tolerant, medium-dependent interface*, INTERNATIONAL STANDARD ISO, pp. 1. ISO 11898-3:2016

[23] *SAE J2411 single wire*, CiA, [En línea]. Disponible en: https://www.can-cia.org/can-knowledge/can/sae-j2411-single-wire/ Acceso: Abril 2018

[24] J.A. López Fresno, E. Gil Dolcet *Nodo de comunicación basada en el bus CAN*, Departament d'Enginyeria Electrònica Elèctrica i Automàtica, Universitat Rovira i Virgili, España, 2004, pp. 30-32.

[25] *CAN data link layers in some detail*, CiA, [En línea]. Disponible en: http://www.can-cia.org/can-knowledge/can/can-data-link-layers/. Acceso: Abril 2018

[26] *CANopen history*, CiA, [En línea]. Disponible en: https://www.can-cia.org/can-knowledge/canopen/canopen-history/ Acceso: Abril 2018

[27] *CANopen Basics - Introduction*, CANopenSolutions.com, [En línea]. Disponible en: http://www.canopensolutions.com/english /about_canopen/about_canopen.shtml Acceso: Abril 2018

[28] *The Basics of CANopen*, National Instruments, [En línea]. Disponible en: http://www.ni.com/white-paper/14162/en/ Acceso: Abril 2018

[29] *CiA 301 CANopen Application layer and communication profile*,Version: 4.0.2, CAN in Automation (CiA) e. V., 2002

[30] R. Sánchez Díaz, *CANopen*, Departamento de Informática y Automática, Universidad de Salamanca,España.

[31] *Heartbeat and Node Guarding*, National Instruments, [En línea]. Disponible en: http://zone.ni.com/reference/en-XX/help/373254C-01/canopenhelp/canopen_heartbeat_and_node_guarding/ Acceso: Mayo 2018

[32] *CANopen Basics-Guarding and Heartbeat*, CANopenSolutions.com,[En línea]. Disponible en: http://www.canopensolutions.com/english/about_canopen/guarding_heartbeat.shtml Acceso: Mayo 2018

[33] *Predefined use of message identifiers (Predefined Connection Set) for simple system structures*, CANopenSolutions.com,[En línea]. Disponible en: http://www.canopensolutions.com/english/about_canopen/predefined.shtml Acceso: Mayo 2018

[34] *CiA® 402 series: CANopen device profile for drives and motion control*,CiA, [En línea]. Disponible en: https://www.can-cia.org/can-knowledge/canopen/cia402. Acceso: Abril 2018

[35] *Technical documents*,CiA, [En línea]. Disponible en: https://www.can-cia.org/standardization/specifications/. Acceso: Abril 2018

[36] *iPOS CANopen Programming User Manual*,Technosoft, 2015.