UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN.

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA EN TIEMPO REAL PARA EL CONTROL DE SERVOMOTORES DIGITALES DE UN BRAZO ROBÓTICO

Miguel Pérez Ávila

2017

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

Reunido el tribunal examinador en el día de la fecha, constituido por

Presidente:

Vocal:

Secretario:

Suplente:

para juzgar el Trabajo Fin de Titulación titulado:

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA EN TIEMPO REAL PARA EL CONTROL DE SERVOMOTORES DIGITALES DE UN BRAZO ROBÓTICO

del alumno D. Miguel Pérez Ávila

dirigido por Dr. D. Álvaro Gutiérrez Martín

del Departamento de Tecnología Fotónica y Bioingeniería

Acuerdan otorgar la calificación de: _____

Y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia

Madrid, de Enero de 2017

El Presidente	El Vocal	El Secretario
Fdo: _____	Fdo: _____	Fdo: _____

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA EN TIEMPO REAL PARA EL CONTROL DE SERVOMOTORES DIGITALES DE UN BRAZO ROBÓTICO

Miguel Pérez Ávila

2017

Resumen

El propósito de este Trabajo de Fin de Grado es migrar el control del brazo robótico comercial PhantomX de la compañía Trossen Robotics a la tarjeta de control NUCLEO-446RE. La primera fase se basa en controlar los motores AX-12A, servos digitales de la compañía Robotis. Se utilizará el hardware de la tarjeta para controlar vía software estos motores, implementando su protocolo de comunicación. Posteriormente, se procede a desarrollar el control como sistema robótico, el cual, mediante el uso de la cinemática directa e inversa del robot, garantiza la trayectoria del efector final en el tiempo y en el espacio. Una vez conseguido el control final, se procede al desarrollo de una interfaz de comunicación para el manejo del brazo.

Abstract

The aim of this Thesis is to migrate the control of a comercial robotic arm from the Trossen Robotics company to the NUCLEO-446RE control board. The first phase is based on the control of the AX-12A motors, which are digital servos from the Robotis company. The hardware from the board is used to control these motors by software, according to the mandatory comunication protocol. Afterwards, the control of the robotic arm is developed as a system, which guarantees the trajectory planning of the end-effector in time and space by using forward and inverse kinematics. Finally, a communication interface is developed to manage the control of the robot by external elements.

Agradecimientos

A mi tutor, Álvaro Gutiérrez Martín, por mostrarme el Robolabo y permitirme entrar en un lugar en el que formarme y crecer.

A Dani, por ser un amigo leal, de esas personas que marcan y que siempre están ahí.

A mis amigos, por ser un pilar indestructible en mi vida y en mis convicciones.

A mi familia, por habérmelo dado todo y haber confiado siempre en mí.

Y, por último, a Paula, por permitirme conocer a una persona única, por ser mi amiga más íntima, por todo,

Gracias.

Índice general

Índice de figuras

Índice de tablas

Lista de Acrónimos

ARM: Advanced Risc Machine.

ST: STMicroelectronics.

DSP: Digital Signal Processing.

UART: Universal Asynchronus Receiver Transmitter.

HAL: Hardware Abstraction Lawyer

Capítulo 1

Introducción y Motivación

1.1. Introducción a la Robótica

El término robot fue introducido por primera vez por el escritor checo de ciencia ficción Karel Capek en 1920 en su obra Rossumś Universal Robots (Wikipedia, 2018), donde el termino *robota* es una palabra checa para trabajo. Desde entonces este término se ha utilizado para una gran variedad de elementos mecánicos que desarrollan una acción sin la interacción humana.

Una definición formal de robot la encontramos en el Robot Institute of America (RIA): *"Un robot es un manipulador multifuncional reprogramable diseñado para mover materiales, piezas, herramientas o dispositivos especiales mediante diversos movimientos programados para una gran variedad de tareas."* El elemento principal de esta definición es la condición de reprogramable, que otorga a los robots una gran utilidad y adaptabilidad a diferentes situaciones.

La robótica es un campo relativamente joven de la tecnología moderna, en el que confluyen diversas tecnologías tradicionales, como la electrónica, mecánica y programación, y en la que sus aplicaciones abarcan muchos campos. La robótica la podemos encontrar tanto en el proceso de fabricación de una empresa, en la que garantiza un aumento de la eficiencia y la precisión en las tareas en las que se ve involucrada, como en campos sociales, como pueden ser el uso de la robótica para el apoyo a personas de la tercera edad, en las que se busca un refuerzo en las tareas cotidianas que actualmente es difícil de lograr. También hay aplicaciones que permitirán al ser humano avanzar hacia el futuro, como es el uso de robots en tareas espaciales o trabajo en centrales nucleares que actualmente son altamente peligrosas o imposibles de realizar.

Aunque las aplicaciones de la robótica sean muy variadas, una de las más extendidas es la robótica industrial. En esta abundan muchos tipos de brazos robóticos para el desarrollo de tareas de fabricación en serie. En este Trabajo Fin de Grado se ha utilizado un modelo de brazo robótico de bajo coste que sigue los mismos principios que los robots de talla superior para el mundo industrial, donde en general, estos brazos son complejos sistemas electromecánicos cuya representación analítica tiene muchas dificultades.

Para introducir completamente la robótica se deben de explicar los términos empleados para definir el modelo matemático con el que se rigen. Estos modelos representan la geometría básica del robot para su manipulación, que permiten desarrollar métodos para planificar y controlar sus movimientos.

En este campo se utiliza una representación simbólica para poder definir a estos manipuladores. Estos están compuestos de distintos enlaces que se juntan con articulaciones para formar cadenas cinemáticas. Estas cadenas cinemáticas permiten obtener movimientos restringidos a los sistemas mecánicos y, sobre todo, permiten su modelado. Las articulaciones son la unión de dos enlaces y las típicas son las de rotación o lineales, que son definidas como R y P, respectivamente. Típicamente, en el extremo de todo manipulador existe una herramienta que es la que realizará las actividades para las que está diseñado el robot.

Esta representación simbólica permite desarrollar una configuración del robot, la cual permite definir una especificación completa de cada punto del robot. Con esto podemos definir los grados de libertad que tiene el robot. El robot tendrá n grados de libertad si se puede definir este espacio de configuración con solo esos n parámetros. Un cuerpo rígido en el espacio se representa completamente con 6 grados de libertad: tres para la posición y tres para la orientación. Pero, aun así, la mayoría de manipuladores robóticos suelen tener menos de 6 grados de libertad, lo que impide que puedan realizar todos los movimientos posibles con cualquier tipo de orientación. Este espacio al que pueden acceder es el que se conoce como espacio de trabajo.

1.2. Clasificación de Manipuladores

Como en muchos campos, existen diferentes tipos de clasificaciones de robots, entre otros, la fuente de potencia, la geometría, el área de aplicación o el método de control. Sin embargo, la mayoría de brazos robóticos se clasifican según su geometría. Como ya hemos dicho, los manipuladores industriales suelen tener 6 o menos grados de libertad y se clasifican cinemáticamente según las 3 primeras articulaciones que poseen. La mayoría recaen en 5 tipos de arquitecturas: articulados (RRR), esféricos (RRP), SCARA (RRP), cilíndricos (RPP) y cartesianos (PPP).

1.2.1. Manipuladores Articulados (RRR)

Un manipulador articulado es también llamado manipulador antropomórfico. Este manipulador proporciona una amplia libertad de movimiento en un espacio compacto. En la Figura 1.1 se puede observar cómo está formada su estructura y en la Figura 1.2 se puede observar su espacio de trabajo.

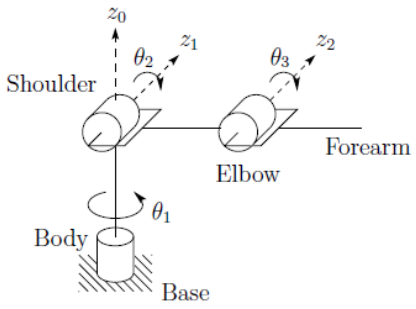

Figura 1.1: Estructura del manipulador articulado (MW Spong, S. Hutchinson, MVidyasagar, 2005).

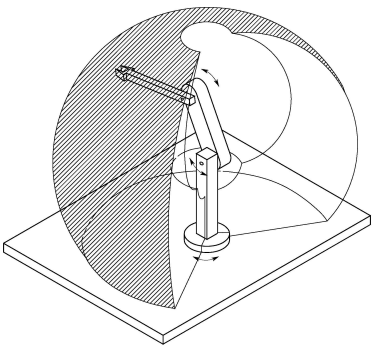

Figura 1.2: Espacio de trabajo del manipulador articulado (MW Spong, S. Hutchinson, MVidyasagar, 2005).

1.2.2. Manipuladores Esféricos (RRP)

Reemplazando la tercera articulación del manipulador antropomórfico por una articulación lineal o prismática, se obtiene un manipulador esférico. El nombre procede de definir la posición de la herramienta del brazo a través de las coordenadas esféricas. En la Figura 1.3 se detalla la estructura genérica del modelo, y en la Figura 1.4 se muestra el espacio de trabajo.

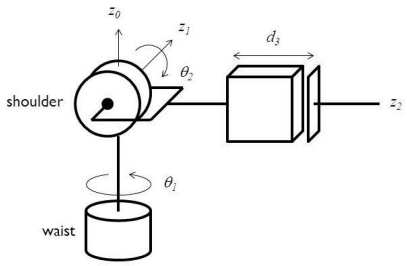

Figura 1.3: Estructura del manipulador esférico (MW Spong, S. Hutchinson, MVidyasagar, 2005).

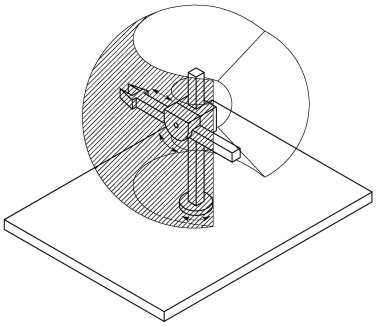

Figura 1.4: Espacio de trabajo del manipulador esférico (MW Spong, S. Hutchinson, MVidyasagar, 2005).

1.2.3. Manipuladores SCARA (RRP)

El SCARA (Selective Compliant Assembly Robot Arm), de Robot Articulado de Conformidad Selectiva para Ensamblaje en sus siglas en inglés, es un manipulador muy popular en tareas de ensamblaje. Aunque el SCARA también tiene estructura RRP es diferente del modelo anterior en apariencia y en rango de aplicaciones. En la Figura 1.5 se muestran las diferencias estructurales comparadas con el caso anterior, pero su espacio de trabajo es idéntico al de la Figura 1.4.

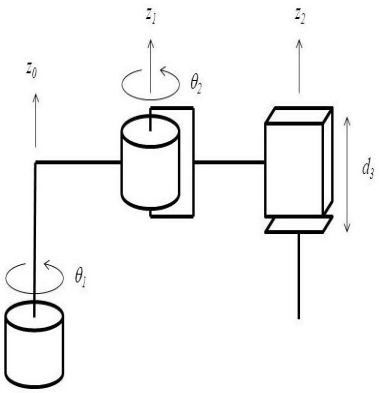

Figura 1.5: Estructura del manipulador SCARA (MW Spong, S. Hutchinson, MVidyasagar, 2005).

1.2.4. Manipuladores Cilíndricos (RPP)

En este manipulador la primera articulación es de rotación, pero las dos siguientes son prismáticas. Como el nombre indica, las variables de las articulaciones y las coordenadas de la herramienta están en función de las coordenadas cilíndricas. La estructura del robot se puede estudiar en la Figura 1.6 y el espacio de trabajo se detalla en la Figura 1.7.

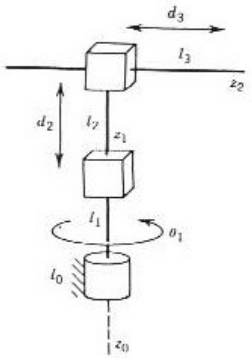

Figura 1.6: Estructura del manipulador cilíndrico (MW Spong, S. Hutchinson, MVidyasagar, 2005).

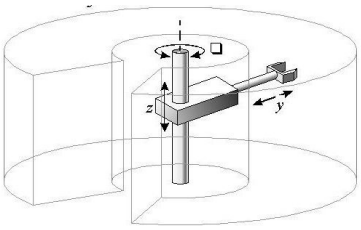

Figura 1.7: Espacio de trabajo del manipulador cilíndrico (MW Spong, S. Hutchinson, MVidyasagar, 2005).

1.2.5. Manipuladores Cartesianos (PPP)

Este manipulador tiene sus tres primeras articulaciones de tipo prismático y es conocido como manipulador cartesiano. Para este manipulador las variables de las articulaciones están en las coordenadas cartesianas y las de la herramienta en base a estas. La descripción matemática de esta estructura es la más sencilla de todas. En la Figura 1.8 se puede observar su estructura y en la Figura 1.9 su espacio de trabajo.

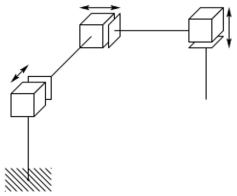

Figura 1.8: Estructura del manipulador cartesiano (MW Spong, S. Hutchinson, MVidyasagar, 2005).

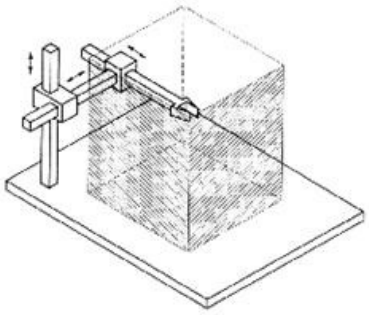

Figura 1.9: Espacio de trabajo del manipulador cartesiano (MW Spong, S. Hutchinson, MVidyasagar, 2005).

1.3. El Robot del Trabajo Fin de Grado

El robot utilizado en este Trabajo Fin de Grado es el PhantomX Reactor Robot Arm (ver Figura 1.10) de la compañía Trossen Robotics[1]. Este manipulador es de tipo Articulado (RRR) puesto que sus tres primeras articulaciones son de rotación. Como herramienta final incorpora una pinza para poder coger objetos.

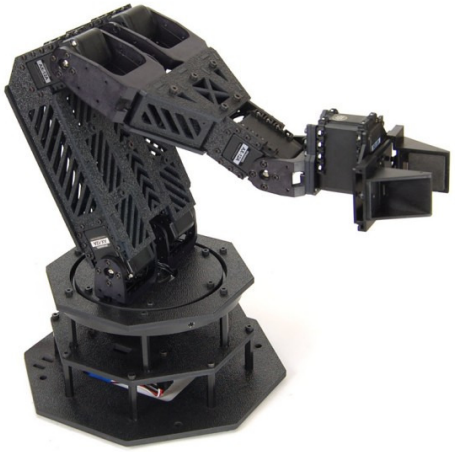

Figura 1.10: Manipulador del Trabajo de Fin de Grado de la compañía Trossen Robotics.

Este manipulador es un robot de bajo coste con 5 grados de libertad que permite iniciarse en el mundo de la robótica de una forma fácil y sencilla. Cada articulación consta de un servomotor digital de Robotis (Robotis, 2018). Gracias a estos servomotores, en los cuales el control de posición y velocidad ya está integrado, se consigue un control preciso del movimiento del manipulador.

[1] http://www.trossenrobotics.com/

Trossen Robotics proporciona hardware y software propio para el control de este robot. En concreto utilizan la tarjeta de control ArbotiX-M Robocontroller, la cual incluye un Atmega como microcontrolador. Esta tarjeta es compatible con el IDE de Arduino[2] el cual posibilita su programación pero no su depuración, lo que genera diversos problemas. El software es de fácil acceso y permite controlar tanto los motores individualmente como el robot como sistema. En cambio, este software está dedicado al manejo del robot, pero no incluye la cinemática directa e inversa.

1.4. Motivación y Objetivos

La principal motivación de este proyecto es migrar el control del brazo robótico PhantomX Reactor Robot Arm de Trossen Robotics a la tarjeta NUCLEO-F446RE[3] de STMicroelectronics(ST)[4] y utilizar el sistema operativo en tiempo real de código libre FreeRTOS[5]. Esta tarjeta de desarrollo posee un microcontrolador STM32F446, un Cortex M4 de ARM (Advanced Risc Machine). El Cortex M4 es un tipo de arquitectura de microcontroladores de la empresa de ARM[6] que tiene como característica principal un periférico de DSP (Digital Signal Processing) e instrucciones propias para este módulo. La elección de esta nueva tarjeta de desarrollo se debe a su bajo coste y sus altas prestaciones. La elección del sistema operativo es debida a la abstracción que permite tener un sistema en tiempo real y la precisión en el tiempo que garantiza.

Para el Laboratorio de Robótica y Control del Departamento de Tecnología Fotónica y Bioingeniería[7], donde se utiliza el brazo robótico para aplicaciones docentes, era de gran interés la migración del control de estos brazos para reducir ciertos problemas provenientes de la tarjeta de control comercial ArbotiX-M Robocontroller incluida en el brazo. Además, este cambio facilita la implementación de futuras aplicaciones gracias a la amplia documentación procedente de ST.

En la primera fase del Trabajo Fin de Grado se requiere familiarizarse con el sistema comercial, para su futura substitución, tanto con la mecánica, como la electrónica y los motores. Estos motores son los AX-12A de Robotis[8] que se controlan a través de la UART. Son servos digitales, los cuales poseen en su interior un microcontrolador y por medio de un protocolo de comunicación se puede controlar la posición a la que estos deben de llegar, su velocidad máxima y diferentes parámetros de interés.

Tras el desarrollo de la aplicación al más bajo nivel de control electrónico, se procederá al estudio e implementación de los métodos matemáticos que permiten conocer la posición de las articulaciones del robot. Además, se requiere estudiar diferentes técnicas que garanticen la trayectoria del efector final del robot, la pinza en nuestro

[2] https://www.arduino.cc/
[3] http://www.st.com/en/evaluation-tools/nucleo-f446re.html
[4] http://www.st.com
[5] www.freertos.org
[6] www.arm.com
[7] www.robolabo.etsit.upm.es
[8] http://www.robotis.us/dynamixel/

caso. Para finalizar, se procederá a implementar una aplicación a nivel de usuario para controlar el brazo, permitiendo a la tarjeta comunicarse con el exterior.

Capítulo 2

Control de los Motores

Para la implementación de la aplicación se cumplieron ciertas restricciones del PhantomX, las cuales provenían del control de los servos digitales AX-12A. Para controlar estos servos digitales se debía utilizar la interfaz de comunicación UART en modo semi-duplex. Esto es debido a que estos motores solo tienen 3 cables, dos para alimentación y otro de comunicación, en vez de los dos de comunicación (Rx y Tx) para full-duplex. Además, la comunicación con estos motores es según un protocolo de comunicación facilitado por el fabricante. En secciones posteriores se detalla las decisiones tomadas para realizar el control del brazo y cómo se solucionaron algunas restricciones. También, se describen los pasos llevados a cabo en el Trabajo Fin de Grado hasta completar la migración.

2.1. Estructura Software

La idea principal para el desarrollo de este software es utilizar la tarjeta NUCLEO-446RE como medio de control del PhantomX. Para posibilitar la creación de futuras aplicaciones, la tarjeta se debía poder comunicar con el exterior y recibir instrucciones para realizar estas tareas. Por ello, el software desarrollado debía suplir tanto los requisitos impuestos por el PhantomX, como los impuestos con la comunicación por puerto serie con un ordenador (ver Figura 2.1.).

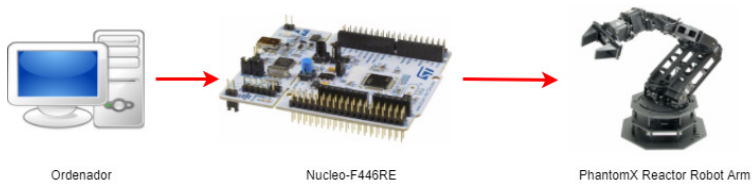

Figura 2.1: Flujo de control de la aplicación.

Mediante una aplicación en el ordenador se enviarán distintas órdenes al NUCLEO que será el encargado de procesarlas y adaptarlas a la comunicación con el robot. Dependiendo de las órdenes que se envíen se impondrán distintos requisitos en el movimiento del robot.

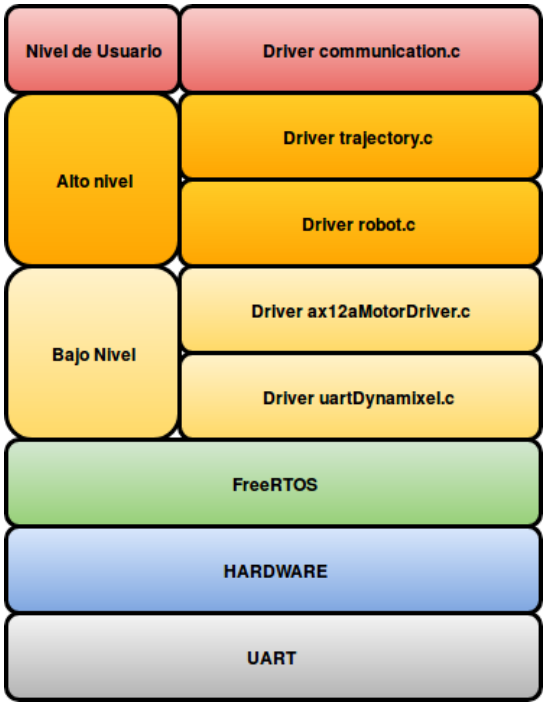

Figura 2.2: Estructura del software.

En la Figura 2.2 se muestra la estructura software de la aplicación. En las capas inferiores está el hardware del sistema y, en las superiores, está el software de control de la aplicación con su propia clasificación. Entre medias se incluye la capa de FreeRTOS, el sistema operativo que controla los recursos del sistema. Este sistema operativo permite el paralelismo en nuestro microcontrolador garantizándose la herencia de tareas y la exclusión mutua. Todas las capas utilizan el sistema operativo para funcionar, usando funciones propias o incorporadas a las tareas del sistema operativo. Volviendo a las capas superiores de la figura, encontramos tres distinciones: software de bajo nivel, de alto nivel y de nivel de usuario.

El software de bajo nivel es el encargado de solucionar los requisitos asociados al control del robot. La capa de *uartDynamixel.c* habilita el control del periférico UART en semi-duplex tal y como requieren los motores y se encarga de enviar la información. Dicha implementación se detalla en la Sección 2.3. La capa superior, *ax12aMotorDriver.c*, es la encargada de implementar el protocolo de comunicación con los motores, que se explica en la Sección 2.4.

El software de alto nivel se vuelve a dividir en dos capas, la más baja, *robot.c*, se encarga del acceso a las articulaciones del robot dando los parámetros necesarios a los servomotores para un correcto funcionamiento. La capa de *trayectory.c* busca perfeccionar el control del robot. Esta capa garantiza la trayectoria de la herramienta final en el espacio. Estas dos capas se detallan en las Secciones 3.2 y 3.3

La última capa, la capa de software de usuario *communication.c*, es la encargada de comunicarse con el exterior y se desarrolla en la Sección 3.4. En ella se implementa un

protocolo de comunicación con distintas órdenes para gestionar tanto el movimiento del robot como su posición. El flujograma seguido se muestra en la Figura 2.3.

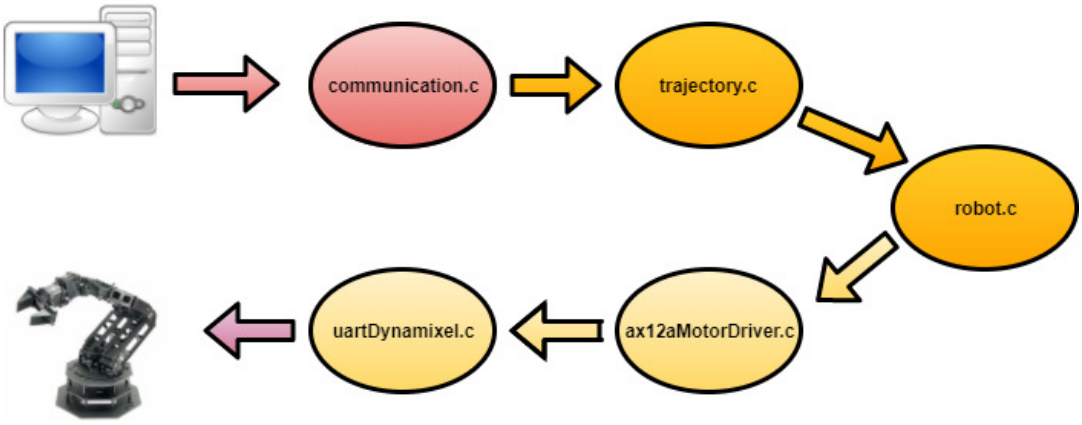

Figura 2.3: Flujograma de control del software.

2.2. Familiarización Núcleo STM32F446

Antes de comenzar con el desarrollo, se explicarán las herramientas utilizadas. Para comenzar se siguió un ejemplo presente en la wiki del departamento[1] en el cual se encendía un led. Una vez hecho eso, se introducía esa misma función dentro de las tareas del sistema operativo FreeRTOS. En las siguientes secciones utilizaremos diversos términos procedentes del sistema operativo, los cuales son: tareas, colas y semáforos. El primero, las tareas, son los hilos de ejecución de nuestro programa, en los cuales introduciremos toda la lógica de nuestra aplicación. El segundo, las colas, son instrumentos que otorga el sistema operativo para poder hacer una comunicación segura de datos entre tareas. Y el tercero, los semáforos, nos permite bloquear un recurso, y por consiguiente una tarea, hasta que otra tarea termine con ese recurso permitiendo que una tarea no gaste recursos del sistema innecesariamente. Por ejemplo, si tenemos dos tareas que acceden al mismo recurso, como puede ser una línea de transmisión, si una tarea está enviando, otra tarea no podrá enviar hasta que la primera haya terminado, la cual se lo hará saber liberando el semáforo.

Dentro de los ejemplos descritos en la wiki, se describía el uso de las herramientas con las que se desarrolló el Trabajo Fin de Grado. Se utilizó el compilador de C para el microcontrolador específico *gcc-arm-none-eabi* y la herramienta para la depuración del código *OpenOCD*[2]. Con esta última herramienta es posible depurar el código dentro del microcontrolador. La tarjeta NUCLEO-446RE dispone de un puerto JTAG y otro microcontrolador dedicado a esta tarea.

La familiarización con todas las herramientas para comenzar el desarrolló del Trabajo Fin de Grado fue lenta. El estudio de las herramientas de FreeRTOS fue la que

[1] `www.wikirobolabo.etsit.upm.es`
[2] `http://openocd.org/`

más complicaciones aportó. La configuración del sistema operativo y su inclusión con la HAL procedente del fabricante requirió de mucho esfuerzo y conocimiento del hardware utilizado. Finalmente, se decidió diseñar un software propio en vez de utilizar el procedente del fabricante.

2.3. Driver UART

La primera capa del bajo nivel de la estructura del software es el driver que controla el periférico de la UART y que se adapta a los requisitos impuestos por los motores AX-12A.

Para el diseño de esta capa se estudió el manual de referencia del microcontrolador STM32F446RE (ST Microelectronics, 2017a), gracias a este manual y al manual de la HAL (ST Microelectronics, 2017b) se configuró e inicializó correctamente el periférico. Entrando más en detalle, se requería para inicializar la UART en semi-duplex que una serie de registros estuvieran vacíos y que el control de las interrupciones tuviera que implementarse por software. En nuestro caso, semi-duplex significa que solo existe un cable entre los dos periféricos UART, que está conectado al transmisor de cada uno y, aunque solo exista un cable, se puede utilizar la trasmisión y recepción por igual. La única diferencia es que internamente las dos están unidas al mismo pin de salida, por ello, se debe hacer una distinción por software de cuándo se va a usar cada tarea, activando o desactivando la interrupción asociada. En la Figura 2.4 se refleja el esquema seguido para el control de las tareas e interrupción del driver.

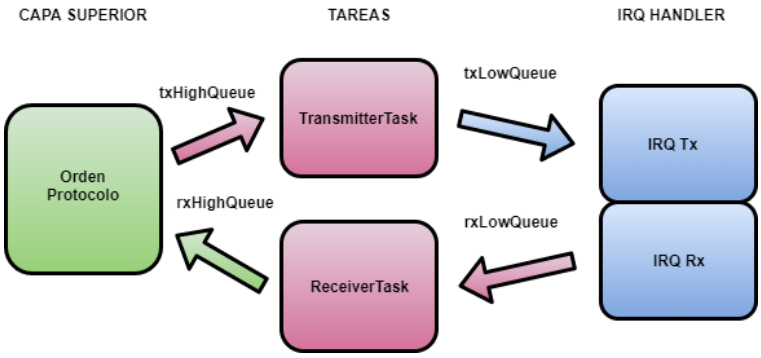

Figura 2.4: Flujograma de la tarea de transmisión.

En este esquema se distinguen varias partes, una capa superior, que se explicará en secciones posteriores, tareas, IRQ Handler y distintas flechas que comunican estas tres partes principales.

Las flechas son estructuras de colas que proporciona el sistema operativo para poder realizar comunicaciones entre distintas tareas en tiempo real. En este esquema tenemos 4 colas distintas que tienen estructuras diferentes debido a la información que transportan dentro de ellas. Después están las tareas, *TransmitterTask* y *ReceiverTask*, asociadas con las funciones de transmisión y recepción respectivamente.

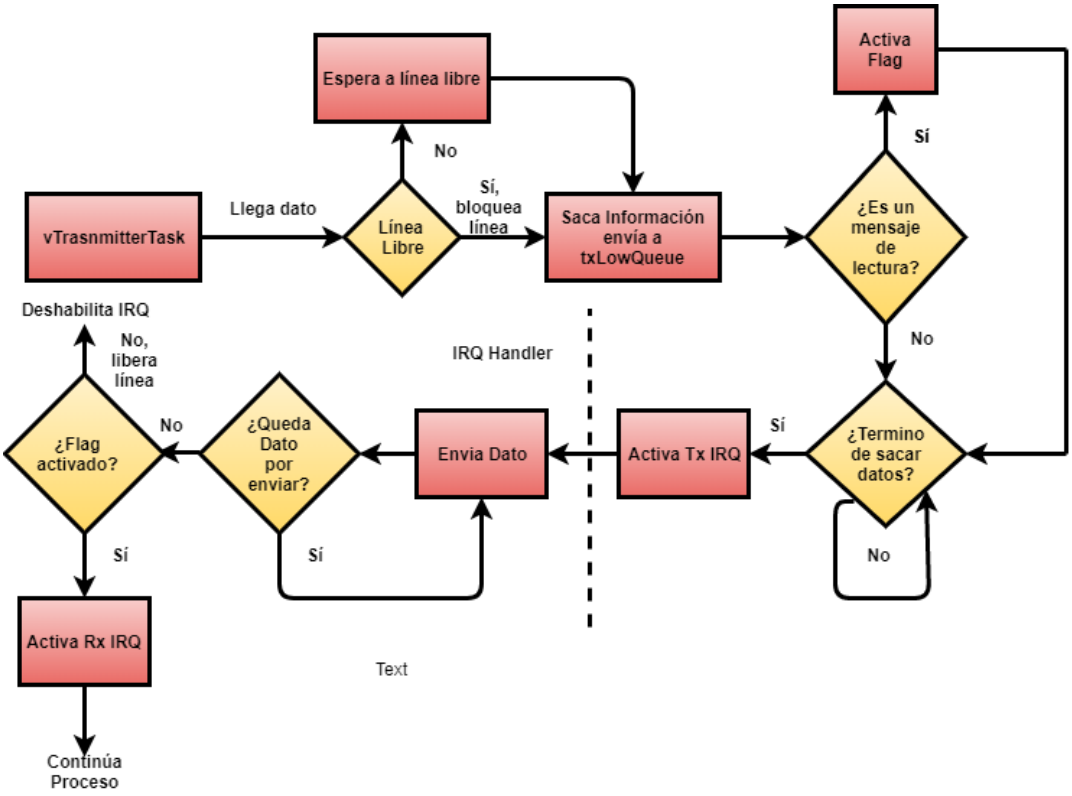

Figura 2.5: Flujograma de la tarea de transmisión.

A la derecha está el IRQ Handler, que es el manejador de interrupciones. Cuando la interrupción está activada realiza la función que tiene asociada. En el caso de la transmisión la interrupción saca los datos procedentes de la cola *txLowQueue* y los envía por la UART. En el caso de la recepción, coge los valores que llegan a la UART y los introduce en la cola *rxLowQueue*. Por último, está la orden de protocolo que pertenece a una abstracción superior, que se explica en la Sección 2.4.

En la parte de tareas se distingue *TransmitterTask* y *ReceiverTask*. Estas son las dos tareas encargadas de la transmisión y recepción de todo lo recibido por el periférico de la UART. Las dos tareas tienen la misma prioridad, por lo que, si las dos tienen que actuar a la vez, el sistema operativo saltará entre ellas dándoles el mismo tiempo de ejecución. En la Figura 2.5 se detalla el proceso entre la tarea de transmisión y la interrupción. La tarea está continuamente esperando la llegada de un dato por medio de la cola de *txHighQueue*. Cuando se produce este evento, la tarea comprueba que la línea de datos no está siendo usada y si se cumple comienza el proceso, en caso contrario, espera. Tras esto, la línea saca los datos procedentes de la cola y los va almacenando en la cola *txLowQueue*. En el quinto byte comprueba si es una orden de lectura, si lo es activa un flag. Cuando se han terminado de introducir todos los datos en la cola se activa la interrupción de transmisión. Aquí pasamos al dominio del manejador de interrupciones el cual se dedica a enviar todos los datos de la cola *txLowQueue*. Cuando ha acabado comprueba si el flag está activado, si no lo está libera la línea y desactiva la interrupción. En cambio, si está activado habilita la interrupción de recepción y el proceso continúa.

La recepción de un dato es más compleja. Esto es debido a que a priori no se conoce la longitud del paquete que se va a recibir, puesto que se pueden pedir datos al servo de distinto tamaño. Para solucionar este problema se implementó una máquina de estados en el manejador de interrupciones, que se detalla en la Figura 2.6.

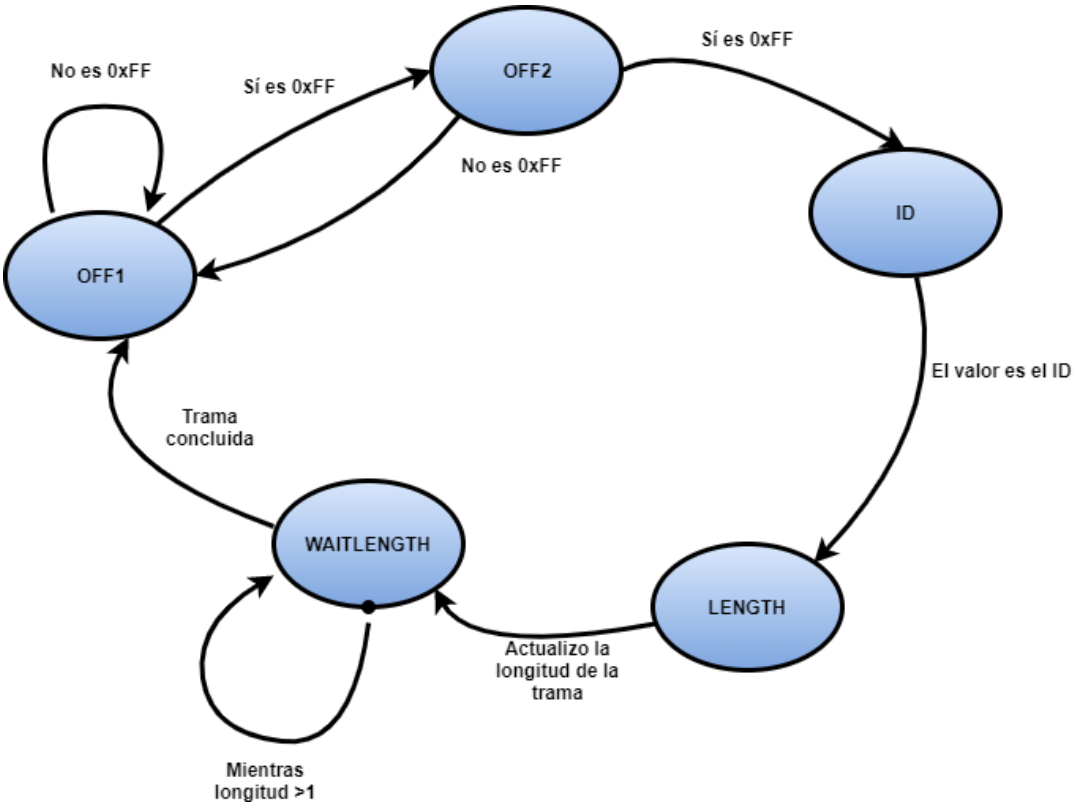

Figura 2.6: Máquina de estados de recepción.

Esta máquina de estados se encarga del control de flujo de información procedente del exterior para, una vez concluido el mensaje recibido, poder devolver el control de la línea de comunicación al sistema operativo. Los distintos estados de la máquina son en función del protocolo de comunicación de los servomotores, que se detalla en la Sección 2.4. La máquina, tras recibir correctamente los dos bytes de señalización, envía continuamente a la cola *rxQueueLow* los datos recibidos y en función del cuarto byte recibido, que es la longitud del paquete, continúa enviado estos bytes a la cola. Una vez acabado, vuelve al estado inicial y libera la cola.

En la tarea de recepción *ReceiverTask* encontramos la misma máquina de estados. La única diferencia es que esta máquina de estados se encarga de controlar la validez del paquete, encapsulándolo dentro de una estructura que será la recibida por las capas superiores. Además de agrupar el paquete, se encarga de controlar si éste ha llegado sin errores y advirtiéndolo en el encapsulado.

El acceso a estas tareas de transmisión y recepción se hace manipulando las colas

Parámetro	Descripción
Tamaño	Número de bytes usados de la estructura
Array de 29 bytes	Array con el máximo número de bytes posibles a usar

Tabla 2.1: Estructura de datos de *txQueueHigh*.

Parámetro	Descripción
ID	Identificador del servo que ha mandado el paquete
Paquete de Error	Tipo de error en el mensaje recibido
Datos	Los datos recibidos en el mensaje
CRC	Si el checksum del paquete es correcto o no

Tabla 2.2: Estructura de datos de *txQueueHigh*.

txQueueHigh y *rxQueueHigh*. Para proteger el código desarrollado y fomentar la modulación se crearon funciones para poder enviar y recibir las estructuras de datos que aceptan las colas. Solo estas funciones se incluyen en el fichero de las cabeceras permitiendo su uso libremente, con esto se pretende evitar un uso incorrecto de las tareas. Además, se incluye la función que crea las tareas y las colas para poder iniciar el controlador de este periférico.

Las estructuras de datos de las colas *txQueueHigh* y *rxQueueHigh* se detallan en las Tablas 2.1 y 2.2.

2.4. Driver Motores AX12A

Para explicar esta sección se debe entender previamente el funcionamiento de los motores AX-12A, puesto que esta capa junto con la anterior son las encargadas de la comunicación con ellos. Los motores AX-12A son servomotores digitales, un tipo de motor que puede ubicarse en cualquier posición dentro de su rango de operación y mantenerse estable en dicha posición[3]. El término digital significa que el motor tiene integrado un microcontrolador que es el que se encarga de su movimiento. En un motor DC la corriente llega a sus bornas y dependiendo de la intensidad el motor se mueve más rápido o más lento, en nuestro caso, nos comunicamos con el microcontrolador por un bus de comunicación y modificamos una serie de registros en la tabla de control del motor. Al cambiarse estos registros, el motor impone restricciones al motor, por ejemplo, si se sobre escribe el registro de la posición final, el microcontrolador mueve el motor a dicha posición abstrayendo del control de la posición y la velocidad. Los registros de control se encuentran en la Tabla A.1 del Apéndice A. En la Figura 2.7 se encuentra una imagen del motor y en la Tabla 2.3 se muestran sus especificaciones.

[3]https://es.wikipedia.org/wiki/Servomotor

Figura 2.7: Motor AX-12A de Robotis.

Característica	Descripción
Voltaje de Operación	12V
Torque	15Kg/cm
Velocidad	0.169seg/60º
Peso	565g
Resolución Temperatura	0.29ºC
Ángulo de operación	300º
Corriente Máxima	900mA
Corriente en espera	50mA
Protocolo	TTL Half Duplex Async Serial
Retroalimentación	Temperatura,Voltaje, Posición
Precisión	0-1023

Tabla 2.3: Especificación del servomotor AX-12A (Dynamixel, 2006)

El protocolo de comunicación de estos motores consta de dos partes, la primera es el paquete de instrucción, que es la primera trama enviada para comunicarse con los motores y después, si éste está activado, el servomotor contesta con el paquete de estado. El paquete de estado es de un tipo u otro dependiendo de la orden emitida por el paquete de instrucción.

En la Figura 2.8 se detalla el paquete de instrucción. Los dos primeros bytes son de señalización, el tercero es el ID que identifica al servo, si el paquete no coincide con el servo al que llega el mensaje éste no lo procesa y alcanza al resto de servos por el bus de comunicación. El cuarto es la longitud de los mensajes que quedan por llegar. El quinto es la orden de la instrucción y el resto de bytes son los parámetros de la instrucción y el último el checksum.

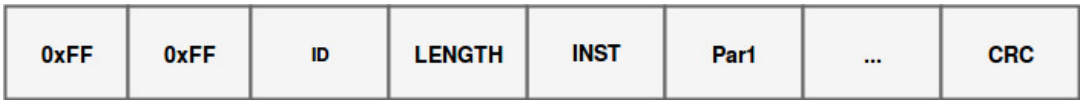

Figura 2.8: Paquete de Instrucción (Dynamixel, 2006)

Bit	Error	Detalles
Bit 7	–	Sin error asociado
Bit 6	Error de Instrucción	Se activa si una instrucción no está definida o una instrucción ACTION es enviada sin otra instrucción REG WRITE
Bit 5	Error de Exceso de Carga	Se activa a 1 si el máximo torque especificado es incapaz de soportar la carga aplicada
Bit 4	Error de CRC	Se activa a uno si el CRC del paquete de instrucción es incorrecto
Bit 3	Error de Rango	Se activa a uno si los parámetros enviados de una instrucción están fuera del rango definido
Bit 2	Error de Sobrecalentamiento	Se activa si los motores sobrepasan la temperatura definida en el registro de temperatura máxima
Bit 1	Error de Ángulo Límite	Se activa si el parámetro mandado a la posición objetivo esta fuera del rango de movimiento
Bit 0	Error de Voltaje de Entrada	Se activa si el voltaje esta fuera del rango de operación definido en la tabla de control

Tabla 2.4: Errores asociados al paquete de estado del AX-12A (Dynamixel, 2006)

En la Figura 2.9 se detalla el paquete de estado. Éste es muy parecido al anterior; los dos primeros son bytes de señalización, el tercero es el ID del servo y el cuarto es la longitud del paquete. Es en el quinto cuando se diferencian, pues este byte es el error que ha tenido el servo, los errores posibles se detallan en la Tabla 2.4. Después vienen los parámetros pedidos en el paquete de instrucción y, por el último, el checksum.

Las distintas órdenes que se pueden mandar al servomotor se detallan en la Tabla 2.5. El software implementado en esta capa es el encargado de construir esos determinados paquetes de instrucción. Debido a que los registros procedentes del servomotor tienen un tamaño máximo de 2 bytes, el número máximo de parámetros en estos paquetes es de 2. En los registros del servomotor, el Status Return Level, es el encargado de configurar al servomotor para devolver los paquetes de estado. Si este registro vale

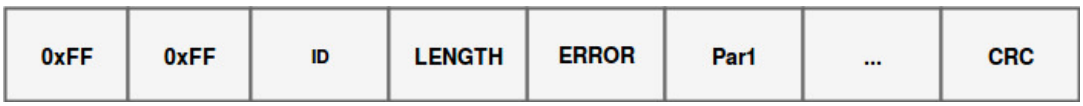

Figura 2.9: Paquete de Estado (Dynamixel, 2006)

Instrucción	Función	Valor
PING	Usado para obtener un paquete de estado del servomotor	0x01
READDATA	Se leen valores de la tabla de registros	0x02
WRITEDATA	Se escriben los valores en la tabla de registros	0x03
REGWRITE	Parecido a WRITE DATA pero solo se escriben cuando llega una orden ACTION	0x04
ACTION	Activa la orden REG WRITE	0x05
RESET	Devuelve los valores de la tabla de registros a su estado de fábrica	0x06
SYNCWRITE	Permite enviar la misma orden de WRITE DATA a todos los servos conectados a la vez	0x83

Tabla 2.5: Instrucciones del protocolo de transmisión (Dynamixel, 2006)

0, el servomotor nunca responde a ningún mensaje, si vale 1, responderá solo a los mensajes con orden READ y si vale 2, devolverá siempre un paquete de estado. En nuestro caso este control lo hacemos en la capa anterior con el flag de la tarea de transmisión. Esto es debido a que se ha tomado la decisión de poner este registro en los motores a 1, para solo tener un paquete de estado cuando se quiera recibir información.

El acoplamiento de esta librería con las capas inferiores se detalla en la Figura 2.10. En ella se muestra cómo el programa llama a cualquier método de control de los motores, pasándole como parámetro los valores deseados y éste los encapsula y los ingresa en las colas de acceso a la tarea de transmisión. Si el método de control no espera ningún parámetro de vuelta la orden termina, si en cambio requiere de valores de vuelta, espera hasta que las tareas de transmisión y recepción de la UART terminen.

Se crearon las funciones de la Tabla 2.6 encapsulando las órdenes del protocolo de los motores, estas funciones están adaptadas para poder acceder a un registro de 8 bits o de 16 dependiendo de a que partes de la tabla de control se quiera acceder.

Función	Orden	Características
ping	PING	Usada igual que la orden PING pero impidiendo el broadcast
getRegister	READ	Se hace un read a un registro de 8 bits
getRegister2	READ	Se hace un read a un registro de 16 bits
setRegister	WRITE	Se escribe en un registro de 8 bits
setRegister2	WRITE	Se escribe en un registro de 16 bits
action	ACTION	Se encapsula la orden ACTION
regWrite	REGWRITE	Se escribe un registro de 8 bits
regWrite2	REGWRITE	Se escribe un registro de 16 bits
syncWrite	SYNCWRITE	Se escribe a 7 servo motores un registro de 8 bits, con los valores que se pasen por parámetro
syncWrite2	SYNCWRITE	Se escribe a 16 servo motores un registro de 8 bits, con los valores que se pasen por parámetro
reset	RESET	Se encapsula la orden de reset

Tabla 2.6: Funciones de la capa de abstracción Driver Motores.

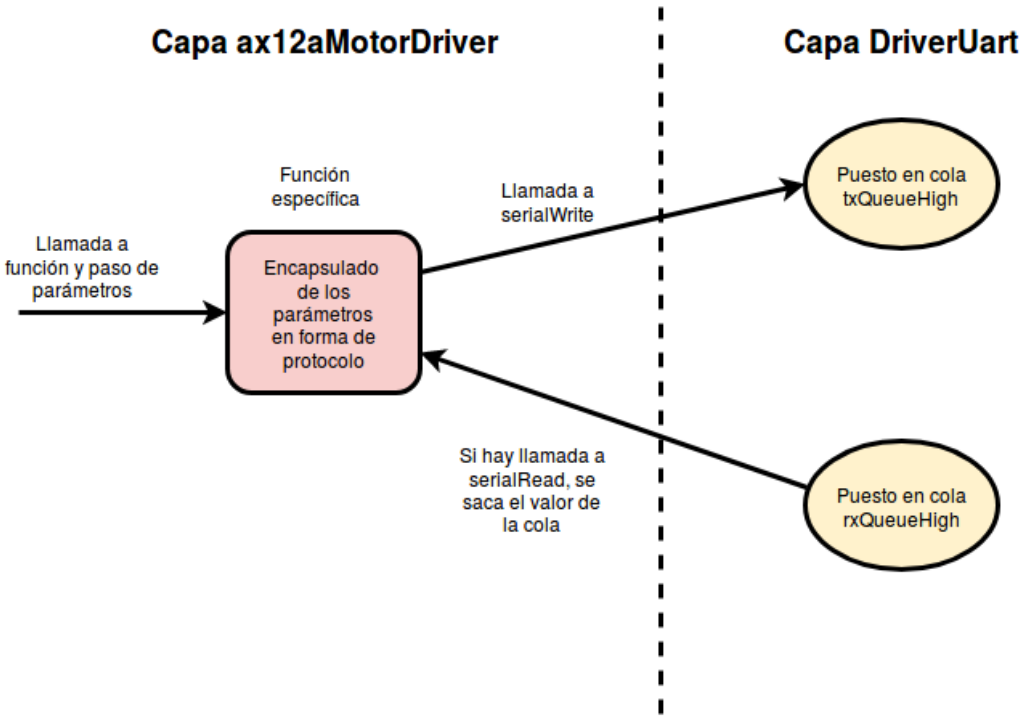

Figura 2.10: Comunicación capas de bajo nivel

Capítulo 3

Control de la Cinemática del Robot

3.1. Modelado Matemático del Robot

Como se ha explicado anteriormente, un cuerpo rígido en el espacio se puede representar completamente con 6 grados de libertad: tres para la posición y 3 para la orientación. Aun así, el control de un robot es muy complejo y se tiene que enfrentar a diversos problemas; en este Trabajo solo se han solucionado e implementado tres problemas: El problema del cálculo de la posición en el espacio del efector final, el problema del cálculo de las coordenadas generalizadas de cada articulación para un punto en el espacio del efector final y el cálculo de la trayectoria que debe seguir el efector final para moverse de un punto a otro del espacio. Estos tres problemas se conocen como: ecuaciones cinemáticas directas (Álvaro Gutiérrez Martín, Félix Monasterio-Huelin, 2017a), ecuaciones cinemáticas inversas y cálculo de trayectorias (Álvaro Gutiérrez Martín, Félix Monasterio-Huelin, 2017b), respectivamente.

El concepto de grado de libertad se asocia a la capacidad de movimiento de un mecanismo, donde un mecanismo está compuesto por un conjunto de cuerpos rígidos conectados por algún tipo de articulación, creando lo que se conoce como una cadena cinemática de cuerpos rígidos. Existen dos tipos de cadenas cinemáticas (ver Figuras 3.1 y 3.2) las abiertas y las cerradas. Los mecanismos en serie son los formados por cadenas cinemáticas abiertas y los mecanismos en paralelo, son los formados por, al menos, una cadena cerrada. Formando estas cadenas cinemáticas están los pares cinemáticos, conexiones entre dos cuerpos rígidos que imponen una restricción en el movimiento del mecanismo y se definen por el número de grados de libertad que posee. En este Trabajo Fin de Grado el robot es un mecanismo en serie, al solo poseer cadenas cinemáticas abiertas y cada par cinemático que tiene otorga un grado de libertad más al mecanismo, siendo un total de 5.

La movilidad de un mecanismo es un concepto cinemático que indica la capacidad de realizar movimientos. El número de grados de libertad es el mínimo número de variables cinemáticas independientes que permiten al mecanismo moverse. Cada grado de libertad es asociado a una articulación $q_i, i = 1, ..., n$ y se las conoce como

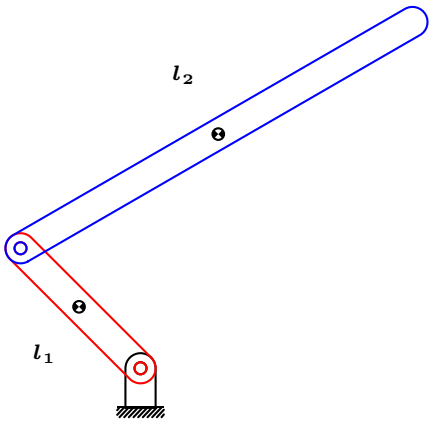

Figura 3.1: Cadena cinemática abierta.

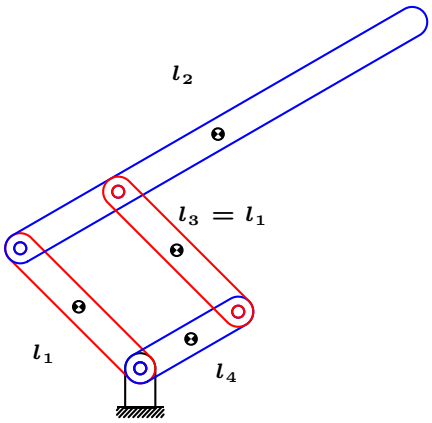

Figura 3.2: Cadena cinemática cerrada.

coordenadas generalizadas o coordenadas de la articulación. Estas coordenadas están definidas en el espacio de trabajo del mecanismo, cuya dimensión es el número de grados de libertad.

En la Figura 3.3 se representan los grados de libertad del robot y en la Figura 3.4 los ejes de coordenadas locales elegidos para su posterior modelización. En la Tabla 3.1 se muestra las modelización de las articulaciones del robot, todas son rotacionales y se presentan sus valores límite. En la Tabla 3.2 se muestran las medidas de los segmentos del robot. Estos segmentos son las uniones entre cada articulación. Como se ve en las tablas, las articulaciones las denotamos con q_i, con cada subíndice i para cada articulación. Cada segmento lo denotamos con l_i siendo la distancia entre las articulaciones q_{i-1} y q_i y l_0 la distancia del anclaje del suelo con la primera articulación.

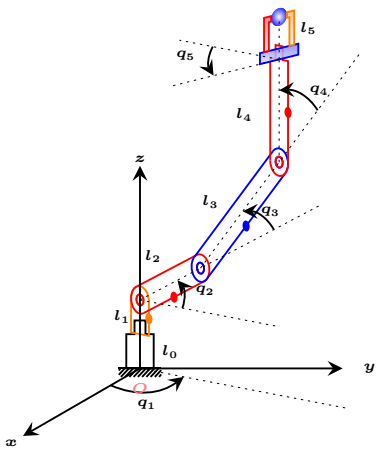

Figura 3.3: Representación de los grados de libertad.

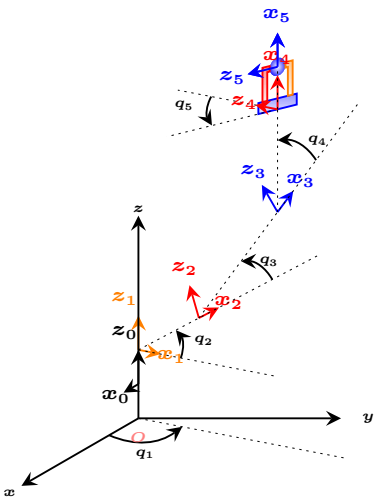

Figura 3.4: Sistemas de coordenadas locales del robot.

Rotación	Mínimo(rad)	Máximo(rad)
q_1	-2.62	2.62
q_2	-0.33	2.97
q_3	-2.98	0.26
q_4	-1.83	1.86
q_5	-2.62	2.62

Tabla 3.1: Limitaciones mecánicas.

Segmento	Medida (mm)
l_0	86.8
l_1	31.0
l_2	150.2
l_3	146.3
l_4	70.0
l_5	66.3

Tabla 3.2: Medidas de cada segmento.

3.1.1. Ecuaciones Cinemáticas Directas

Todo punto P en el espacio de tres dimensiones puede ser representado por un sistema de coordenadas referenciado a otro sistema de coordenadas. Además, un sistema de coordenadas puede ser representado en función de otro sistema de coordenadas en términos de rotación de sus coordenadas o translación de estas. Entonces, si las coordenadas de un punto P referenciado a un sistema de coordenadas S_1 son conocidas, es posible obtener las coordenadas de ese punto según el sistema S_0, si la matriz de rotación R_0^1 y la de translación T_0^1 entre los dos sistemas de coordenadas son también conocidos:

$$p_0 = R_0^1 p_1 + T_0^1 \tag{3.1}$$

donde p_0 representa el punto referenciado al sistema S_0 y p_1 al sistema S_1 y las matrices R_0^1 y T_0^1 son el vector de rotación o translación respectivamente representados en el sistema S_0.

En la Figura 3.3 se representan los sistemas de referencias locales S_i de cada segmento del robot, donde el sistema S_0 es el sistema de coordenadas de referencia, en el cual está anclado el robot. Se define el punto Q como el efector final de nuestro robot y el conjunto de ejes de rotación U de cada articulación como:

$$U = \{z_0, -y_1, -y_2, -y_3, x_4\} \tag{3.2}$$

El origen o_i de cada sistema de referencia local S_i está localizado en la articulación que une los elementos $(i, i+1)$ (ver Figura 3.5). La longitud de cada segmento (l_i) la asociamos a la norma del vector $l_i = |\overrightarrow{o_{i-1}o_i}|$

Se define además el vector d_{i-1}^i en el sistema de referencia S_{i-1} como :

$$d_{i-1}^i = \overrightarrow{o_{i-1}o_i} \tag{3.3}$$

Entonces, definimos el conjunto $O = \{o_1, o_2, o_3, o_4, o_5\}$

$$O = \{(l_0 + l_1)z, l_2x_1, l_3x_2, l_4x_3, l_5x_5\} \tag{3.4}$$

De acuerdo con este convenio:

$$d_{i-1}^i = R_{i-1}^i \vec{o_i} \tag{3.5}$$

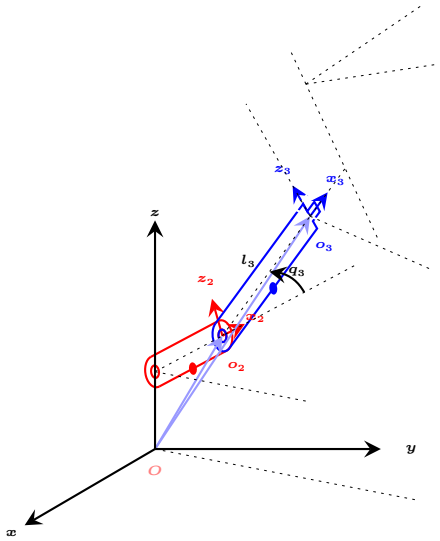

Figura 3.5: Sistemas de referencia locales sobre el par cinemático (2,3).

Por último, se definen las matrices de rotación de cada eje de coordenadas:

$$R_0^1 = R_{z,q_1} = \begin{bmatrix} \cos q_1 & -\sin q_1 & 0 \\ \sin q_1 & \cos q_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

$$R_1^2 = R_{-y_1,q_2} = \begin{bmatrix} \cos q_2 & 0 & -\sin q_2 \\ 0 & 1 & 0 \\ \sin q_2 & 0 & \cos q_2 \end{bmatrix} \tag{3.7}$$

$$R_2^3 = R_{-y_2,q_3} = \begin{bmatrix} \cos q_3 & 0 & -\sin q_3 \\ 0 & 1 & 0 \\ \sin q_3 & 0 & \cos q_3 \end{bmatrix} \tag{3.8}$$

$$R_3^4 = R_{-y_3,q_4} = \begin{bmatrix} \cos q_4 & 0 & -\sin q_4 \\ 0 & 1 & 0 \\ \sin q_4 & 0 & \cos q_4 \end{bmatrix} \tag{3.9}$$

$$R_4^5 = R_{x_4,q_5} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos q_5 & -\sin q_1 \\ 0 & \sin q_1 & \cos q_5 \end{bmatrix} \tag{3.10}$$

Con estas definiciones y a partir de las ecuaciones fundamentales de la cinemática directa de posición 3.11 y de orientación 3.12 se puede obtener la posición el punto Q y la orientación del elemento final representados en el sistema S_0.

$$d_0^i = d_0^{i-1} + R_0^{i-1} d_{i-1}^i \tag{3.11}$$

$$R_0^i = R_0^{i-1} R_{i-1}^i \tag{3.12}$$

Por lo que las ecuaciones cinemáticas directas del robot de este Trabajo Fin de Grado, son:

$$d_0^5 = (l_0 + l_1)\begin{bmatrix}0\\0\\1\end{bmatrix} + l_2\begin{bmatrix}\cos q_1 \cos q_2\\ \sin q_1 \cos q_2\\ \sin q_2\end{bmatrix} + l_3\begin{bmatrix}\cos q_1 \cos q_{23}\\ \sin q_1 \cos q_{23}\\ \sin q_{23}\end{bmatrix} + (l_4 + l_5)\begin{bmatrix}\cos q_1 \cos q_{234}\\ \sin q_1 \cos q_{234}\\ \sin q_{234}\end{bmatrix} \tag{3.13}$$

y las de orientación:

$$R_0^5 = \begin{bmatrix}\cos q_1 \cos q_{234} & -\sin q_1 \cos q_5 - \cos q_1 \sin q_{234} \sin q_5 & -\sin q_1 \sin q_5 - \cos q_1 \sin q_{234} \cos q_5\\ \sin q_1 \cos q_{234} & \cos q_1 \cos q_5 - \sin q_1 \sin q_{234} \sin q_5 & -\cos q_1 \sin q_5 - \sin q_1 \sin q_{234} \cos q_5\\ \sin q_{234} & \cos q_{234} \sin q_5 & \cos q_{234} \cos q_5\end{bmatrix} \tag{3.14}$$

donde:

$$\begin{aligned} q_{23} &= q_2 + q_3\\ q_{234} &= q_2 + q_3 + q_4 \end{aligned} \tag{3.15}$$

3.1.2. Ecuaciones Cinemáticas Inversas

El cálculo de la cinemática inversa permite obtener las coordenadas generalizadas de cada articulación a partir de la posición y orientación del efector final o punto Q.

Antes de comenzar a explicar el cálculo de la cinemática inversa, se debe definir la orientación del efector final. Es común definir esta orientación con tres vectores ortonormales $\{\vec{a}, \vec{n}, \vec{s}\}$ que se llaman vector avance, vector normal y vector deslizamiento, donde:

$$\begin{aligned} \vec{a} &= \begin{bmatrix}a_x\\a_y\\a_z\end{bmatrix}\\ \vec{n} &= \begin{bmatrix}n_x\\n_y\\n_z\end{bmatrix}\\ \vec{s} &= \begin{bmatrix}s_x\\s_y\\s_z\end{bmatrix} \end{aligned} \tag{3.16}$$

La Figura 3.6 muestra la orientación del efector final en el espacio. La orientación de la matriz R_{S_O} respecto al sistema de referencia, se representa de distintas formas dependiendo de la definición del vector de avance. En nuestro caso, el vector de avance coincide con el eje X, por lo que se representa:

$$R_{S_O} = \begin{bmatrix}a_x & -n_x & -s_x\\ a_y & -n_y & -s_y\\ a_z & -n_z & -s_z\end{bmatrix} \tag{3.17}$$

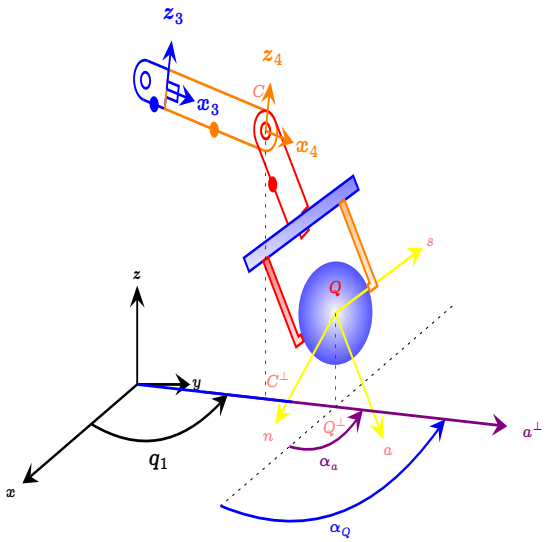

Figura 3.6: Orientación de la pinza.

En la resolución de la cinemática inversa se usó la técnica del desacoplo cinemático. Esto es posible gracias a que las tres últimas articulaciones intersecan en un punto, permitiendo este desacoplo de posición y orientación. Al punto de intersección de estas tres últimas articulaciones se le denomina centro de la muñeca. Esto nos permite resolver la cinemática inversa de la posición y de la orientación independientemente. En la Figura 3.7 se muestra el manipulador donde Q representa la posición y orientación $\{\vec{a}, \vec{n}, \vec{s}\}$ de la pinza.

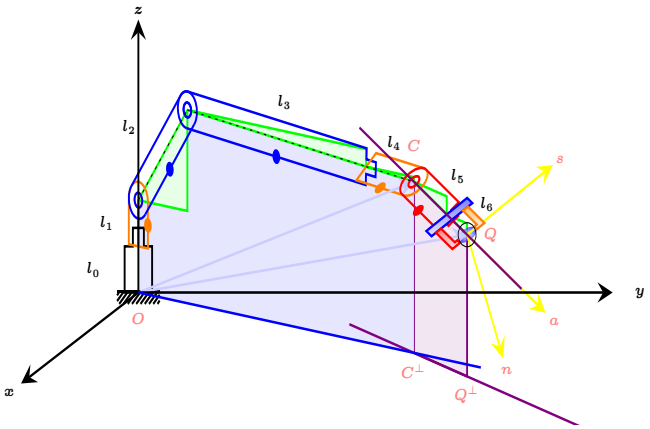

Figura 3.7: Desacoplo cinemático.

Para ello se siguen estos pasos:

- Obtención del centro de la muñeca, cuando el vector de aproximamiento $\vec{a}$, el punto de final de la mano y la longitud de la muñeca $(l_4 + l_5)$ son conocidas. El

punto C puede ser definido como:

$$\overrightarrow{OC} = \overrightarrow{OQ} - (l_4 + l_5)\vec{a} \tag{3.18}$$

- Resolver el problema de la cinemática inversa de la posición de C para obtener $\{q_1, q_2, q_3\}$

  Esto supone resolver este sistema de ecuaciones:

$$\begin{aligned} C_x &= \cos q_1(l_2 \cos q_2 + l_3 \cos q_{23}) \\ C_y &= \sin q_1(l_2 \cos q_2 + l_3 \cos q_{23}) \\ C_z &= (l_0 + l_1) + l_2 \sin q_2 + l_3 \sin q_{23} \end{aligned} \tag{3.19}$$

  De donde se obtiene:

$$\begin{aligned} q_1 &= \arctan \frac{C_y}{C_x} \\ q_2 &= -\beta + \arctan \frac{L}{+-\sqrt{1-L^2}} \end{aligned} \tag{3.20}$$

$$\beta = \arctan \frac{\frac{C_x}{\cos q_1}}{C_x - (l_0 + l_1)} \tag{3.21}$$

$$L = \frac{l_2^2 + (\frac{C_x}{\cos q_1})^2 + (C_z - (l_0 + l_1))^2 - l_3^2}{2l_2\sqrt{(\frac{C_x}{\cos q_1})^2 + (C_z - (l_0 + l_1))^2}}$$

$$q_3 = \arctan \frac{C_z + (l_0 + l_1) - l_2 \sin q_2}{\frac{C_x}{\cos q_1} - l_2 \cos q_2} \tag{3.22}$$

- Cálculo de la matriz de rotación $R_0^3 = R_0^1 R_1^2 R_2^3$ que se llama R_{S_A}

  Las matrices R_0^1,R_1^2 y R_2^3 se han presentado en la Sección 3.1.1 y la obtención de R_{S_A} se consigue tras realizar la multiplicación consecutiva de las tres matrices, obteniendo:

$$R_{S_A} = \begin{bmatrix} \cos q_1 \cos q_{23} & -\sin q_1 & -\cos q_1 \sin q_{23} \\ \sin q_1 \cos q_{23} & \cos q_1 & -\sin q_1 \sin q_{23} \\ \sin q_{23} & 0 & \cos q_{23} \end{bmatrix} \tag{3.23}$$

- Calcular la matriz R_{S_H} cuando la orientación deseada del manipulador R_{S_O} y R_{S_A} son conocidas

  El cálculo de la matriz $R_{S_H} = R_{S_A}^T R_{S_0}$ donde se define la matriz $R_{S_A}^T$ como la matriz traspuesta de la matriz R_{S_A} y la matriz R_{S_O} como una matriz de rotación que depende de cómo se haya definido el eje de avance de la mano:

$$R_{S_O} = \begin{bmatrix} a_x & -n_x & -s_x \\ a_y & -n_y & -s_y \\ a_z & -n_z & -s_z \end{bmatrix} \tag{3.24}$$

- Resolver el problema de la cinemática inversa de orientación R_3^5 para obtener $\{q_4, q_5\}$

  Considerando la matriz R_{S_H} como:

$$R_{S_H} = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \tag{3.25}$$

Se pueden obtener $\{q_4, q_5\}$ como:

$$q_4 = \arctan \frac{-a_x \cos q1 \sin q_{23} - a_y \sin q_1 \sin q_{23} + a_z \cos q_{23}}{} \tag{3.26}$$

$$q_5 = \arctan \frac{s_y \cos q_1 - s_x \sin q_1}{n_x \sin q_1 - n_y \cos q_1} \tag{3.27}$$

3.1.3. Calculo de Trayectorias

El cálculo de trayectorias se define como la representación de las coordenadas generalizadas de las articulaciones como funciones del tiempo $q_i(t) \forall i = 1, 2, ..., n$.

Los manipuladores robóticos son mecanismos complejos que poseen actuadores que permiten implementar los movimientos. El brazo de este Trabajo Fin de Grado posee en cada motor de las articulaciones un sistema en lazo cerrado para el control de los movimientos que se les soliciten. Este sistema en lazo cerrado es como el que se detalla en la Figura 3.8

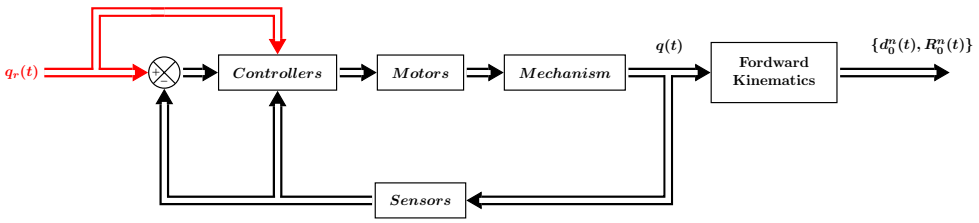

Figura 3.8: Representación de un sistema en lazo cerrado.

En esta Sección se plantea el problema de las trayectorias de cada una de las articulaciones en el tiempo con el objetivo de obtener un movimiento suave del robot y de su efector final. Para esto se deben considerar las posiciones de cada articulación como funciones en el tiempo de la forma $q_i(t) = a_{io} + a_{i1}t + a_{i2}t^2 + ... + a_{in}t^n$ donde en nuestro caso $n <= 3$.

Hay varias formas de solucionar este problema, en nuestro caso se definen las trayectorias del robot en el espacio de las articulaciones. Esto nos permite obligar a que la trayectoria pase a través de unas posiciones y orientaciones predefinidas.

El cálculo de las trayectorias del robot se obtiene según distintos requisitos, estos se enumeran con la solución matemática a las ecuaciones planteadas (Álvaro Gutiérrez Martín, Félix Monasterio-Huelin, 2017b):

- Trayectoria cúbica con puntos inicial y final definidos y con velocidad inicial y final cero.

  Se obtiene esta solución al problema en el tiempo continuo:

$$q_i(t) = q_i(0) + 3\alpha_i t_f t^2 - 2\alpha_i t^3 \tag{3.28}$$

Donde:

$$\alpha_i = \frac{q_i(t_f) - q_i(0)}{t_f^3} \tag{3.29}$$

Pero para poder ser usada en un ordenador como es nuestro caso, el tiempo continuo debe de ser discretizado $t = KT$.
Si además consideramos $t_f = NT$ obtenemos:

$$q_i[k] = q_i[0] + \beta_i(3N - 2(k+1))(k+1)^2, k \in 1, 2, ..., N \tag{3.30}$$

Donde:

$$\beta_i = \frac{q_i[N] - q_i[0]}{N^3} \tag{3.31}$$

- Trayectoria cúbica con puntos inicial, intermedio y final conocidos y con la continuidad de la velocidad y la aceleración en el punto intermedio.

$$q_i[k] = \{ \begin{array}{l} q_i[0] + a'_{i2}(k+1)^2 a'_{i3}(k+1)^3 \, , k \in \{1, 2, ..., N1\} \\ q_i[N] + b'_{i2}(N-k-1)^2 + b'_{i3}(N-K-1)^3 \, , k \in \{N1+1, N1+2, ..., N\} \end{array} \tag{3.32}$$

Donde:

$$\begin{array}{l} a'_{i3} = -\frac{\beta_{i1} + \beta_{i2} N_2}{N N_1} \\ b'_{i3} = -\frac{\beta_{i1} - \beta_{i2} N_2}{N N_2} \\ a'_{i2} = \frac{3}{2} \frac{\beta_{i1} + 2\beta_{i2} N_2}{N} \\ b'_{i2} = \frac{3}{2} \frac{\beta_{i1} - 2\beta_{i2} N_2}{N} \end{array} \tag{3.33}$$

Y

$$\begin{array}{l} \beta_{i1} = \frac{2(q_i[N_1] - q_i[0])}{N1} + \frac{2(q_i[N_1] - q_i[N_1+N_2])}{N_2} \\ \beta_{i2} = \frac{q_i[N_1] - q_i[0]}{2N_1^2} - \frac{q_i[N_1] - q_i[N_1+N_2]}{2N_2^2} \end{array} \tag{3.34}$$

3.2. Driver Control Robot

Este driver busca modelar el robot como un sistema de cara a las capas superiores. Permite abstraerse de las limitaciones de los motores o del número de ellos permitiendo manipular el concepto de grado de libertad del brazo. Además, permite acceder a todos los registros de la tabla de control de los servo motores, para sacar la información de ellos o para modificarlos.

Para el control del movimiento del robot, se buscó la abstracción de los motores del brazo robótico por el concepto de grados de libertad. El PhantomX posee en total 8 motores, 7 de los cuales generan los 5 grados de libertad del brazo y el octavo es el encargado del efector final, la pinza, como se explica en la Figura 3.9.

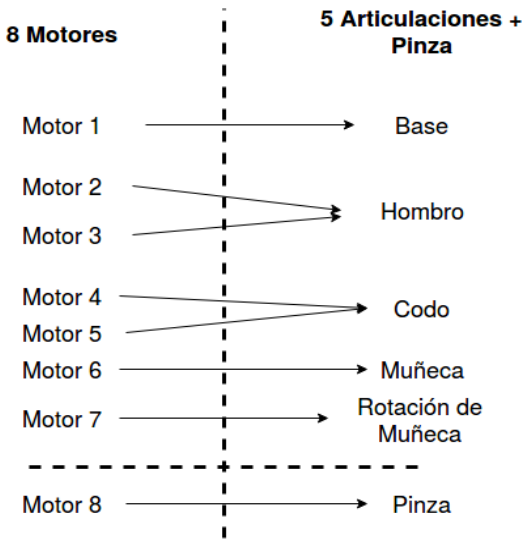

Figura 3.9: Abstracción implementada en el robot

Los motores están conectados en serie como se aprecia en la Figura 3.10 y como se explicó en la Sección 2.4, cuando un paquete no es para un servo este no lo procesa y llega el resto de servos por el bus de comunicación. Esto permite controlar todos los motores con un solo bus de comunicación. Por ello, en la librería implementada existen diferentes formas de acceder al control del robot. Por una parte, se puede generar un paquete para cada motor y enviarlos por separado o utilizar la orden SYNC WRITE que permite enviar en el mismo paquete una misma información para todos los servos.

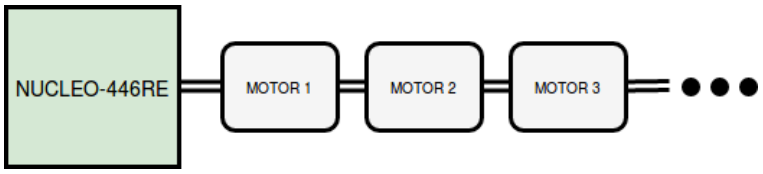

Figura 3.10: Conexión en serie motores AX-12A.

Por otra parte, el cálculo de las coordenadas generalizadas de los motores se realiza en números reales $\mathscr{R}$, pero los servo motores solo tienen un control de posición de 1024 pulsos de precisión, como se menciona en la Tabla 2.3 del Apéndice A. Para ello, se realiza un control de los parámetros proporcionados a las funciones de control de la posición de los servo motores. Este control se detalla en la Figura 3.11 y, en ella, se puede ver primero que solo se aceptan datos en tipo *double* , que es la representación de los números reales en C, después se modifican los valores a los límites de cada motor. Estos límites son los mismo de la Tabla 3.1 pero convertidos en los pulsos de los servo motores, que se detallan en la Tabla 3.3. Como se puede apreciar, los límites no son iguales al máximo movimiento de los servo motores, esto es debido a que el robot, por el diseño que tiene, no puede alcanzar su máximo rango de movimientos.

Motor	Mínimo(Pulsos)	Máximo(Pulsos)
M_1	0	1023
M_2	191	836
M_3	191	836
M_4	118	819
M_5	209	903
M_6	303	1023
M_7	0	1023
M_8	140	512

Tabla 3.3: Limitaciones de cada servomotor según el diseño del Robot.

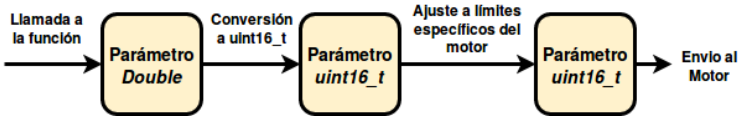

Figura 3.11: Conversión de la información proporcionada a los motores.

Para evitar posibles interferencias en el bus de comunicación, se manda la información a varios servos a la vez. Esto es una función implementada del protocolo de los servo motores (Sección 2.4, Tabla 2.5) que permite enviar la misma orden WRITE a todos los servo motores con información específica para cada uno. Por ello, es posible controlar cada articulación por separado o todas en conjunto a la vez. Esta última forma es la utilizada en el resto de capas del software, debido a la facilidad de empaquetamiento de la información, a la velocidad de actuación de los motores y al menor número de errores por solapamiento en el bus de comunicación.

Esta capa de abstracción también es la encargada del control de los registros de control de todos los servomotores del robot como sistema. Para ello se implementó una librería que posibilita el acceso a todos los registros de la Tabla A.1, con funciones de lectura y de escritura o solo de lectura, dependiendo del tipo de acceso que tiene el registro. Con esto, es posible realizar una depuración versátil de todos los motores del robot. Además, se implementó una función para inicializar todos los motores ajustando ciertos parámetros, como son: la velocidad de los motores, el ajuste del StatusReturnLevel a 0, la habilitación del torque de los motores y el límite de temperatura soportado.

Durante el desarrollo de esta capa software se incluyó el mapeo de todas las tablas de control del robot. Con esto, se consiguió depurar diversos problemas asociados al exceso de temperatura del robot, puesto que las alarmas de los robots saltaban al superar dichos valores. Además, permitió la modificación de los valores de control de la velocidad del robot, esto es debido a que los motores incluyen un controlador de velocidad del cual no hacen uso hasta su habilitación. La habilitación de este

controlador elimina muchos errores en el controlador de posición del motor y, si el parámetro no está cerca del límite del rango, no produce limitaciones en el bus de comunicación, por la falta de tiempo de respuesta del microcontrolador del motor.

3.3. Driver Trayectorias

Este driver es el encargado de implementar los métodos matemáticos para el cálculo de trayectorias (Sección 3.1.3). Hay dos tipos de trayectorias implementadas: la primera tiene un punto inicial y otro final y la segunda incluye, además, un punto intermedio (Álvaro Gutiérrez Martín, Félix Monasterio-Huelin, 2017b).

El software desarrollado consta de tres funciones y una tarea. En estas funciones, una es la encargada de crear la tarea, un semáforo y una cola. Esto es debido a que las otras dos funciones son las encargadas de realizar los dos tipos de trayectorias. Para ello, estas tareas se encargan de pedir los distintos parámetros para el cálculo de la trayectoria y después calculan los parámetros β o α que se necesitan. Una vez hecho esto, pasan estos valores por medio de una cola de estructuras a la tarea, la cual es la encargada de mandar los valores de la curva discretizada por donde deben pasar cada articulación del robot, en los instantes de tiempo discretizados.

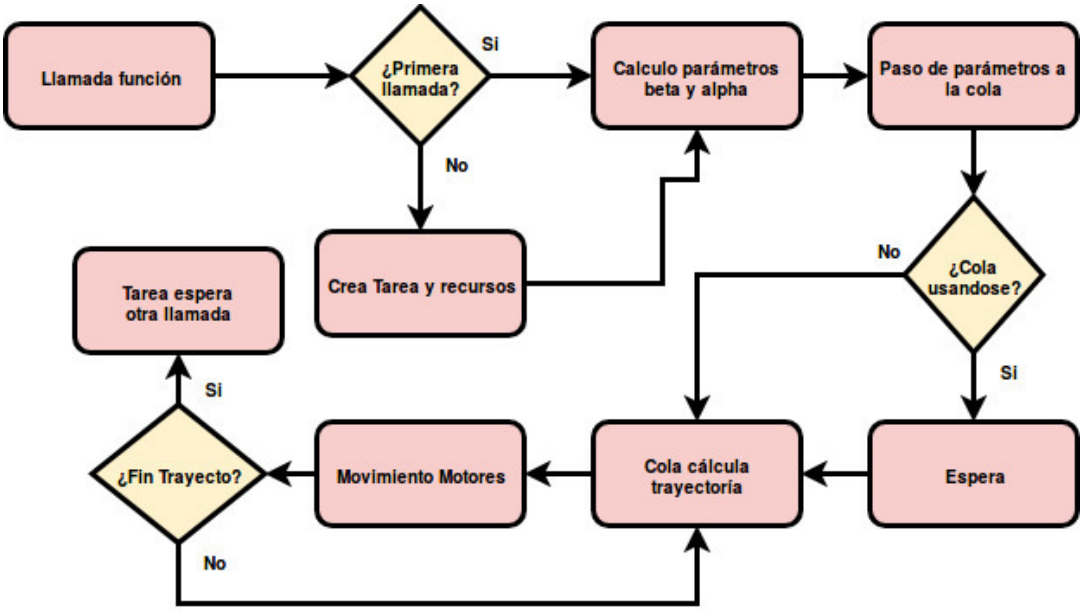

Figura 3.12: Flujograma del software de control de la trayectoria.

En la Figura 3.12 se muestra en detalle lo mencionado previamente. En ella se aprecia cómo después de realizar la llamada a la función se comprueba si es la primera vez que se ha llamado a esta función y entonces se crea la tarea encargada del cálculo de las trayectorias. Una vez hecho se procede al cómputo de los parámetros β y α y se envían a la cola de la tarea. Antes de enviarse se comprueba que la tarea no está

siendo utilizada y, si lo está, se espera hasta que acabe. Una vez que la tarea recoge los valores de la cola, se encarga de mandar las posiciones a los motores en cada instante de tiempo y cuando ha terminado espera otra cola.

La tarea es periódica, de periodo 30ms. Para ello entre los valores que se le pasan por la cola de estructuras, se incluye un parámetro del tiempo que tiene que durar la trayectoria. Con este parámetro se obtienen todos los instantes de tiempo discretizados en los que se va a mandar la posición al robot. En este caso existen dos formas de solucionar este problema, una es realizar el cálculo de todas las posiciones antes de iniciar la tarea e ir enviándolas en los instantes de tiempo precisos o realizar los cálculos de la siguiente trayectoria entre cada envió de posiciones. En nuestro caso, se optó por la segunda solución debido a la suficiente potencia de cómputo del microcontrolador de la tarjeta. La elección del periodo de 30ms se eligió para evitar problemas de solapamiento en el bus de comunicación, puesto que en algunos casos los motores no eran capaces de procesar una orden antes de recibir la siguiente.

Con este driver se consigue la planificación de la trayectoria del efector final. Como hemos comentado, la diferencia entre las dos trayectorias implementadas es que en la primera se mueve el efector final de un punto inicial a otro final y en la segunda se exige el paso por un punto intermedio.

El desarrollo de esta capa provocó ciertos problemas de sincronía entre las tareas. Se debe de tener cuidado puesto que los límites de memoria de las colas de comunicación provocan la interrupción del programa. Si se envían demasiadas peticiones de trayectorias, el brazo deja de responder al no poder el software realizar todas las peticiones. Este problema que no se consiguió depurar y se soluciona esperando el tiempo necesario entre cada envío de trayectorias.

Aún con los problemas encontrados, muchos proceden de los motores AX-12A. Controlando el parámetro de la velocidad máxima en la tabla de control de los motores, se puede garantizar un movimiento correcto y fluido de la trayectoria del brazo robótico. Si este parámetro no se controla, es posible que el brazo no consiga llegar al punto deseado de la forma esperada, e incluso que no llegue al final por algún error de sobrecalentamiento.

3.4. Driver Comunicación

Este driver es el encargado de gestionar las órdenes recibidas para el robot. Para ello se diseñó e implementó un protocolo de comunicación. Éste se utiliza mediante el periférico UART6, para no solaparse con la otra UART2 utilizada para la comunicación con los servo motores. Aquí se implementó como una UART full-duplex, debido a que la comunicación es bidireccional y la implementación y manejo del periférico es de menor complejidad que semi-duplex.

Este protocolo de comunicación, además de utilizar el periférico de la UART6, utiliza una tarea que es la encargada de gestionar la información recibida. En la Figura 3.13

se muestra el esquema general de funcionamiento de esta capa.

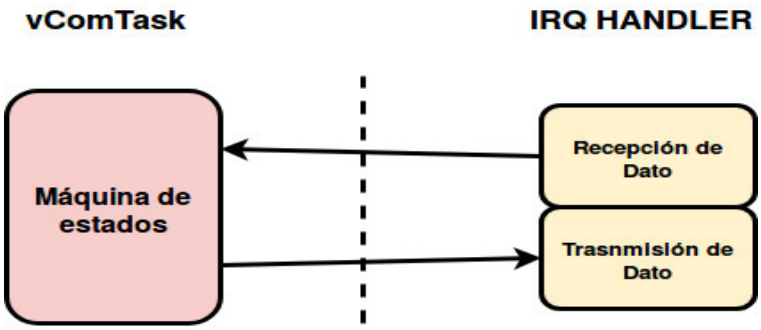

Figura 3.13: Esquema de funcionamiento.

El manejador de interrupciones de la UART6, es más sencillo que el de la UART2, puesto que toda la información recibida la incluye directamente en la cola de comunicación con la tarea. No realiza ningún tipo de procesado de la información puesto que no debe de liberar la línea de transmisión tras su uso. Cuando se trata de enviar información es idéntico, se saca la información de la cola y se envía por el periférico.

El protocolo de comunicación se muestra en la Figura 3.14. Los dos primeros bytes son de señalización, el tercero es el identificador, que en nuestro caso es siempre 1, el cuarto es la longitud de parámetros, el quinto es el comando y los siguientes son el número de parámetros y el checksum.

Figura 3.14: Protocolo de comunicación.

En nuestro caso, la longitud puesta siempre es N + 2, donde el valor 2 procede del comando y del checksum y el valor de N se cuenta por duplas de bytes. Esto es debido a que cada parámetro que se envía tiene dos bytes asociados. El orden de envío de parámetros es siempre el mismo, independientemente de si es de transmisión o de recepción. El orden se especifica en la Figura 3.15, en el que primero está siempre la primera coordenada generalizada y al terminar estas, se incluye el tiempo. Si se quieren enviar dos puntos, el primero será el punto intermedio y el segundo el final, y para ello se envían contiguos con el mismo formato que el especificado previamente.

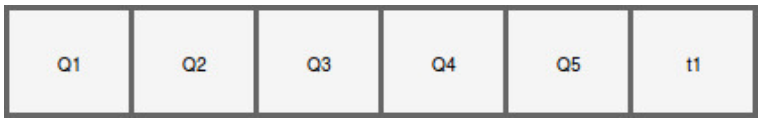

Figura 3.15: Orden de parámetros en el protocolo.

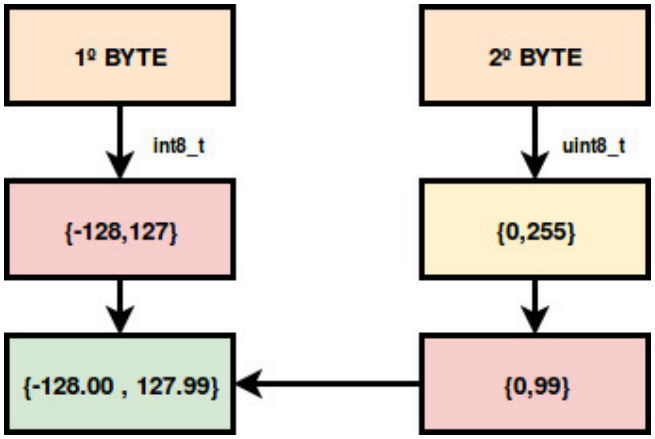

Figura 3.16: Conversión de las coordenadas generalizadas.

Cada parámetro tiene dos bytes asociados. En el caso del tiempo, se pueden mandar valores en el rango $\{0, 65535\}$ y el primer byte enviado es el correspondiente a los 8 bits más bajos del número representado. En el caso de las posiciones generalizadas se tiene que enviar un número real, con su parte entera y su parte decimal. El primer byte es el encargado de la parte entera y el segundo byte de la parte decimal. El primer byte posee signo, por lo que se pueden mandar posiciones entre $\{-128, 127\}$. El segundo byte puede darnos valores entre $\{0, 255\}$, pero este valor se limita entre $\{0, 99\}$ obteniendo una precisión de dos decimales en las posiciones. Esta precisión de dos decimales es suficiente para el manejo de las posiciones debido a que la máxima resolución de los motores es de 1024 pulsos. En la Figura 3.16 se muestra la conversión de las coordenadas cartesianas.

En la Figura 3.17 se muestra la máquina de estados presente en la tarea *vComTask*. En esta tarea se sigue el protocolo de comunicación implementado. Los dos primeros estados son para los bytes de señalización, el tercero para el identificador, el cuarto modifica la variable length y el quinto guarda el comando recibido. Durante el estado de WAITLENGTH la máquina se dedica a coger las duplas de los parámetros guardados en las colas y los manipula siguiendo el orden y las conversiones antes descritas. Una vez acabado y comprobado el checksum, la máquina discrimina un nuevo camino según el comando recibido. Si el comando es READ, se salta al estado GET-POS, en el cual se procede a obtener una lectura de las coordenadas generalizadas en las que está el robot y se empaquetan para su envió por la UART6. Si el comando es WRITE, se lanza una trayectoria. Esta es simple si solo se han recibido cinco coordenadas generalizadas y un valor de tiempo o es doble si se han recibido 10 coordenadas generalizadas y dos valores de tiempo. Una vez terminado, se vuelve al comienzo de la máquina de estados para esperar otra orden distinta.

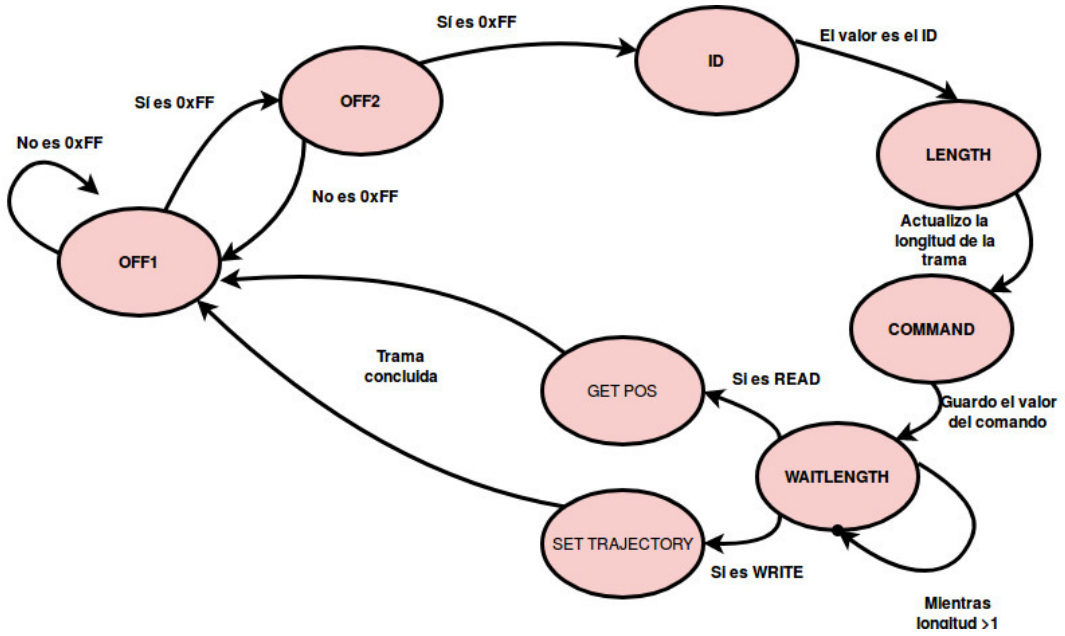

Figura 3.17: Máquina de estados de la tarea *vComTask*.

Capítulo 4

Conclusiones y líneas futuras

4.1. Conclusiones

En cuanto al desarrollo software de bajo nivel para la implementación de la comunicación con los servo motores, se encontraron muchas dificultades. Se terminó por decidir que el uso de la HAL que proporciona el fabricante no era el mejor medio para el desarrollo del Trabajo Fin de Grado. En cambio, se decidió incidir más en el Datasheet del fabricante para desarrollar el driver de control de bajo nivel utilizando únicamente lo necesario. Además, esta tarea facilitó la implementación del sistema operativo FreeRTOS, puesto que de otra forma no era posible incluirlo en el código hasta el punto deseado. El desarrollo del software adaptado a un sistema operativo en tiempo real, facilitó en gran medida la implementación de la exclusión mutua.

En relación al control de los motores, se encontraron grandes lagunas en la información procedente del fabricante. Además, el control de los motores se encontró errático y en muchas ocasiones de forma aleatoria. Se depuraron todos los errores posibles y se siguió, en gran medida, el concepto de comunicación que utilizaba el fabricante en el desarrollo de sus robots. Aun así, los motores se sobrecalentaban demasiado y como medida de emergencia impuesta se apagaban, lo cual bloqueaba el bus de comunicación e impedía el desarrollo total de las pruebas. En líneas futuras la mejor idea sería la búsqueda de nuevos motores.

Respecto al control del robot, se estudiaron las ecuaciones cinemáticas directas e inversas y el cálculo de trayectorias, presentando un nuevo campo de estudio al alumno. Éste se encontró francamente interesante y presentó un pequeño desafió para su implementación.

En cuanto al protocolo de comunicación se consiguió con éxito el desarrollo de una plataforma para el control del robot, de fácil tratamiento, con futuras posibles mejoras.

4.2. Líneas futuras

Este Trabajo de Fin de Grado tenía como objetivo migrar el control del brazo robótico PhantomX a la tarjeta de desarrollo NUCLEO-446RE que tras ser implementada del todo presenta las siguientes propuestas futuras:

- Se plantea el cambio de los motores AX-12A por otros nuevos motores que permitan un mejor control y ponga menos limitaciones en la comunicación.
- Se propone el desarrollo de un simulador que permita comunicarse con el brazo robótico para diseñar movimientos en un entorno virtual y, posteriormente, el brazo robótico desarrollarlos en el mundo real.
- Incluir el cálculo de las ecuaciones cinemáticas directas e inversas dentro de la tarjeta NUCLEO-446RE. Con esto se conseguiría liberar al simulador de realizar estos cálculos permitiendo poder simular más manipuladores robóticos dentro de la misma simulación.
- Incorporar sensores de presión al efector final para incluir una realimentación háptica en el manipulador. Con esto se podría saber si el brazo robótico está cogiendo el objeto deseado y controlar la fuerza necesaria para ni dañarlo ni soltarlo durante la realización de la trayectoria.
- La inclusión de un nuevo grado de libertad para poder obtener un manipulador de 6 grados. Con esto se conseguiría un manipulador completo con un gran abanico de movimientos.

Apéndice

Apéndice A

Especificaciones del Motor

Dirección	Elemento	Acceso
0	Model Number L	RD
1	Model Number H	RD
2	Version of Firmware	RD
3	ID	RD,WR
4	Baud Rate	RD,WR
5	Return Delay Time	RD,WR
6	CW Angle Limit L	RD,WR
7	CW Angle Limit H	RD,WR
8	CCW Angle Limit L	RD,WR
9	CCW Angle Limit H	RD,WR
11	Highest Limit Tempeture	RD,WR
12	Lowest Limit Voltage	RD,WR
13	Highest Limit Voltage	RD,WR
14	Max Torque L	RD,WR
15	Max Torque H	RD,WR
16	Status Return Level	RD,WR
17	Alarm Led	RD,WR
18	Alarm Shutdown	RD,WR
20	Down Calibration L	RD
21	Down Calibration	RD
22	Up Calibration L	RD
23	Up Calibration H	RD
24	Torque Enable	RD,WR
25	LED	RD,WR
26	CW Compliance Margin	RD,WR
27	CCW Compliance Margin	RD,WR
28	CW Compliance Slope	RD,WR
29	CCW Compliance Slope	RD,WR
30	Goal Position L	RD,WR
31	Goal Position H	RD,WR
32	Moving Speed L	RD,WR
33	Moving Speed H	RD,WR
34	Torque Limit L	RD,WR
35	Torque Limit H	RD,WR
36	Present Position L	RD
37	Present Position H	RD
38	Present Speed L	RD
39	Present Speed H	RD
40	Present Load L	RD
41	Present Load H	RD
42	Present Voltage	RD
43	Present Tempeture	RD
44	Registered Instruction	RD,WR
46	Moving	RD
47	Lock	RD,WR
48	Punch L	RD,WR
49	Punch H	RD,WR

Tabla A.1: Tabla de Registros del servomotor AX-12A (Dynamixel, 2006)

Bibliografía

Dynamixel (2006). *User's Manual dynamixel AX-12A*. Robotis, `http://www.trossenrobotics.com/dynamixel-ax-12-robot-actuator.aspx`. [Online: acessed 05/01/18].

MW Spong, S. Hutchinson, MVidyasagar (2005). *Robot Modeling and Control*. JOHN WILEY and SONS, INC.

Robotis (2018). Dynamixel. `http://www.robotis.us/dynamixel/`. Online, accessed 4/01/18.

ST Microelectronics (2017a). *RM0390 Reference Manual*. ST Microelectronics, `www.st.com/resource/en/reference_manual/dm00135183.pdf`. [Online: acessed 05/01/18].

ST Microelectronics (2017b). *UM1795 User Manual*. ST Microelectronics, `www.st.com/resource/en/user_manual/dm00105879.pdf`. [Online: acessed 05/01/18].

Wikipedia (2018). R.U.R. `https://es.wikipedia.org/wiki/R.U.R._(Robots_Universales_Rossum)`. [Online; accessed DATE].

Álvaro Gutiérrez Martín, Félix Monasterio-Huelin (2017a). *Kinematics*. Etsit, `robolabo.etsit.upm.es/asignaturas/rob/apuntes/kinematics.pdf`. [Online: acessed 05/01/18].

Álvaro Gutiérrez Martín, Félix Monasterio-Huelin (2017b). *Trajectory*. Etsit, `robolabo.etsit.upm.es/asignaturas/rob/apuntes/trajectory.pdf`. [Online: acessed 05/01/18].