Trabajo Fin de Máster
Máster en Ingeniería Industrial

Diseño, Impresión 3D, Modelado y Control de un Brazo Robótico de 3 Grados de Libertad

Autor: Isidro Marcelo Jáñez Vaz

Tutor: Ignacio Alvarado Aldea

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Máster
Máster en Ingeniería Industrial

Diseño, Impresión 3D, Modelado y Control de un Brazo Robótico de 3 Grados de Libertad

Autor:
Isidro Marcelo Jáñez Vaz

Tutor:
Ignacio Alvarado Aldea
Profesor Titular

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Máster: Diseño, Impresión 3D, Modelado y Control de un Brazo Robótico de 3 Grados de Libertad

Autor: Isidro Marcelo Jáñez Vaz

Tutor: Ignacio Alvarado Aldea

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A mi familia
A mis amigos
A mis profesores

Agradecimientos

En primer lugar, dar las gracias a mi familia. A mis padres Isidro y Natalia, y a mis hermanos Magdalena y Ángel, por ponérmelo todo tan fácil en casa.

Gracias a Ignacio Alvarado Aldea, por haber depositado en mi su confianza cuando le propuse la realización de este trabajo en unos plazos y fechas exigentes, y por el asesoramiento prestado durante la realización del mismo.

Gracias a los profesores de esta Escuela y a los profesores del Colegio Claret, por la labor que desempeñan.

Gracias a mis amigos, por seguir ahí.

Isidro Marcelo Jáñez Vaz

Sevilla, 2021

Resumen

El objetivo principal de este proyecto es poner en funcionamiento un robot de tres grados de libertad que permita ser programado como si de un robot insutrial se tratara, siendo capaz de ejecutar movimientos tanto en coordenadas articulares como cartesianas coordinando a todos sus motores durante el trayecto.

En primer lugar, se realizará en *Catia* un diseño del conjunto de piezas de plástico necesarias para construir la estructura del brazo articulado. Este diseño deberá garantizar unas prestaciones mínimas en cuanto a holguras, para permitir posteriormente un control satisfactorio de los movimientos del sistema. El robot contará, además, con un efector final con la función de pinza, que permitirá la manipulación de objetos de pequeño tamaño. A este diseño lo deberá acompañar un modelado de la cinemática del sistema, que haga posible su posterior control en coordenadas articulares y cartesianas.

Para la programación del control del sistema se trabajará con el IDE de *Arduino*, compatible con el microcontrolador del que se hará uso. El objetivo final será la programación de una librería capaz de ejecutar el control de bajo nivel de los motores paso a paso y permitir a su vez, a un hipotético usuario, programar tareas a ejecutar por el robot mediante códigos de sintaxis genérica.

Como parte final del trabajo, se pondrá a prueba el funcionamiento del sistema diseñado y construido, preparando la realización de tareas en un espacio de trabajo habilitado para este fin, y se comprobará la adecuación de las prestaciones del control logrado.

Abstract

The main objective of this project is to put into operation a robot with three degrees of freedom that can be programmed as if it were an insutrial robot, capable of executing movements in both joint and Cartesian coordinates, coordinating all its motors along the way.

Firstly, a design will be made in *Catia* of the set of plastic parts needed to build the structure of the articulated arm. This design will have to guarantee minimum performance in terms of clearances, in order to subsequently allow satisfactory control of the system's movements. The robot will also have an end effector with a gripper function, which will allow the manipulation of small objects. This design must be accompanied by a modelling of the kinematics of the system to enable its later control in joint and cartesian coordinates.

For the programming of the control of the system, the *Arduino IDE* will be used, which is compatible with the microcontroller that will be employed. The final objective will be the programming of a library capable of executing the low-level control of the stepper motors and at the same time allowing a hypothetical user to program tasks to be executed by the robot using generic syntax codes.

As a final part of the work, the operation of the system designed and built will be tested, preparing the execution of tasks in a workspace set up for this purpose, and the suitability of the control performance achieved will be verified.

Índice Abreviado

Índice

Índice de Figuras

Índice de Tablas

Índice de Códigos

1 Introducción

En este primer capítulo se presentarán las ideas que motivan la realización de este proyecto, se expondrán los propósitos con los que se decide abordarlo, y se realizará una introducción al contenido de este documento.

1.1 Motivación

La utilización de los robots industriales en la industria actual está ya ampliamente afianzada, presentando un constante crecimiento y expandiendo sus campos de aplicación año a año. La variada combinación de disciplinas que intervienen en el funcionamiento de estas máquinas hace que despierten un importante interés en cualquier estudiante de ingeniería que pretenda adquirir conocimientos transversales pero enfocados al ámbito de la automatización.

Tras el paso por unos estudios de grado y máster con una gran carga teórica, en el alumno siempre se encuentra latente un fuerte deseo de comprobar la aplicabilidad de todos esos conocimientos adquiridos y demostrar, tanto a sí mismo como a su entorno, que lo aprendido se puede llevar a la práctica con éxito. Con este proyecto se buscaba, por tanto, satisfacer esa fuerte voluntad de hacer ingeniería, y de desarrollar una idea desde su concepción sobre el papel hasta su construcción y puesta en funcionamiento.

Existe una cantidad ingente de vídeos en Internet en los que se muestran brazos articulados de características similares a las del desarrollado en este proyecto, tanto en estructura y materiales utilizados como en funcionamiento, de los que se logra hacer un control de mayores o menores prestaciones en cada caso. De esta forma, no cabe duda de que el presente trabajo resulta un proyecto atractivo y que permite poner mucho de lo aprendido en práctica.

Por último, la impresión 3D resulta ser una tecnología en constante crecimiento y con innumerables aplicaciones, con una gran importancia en el ámbito del prototipado y utilizada cada vez por más empresas, haciendo muy interesante adquirir experiencia en el trabajo con esta disciplina tan eminentemente práctica.

1.2 Objetivos

Los principales objetivos que se desean alcanzar con este trabajo son los presentados a continuación:

- Diseñar una estructura para el brazo articulado lo más estilizada posible, que permita lograr la movilidad deseada y que sea apta para su impresión en 3D. Se tratará de un robot con tres grados de libertad que presente un efector final que permita hacer las veces de una pinza para la manipulación de objetos de pequeño tamaño. Deberá ser un diseño lo suficientemente estricto en cuanto la existencia de holguras como para que el control del robot pueda tener unas prestaciones satisfactorias.

- Llevar a cabo la impresión en 3D de las piezas diseñadas, seguida de la construcción y montaje del robot, haciendo uso de motores paso a paso bipolares para el accionamiento de las articulaciones.
- Modelar mediante ecuaciones parametrizadas la cadena cinemática del brazo articulado.
- Programar una librería para el control del robot que sea capaz de llevar a cabo el control de bajo nivel de los motores paso a paso, y permita a su vez programar tareas para el robot con códigos con una sintaxis similar a la utilizada en lenguajes como *RAPID*.

1.3 Estructura

La estructuración del contenido de este documento lo divide en los siguientes capítulos posteriores a este primero:

- **Capítulo 2:** en el segundo capítulo se describe el procedimiento seguido para obtener la geometría de los sólidos que componen el brazo articulado, tanto los cuerpos asociados a los propios eslabones como los elementos auxiliares necesarios para lograr la estructura completa con la movilidad deseada.
- **Capítulo 3:** en el tercer capítulo se detalla el hardware completo necesario para la puesta en funcionamiento del robot desarrollado, tanto desde el punto de vista mecánico como del control.
- **Capítulo 4:** en el cuarto capítulo se desarrolla la obtención de los modelos cinemáticos directo e inverso del robot, se definen los límites de su espacio de trabajo, y se estudian los puntos singulares.
- **Capítulo 5:** en el quinto capítulo se explica el funcionamiento del código que hace posible desde el control de bajo nivel de los motores paso a paso hasta la ejecución de movimientos coordinando a los tres motores durante el trayecto.
- **Capítulo 6:** en el sexto capítulo se trata la capa superior de la jerarquía de control del robot, describiendo el funcionamiento de la máquina de estados que permite utilizarlo como un robot industrial convencional.
- **Capítulo 7:** el séptimo capítulo presenta el diseño y la fabricación del *Escudo* que permite controlar el robot de forma ordenada y compacta, haciendo uso de botones y joysticks para gestionar las conmutaciones entre los modos de funcionamiento.
- **Capítulo 8:** el octavo y último capítulo proporciona las conclusiones en base a la visión global del trabajo realizado, y propone posibles ampliaciones futuras de este proyecto.

2 Construcción del modelo

En este capítulo se describirá el procedimiento seguido para obtener la geometría de los sólidos que componen al brazo articulado, tanto los cuerpos asociados a los propios eslabones como los elementos auxiliares necesarios para lograr la estructura completa con la movilidad deseada.

2.1 Requisitos de diseño

En el presente proyecto se ha llevado a cabo un diseño libre de un brazo articulado formado por tres eslabones móviles más un eslabón fijo, con las respectivas articulaciones entre ellos, y que presenta una pinza en como efector final. Para este fin se han tomado como punto de partida únicamente los innumerables ejemplos de robots de características similares que existen en la bibliografía, así como los bocetos realizados por el propio alumno recopilando ideas tanto extraídas de dichos robots como propias.
El proceso de diseño del brazo presentaba, por tanto, una cantidad indefinida de grados de libertad y variables de decisión a establecer, para lo que se ha seguido una serie de criterios comunes que se exponen a continuación:

- **Reducido peso y uso de material en los eslabones:** en cualquier proyecto que involucre a la impresión 3D prima la reducción del uso del material, no solo por cuestiones económicas sino, en mayor medida, por la minimización del tiempo de impresión requerido para la obtención de cada pieza.
  Por esta razón, en este proyecto se han diseñado eslabones huecos de pared delgada, rondando éstas los 2-3 mm de espesor en todos los casos; y se ha evitado la presencia de elementos en voladizo en la medida de lo posible, con el fin de eliminar la necesidad de *soportes* en el proceso de impresión. Para lograr esto último, ha sido frecuente la utilización de ángulos de 45° en la silueta de los cuerpos según la dirección ascendente del proceso de deposición de material, y se han dejado abiertas las caras superiores de los cuerpos de los eslabones, preparándolos para posteriormente disponer tapaderas.

- **Esbeltez y rigidez:** la minimización del uso del material, así como la no utilización de secciones macizas, llevan consigo de forma inevitable la obtención de piezas flexibles o incluso frágiles. Con el fin de evitar estos inconvenientes sin renunciar a la esbeltez del diseño, se han dispuesto refuerzos internos tanto longitudinales como transversales en los diseños de todos los eslabones del robot, que permitan dar a la estructura del mismo las prestaciones de rigidez y resistencia requeridas. La presencia de estos refuerzos ha sido especialmente necesaria en el anclaje de los motores a los cuerpos de plástico, así como en la construcción de las bisagras que hacen posibles las articulaciones del brazo.

- **Minimización de las inercias usando reductoras:** pese a que en un principio se planteó la posibilidad de diseñar un robot de accionamiento directo, finalmente se adoptó la utilización de reductoras para la transmisión del movimiento de los motores a los eslabones, fundamentalmente por el hecho de que el diseño sin reductoras de un robot de reducidas dimensiones resultaba poco práctico: el poner los motores de la segunda y tercera articulación en el eje de la propia articulación para el accionamiento directo implicaría que tanto el motor como los rodamientos y la varilla que construyen la articulación deben estar todos en una misma línea recta en el espacio. Para que la articulación funcione correctamente, necesita de al menos dos rodamientos por eslabón, que junto al ancho del motor y a las partes de

plástico que harían falta para sujetar cada elemento, hacen que el ancho total de la articulación debiera ser demasiado grande y complejo, resultando articulaciones demasiado aparatosas en comparación con las dimensiones del robot.
Sin embargo, utilizando reductoras para la transmisión del movimiento, el motor pasa a estar en un eje paralelo al de la articulación, en lugar de en su mismo eje, de forma que la articulación puede ser mucho más estrecha y, además, sencilla de diseñar.
A las ventajas anteriores se les une el hecho de que, si los motores pasan a estar a mitad del cuerpo de cada eslabón en lugar de en el extremo, las inercias que provocan se reducen considerablemente, otorgando mayor agilidad al robot y permitiendo posteriormente alcanzar mayores velocidades en el control.
El uso de reductoras también aumenta la resolución del control de los movimientos, al ser menor el desplazamiento que experimentan los eslabones por cada paso de giro de los motores.
Por último, el uso de reductoras queda también abalado por las implicaciones mecánicas que éstas involucran: su utilización permitirá demandar menor par a los motores paso a paso, y por tanto permitirá hacer uso de motores de menor tamaño o, visto de otro modo, aumentará el margen de seguridad con el que los motores que se escojan sean capaces de accionar al robot.

- **Ausencia de fricción y de holguras en las articulaciones:** para lograr un comportamiento aceptable y un desempeño decente de los movimientos por parte del robot, es necesario garantizar la ausencia de holguras en las articulaciones.
  Con este objetivo, la conexión de cada eslabón con el siguiente en el presente diseño se realiza haciendo uso de rodamientos y varillas de acero que, además de permitir disminuir el juego de las articulaciones, reducen la fricción entre eslabones que deberían vencer los motores.

- **Amplia movilidad:** en el diseño de los eslabones, y en especial de las articulaciones, se ha tenido muy presente la importancia de permitir un movimiento relativo de suficiente amplitud de cada eslabón respecto al anterior antes de que se produzca la intersección entre los cuerpos de ambos.

- **Facilidad de montaje y uso de tornillería disponible:** para la construcción de la estructura del brazo articulado se previó el uso de una única métrica de tornillería, que facilitase tanto el proceso de montaje como la adquisición de los tornillos y tuercas en el mercado. Se escogió la métrica M3 por ser la métrica de menor tamaño que se puede encontrar de manera habitual en ferreterías y establecimientos dedicados al bricolaje.
  Por tanto, los diseños de las piezas presentan numerosos orificios destinados a permitir la introducción de tornillos destinados a: el anclaje de los motores, la sujección de los finales de carrera, y la fijación de todas las partes auxiliares que se unen al cuerpo de los eslabones, entre ellas las ruedas dentadas que posibilitan la transmisión del movimiento de los motores a la articulación.

- **Diseño ajustable y tolerante:** dado que se trata de un robot construido en plástico impreso en 3D haciendo uso de una impresora de prestaciones humildes, se debe tener en cuenta que las dimensiones de las piezas nunca se ajustarán perfectamente a las dimensiones del diseño. Por otra parte, existen parámetros como el recorrido de los Finales de Carrera que no pueden asumirse como conocidos durante la fase de diseño del brazo. Es por esto que los anclajes de dichos Finales de Carrera al cuerpo de los eslabones se han ideado de forma que pueda ajustarse la penetración del Final de Carrera al eslabón correspondiente, haciendo uso de unos carriles embebidos en las propias piezas.

- **Adecuada tensión de las correas:** por otra parte, también es fundamental lograr la adecuada tensión en las correas que transmiten el movimiento. Para esto se han utilizado correas de longitud estandarizada, de manera que siempre puedan encontrarse en el mercado en caso de necesidad de ser reemplazadas; y se han realizado numerosas pruebas, mediante la impresión de prototipos, hasta encontrar los valores adecuados para la separación entre el eje de giro de cada motor y el eje de giro de la articulación a la que provoca movimiento. Además, para garantizar que el estado de tensión de la correa una vez realizado el montaje es adecuado, se ha ideado un sistema de anclaje de las ruedas dentadas al cuerpo de los eslabones que logra aumentar dicha tensión de forma progresiva, y que será mostrado más

adelante en este capítulo.

- **Cableado oculto:** por último, el proceso de diseño del robot ha buscado que el cableado del mismo pueda recorrer los eslabones quedando oculto al exterior, siendo esto no solo conveniente por cuestiones estéticas, sino también para evitar los posibles enredos o estorbos que conllevaría la presencia de los cables por el Espacio de Trabajo del brazo.

2.2 Diseño de los sólidos en Catia

Para llevar a cabo el diseño de todas las piezas de plástico del robot se ha hecho uso del software *Catia V5*, teniendo en cuenta las consideraciones expuestas en el apartado anterior. En la Figura 2.1 se representan las partes principales involucradas en la construcción del mismo. A continuación, se describen las características de los sólidos diseñados y de la estructura obtenida.

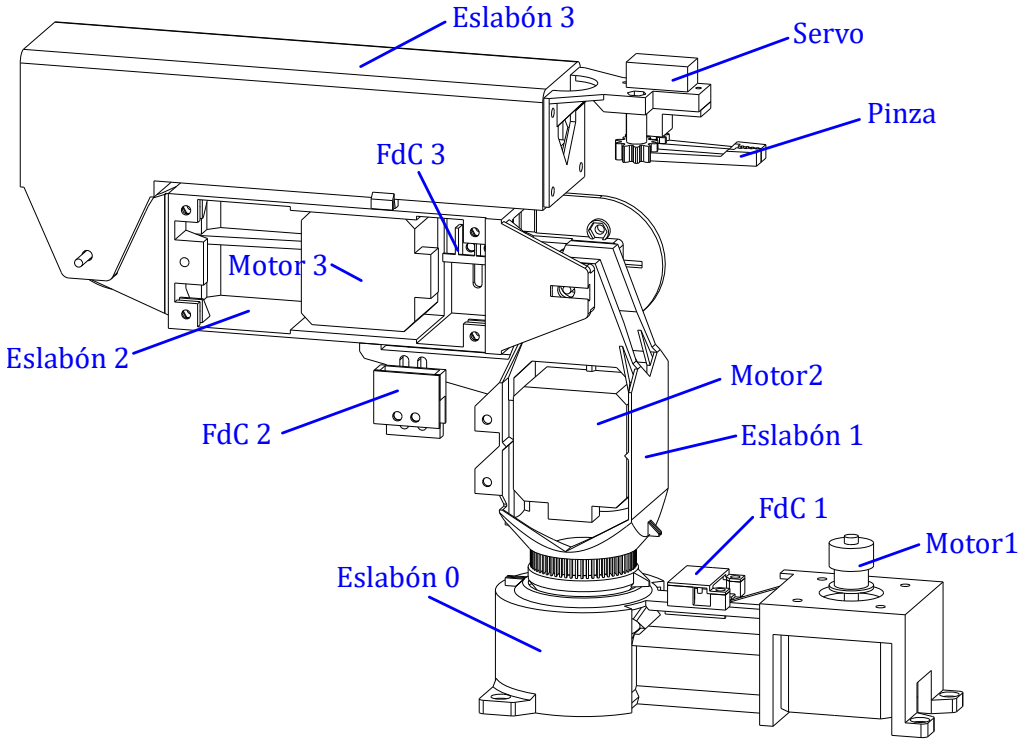

Figura 2.1 Partes del brazo articulado.

2.2.1 Cuerpos de los eslabones y articulaciones

El robot diseñado consta de tres eslabones móviles más en eslabón fijo que va anclado a la mesa de trabajo. Se trata de un brazo articulado de configuración TRR, ya que las tres articulaciones que presenta son de Torsión, Rotación y Rotación. Los cuerpos de los eslabones son las piezas de mayor complejidad, y atienden cada una a funciones diferentes, por lo que requieren ser descritas por separado:

Eslabón 0

El Eslabón 0, mostrado en la Figura 2.2, es el primero de los eslabones del robot. Como se ha indicado anteriormente, estará anclado a la mesa de trabajo, motivo por el cual debe presentar unos salientes con los orificios correspondientes para introducir los tornillos que permitirán dicha fijación a la madera que hace de base del sistema, requiriendo estos salientes del espesor adecuado para ser capaces de evitar el vuelco del robot sin que se produzca su rotura.

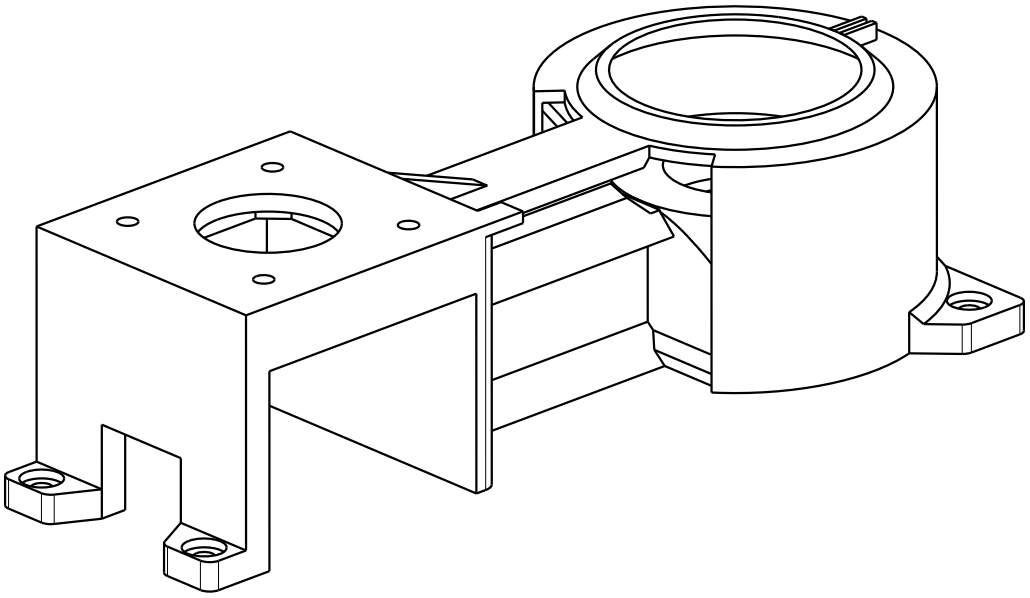

Figura 2.2 Eslabón 0.

A lo largo de la dirección longitudinal de este eslabón se encontrará dispuesta la correa que permite al Motor 1 accionar al Eslabón 1, lo que hace que este Eslabón 0 deba soportar la torsión que provoca la tensión de dicha correa. Es por esto que en su parte central se encuentra una porción de plástico que imita a una viga en doble T, que reducirá la tendencia del plástico a plegarse sobre hacia la correa.

El Eslabón 0 debe presentar también, en su parte central superior, los salientes que permiten el posterior anclaje y ajuste del Final de Carrera 1, como se mostraba en la Figura 2.1.

Por último, la parte final del Eslabón 0, que es la más compleja, está asociada a alojar los rodamientos de la primera articulación y permitir adicionalmente la salida de los cables del robot sin limitar el movimiento del mecanismo. Sobre dichos rodamientos recaerá el peso del resto de eslabones y motores del robot, por lo que esta parte del Eslabón 0 requiere de una gran robustez.

Articulación 1

En la Figura 2.3 se muestra el detalle de la unión del Eslabón 1 (pieza superior) al Eslabón 0 (pieza central) en la Articulación 1. Como se ha mencionado previamente, sobre esta articulación descansa el peso del robot completo, por lo que requiere de una robustez especial. Es por eso que, a diferencia de como ocurrirá en las articulaciones posteriores, en esta articulación se han utilizado dos rodamientos 608ZZ; y en lugar de hacerse uso de varillas, será el propio plástico del Eslabón 1 el que pase por el interior de estos rodamientos de gran diámetro, mientras que el Eslabón 0 rodeará su periferia.

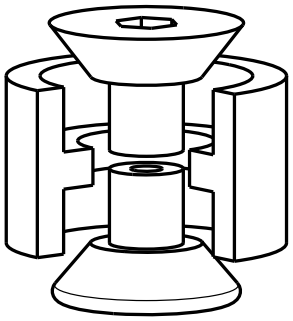

Figura 2.3 Detalle de la Articulación 1.

La unión entre los dos eslabones debe ser tal que soporte fuerzas en todas direcciones, por lo que no

solo debe soportar fuerzas verticales de sentido descendente sino también de sentido ascendente, para evitar que el Eslabón 1 se pueda elevar de forma inintencionada. Es aquí donde interviene la tercera de las piezas mostradas en la figura (pieza inferior), que hará de "tapón" para el Eslabón 1 de forma que a través de estas piezas se introducirá un tornillo que comprimirá al Eslabón 1 y al tapón contra los rodamientos y garantizará su fijación al Eslabón 0.

Eslabón 1

El Eslabón 1, mostrado en la Figura 2.4, es el primero de los eslabones móviles del mecanismo.

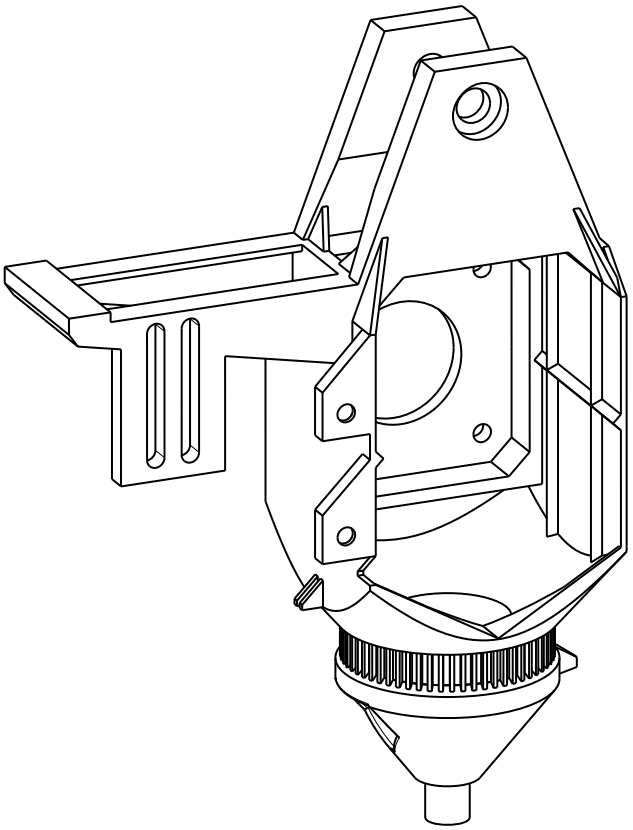

Figura 2.4 Eslabón 1.

En primer lugar, en su parte inferior debe presentar un orificio de suficiente tamaño para permitir la salida de forma holgada de todo el cableado que llega proveniente de los motores y finales de carrera del robot.

Posteriormente, se debe incluir un saliente que forme parte de este eslabón y que permitirá accionar al Final de Carrera 1 que se encontraba anclado al Eslabón 0.

A continuación se encuentra la rueda dentada de la primera reductora. Para todas las reductoras del diseño se ha utilizado una relación de reducción de 1:3, de manera que, dado que las poleas dentadas usadas en los motores presentan 20 dientes, las ruedas dentadas de las piezas de plástico presentarán 60 dientes. Por supuesto, para encontrar las dimensiones de diámetro adecuadas para dichas ruedas así como del tamaño correcto de los dientes fueron necesarios numerosos prototipos previos hasta encontrar los valores utilizados en las piezas finales.

En la parte central del Eslabón 1 se encuentra la cavidad que alojará al Motor 2, convenientemente dotada con refuerzos que permiten alcanzar una adecuada rigidez sin tener que renunciar al uso de paredes delgadas en las piezas. En esta fracción del eslabón se observan también los salientes que permitirán el anclaje de la Tapadera del eslabón a éste.

Sobre el Eslabón 1 descansará el Eslabón 2 cuando el robot se encuentre en su posición de reposo, lo que hace necesario que disponga de un voladizo destinado a soportar el peso del resto del robot. Este voladizo se aprovecha, además, para disponer los carriles que permitirán el anclaje y ajuste del Final de Carrera 2.

Por último, en la parte superior del eslabón se encuentran los huecos que alojarán a los rodamientos de la Articulación 2, estando toda esta zona convenientemente reforzada para paliar el hecho de que el eslabón, de sección hueca, pasa de tener cuatro paredes a tener solo dos caras enfrentadas en la zona de la articulación.

Articulación 2

En la Figura 2.5 se muestra un detalle de la segunda articulación del mecanismo, donde se une el Eslabón 2 al Eslabón 1 haciendo uso de cuatro rodamientos 623ZZ y una varilla de acero de 3 mm de diámetro. Esta estrategia será también la utilizado en la tercera articulación.

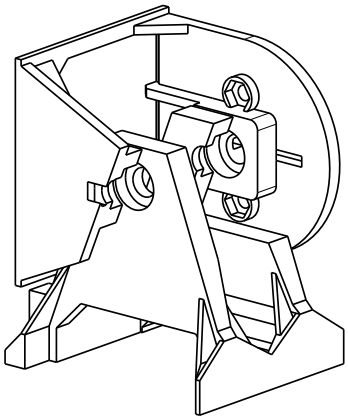

Figura 2.5 Detalle de la Articulación 2.

En ambas articulaciones ha sido fundamental el refuerzo de la unión de estas al cuerpo de los eslabones para combatir la flexibilidad del plástico, así como prestar especial atención a la forma que se daba a las piezas para evitar que se pudiera producir la interferencia entre éstas durante el movimiento del robot dentro de los rangos deseados.

Rueda dentada de las articulaciones 2 y 3

En la Figura 2.6 se muestra un detalle del anclaje de las ruedas dentadas de las reductoras 2 y 3 al cuerpo de los eslabones.

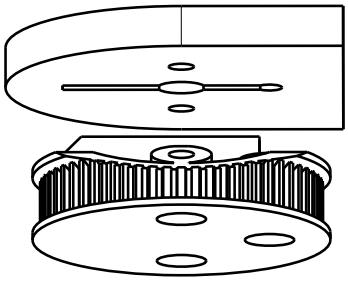

Figura 2.6 Detalle de la rueda dentada.

Como se han mencionado en las Sección 2.1, es necesario garantizar que el estado de tensión de la correa una vez realizado el montaje sea adecuado. Para ello, la parte de la pieza de las ruedas dentadas que penetra en el cuerpo de los eslabones cuenta con una primera parte en rampa (lado izquierdo de la Figura 2.6) que permite el incremento progresivo de la tensión de la correa conforme se conecta la rueda dentada al eslabón en el proceso de montaje del robot. Posteriormente, se realiza la fijación de ambas partes mediante el uso de tres tornillos, evitando que se pueda producir la relajación de la correa durante los movimientos del robot.

Eslabón 2

En la Figura 2.7 se muestra el diseño del Eslabón 2 del brazo articulado.

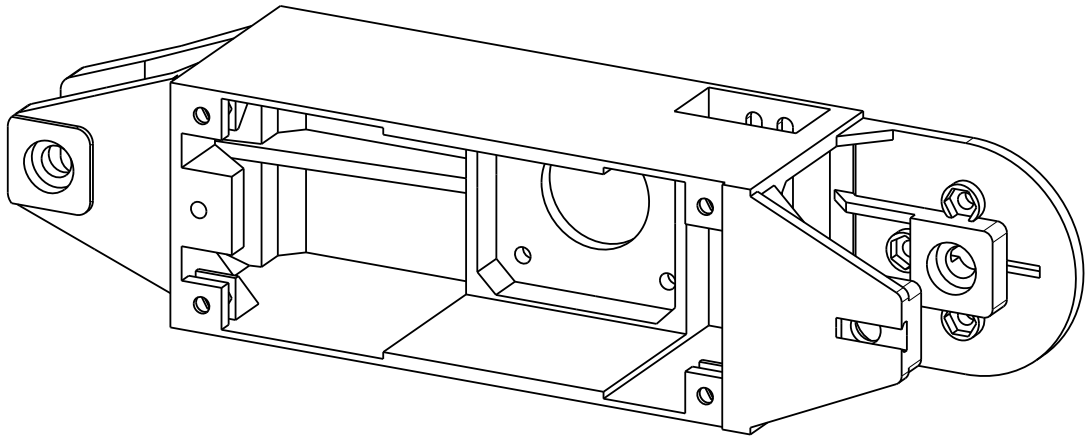

Figura 2.7 Eslabón 2.

Este diseño mantiene la línea de las ideas que se vienen exponiendo para las piezas anteriores. En primer lugar, en la parte derecha, se encuentra la región asociada a alojar los rodamientos que intervienen en la Articulación 2, así como al anclaje de la rueda dentada de esta articulación al eslabón.

Posteriormente, parcialmente ocultos en la figura, están presentes los carriles que hacen posible el ajuste de la posición del Final de Carrera 3, que en esta ocasión quedará inmerso dentro del habitáculo del cuerpo del eslabón, siendo necesario disponer de un orificio en la cara superior para que éste sobresalga.

En la parte central del eslabón, por la cara posterior, se encuentran la zona asociada al anclaje del Motor 3 seguida de una viga de refuerzo que permitirá combatir la torsión provocada por la tensión de la tercera correa. En la misma zona pero en la cara anterior se encuentran los orificios asociados al montaje de la Tapadera de este eslabón, así como un orificio adicional que permitirá la introducción de una varilla para contrarrestar, haciendo uso de un elástico, la torsión ya mencionada de este eslabón en caso de que fuera demasiado elevada.

Por último, en la parte izquierda de la figura se encuentra la región del eslabón que interviene en el mecanismo de la tercera articulación.

Articulación 3

En la Figura 2.5 se muestra un detalle de la tercera articulación del mecanismo, que conecta a los eslabones 2 y 3.

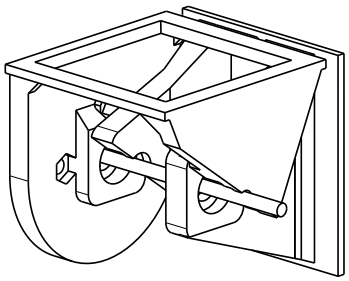

Figura 2.8 Detalle de la Articulación 3.

Esta articulación sigue las mismas ideas ya explicadas para la Articulación 2, haciéndose uso de cuatro rodamientos 603ZZ y una varilla de acero (que en esta ocasión se muestra en la figura) para construir la unión sin permitir holguras y evitando la fricción.

Eslabón 3

En la Figura 2.9 se muestra el tercer y último eslabón del mecanismo del brazo articulado. El diseño de este eslabón atiende a menos exigencias mecánicas que el de los anteriores, al no alojar ningún motor paso a paso,

siendo, por tanto, su peso total también mucho menor. Esto permite el uso de paredes delgadas de espesor constante, aunque reforzadas, a lo largo de todo el eslabón.

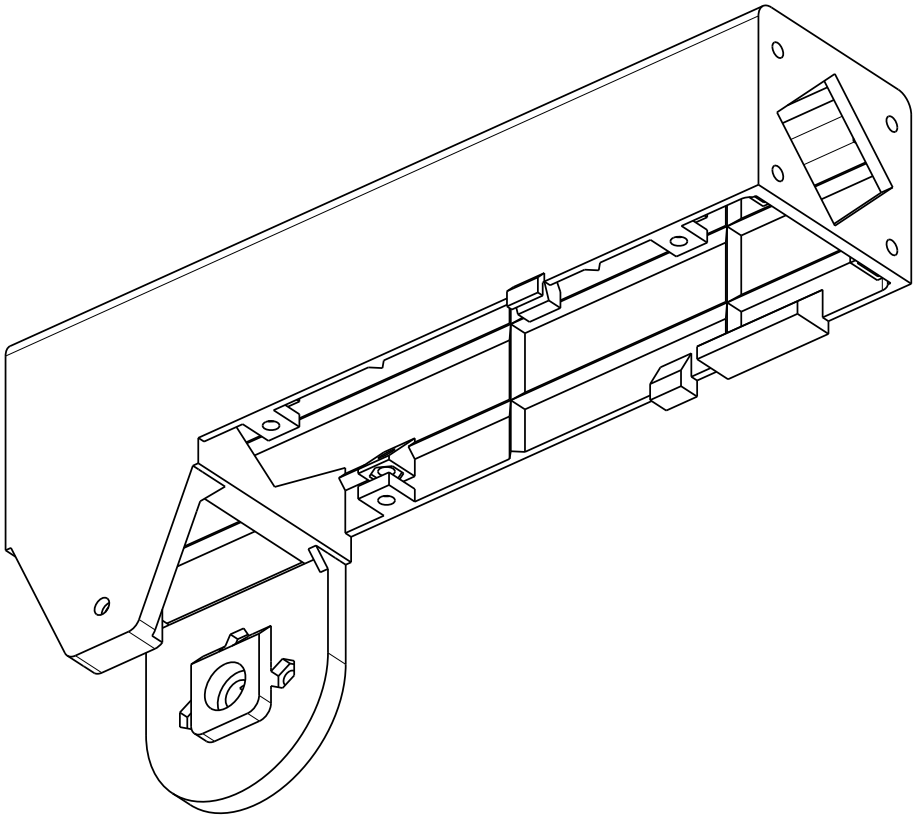

Figura 2.9 Eslabón 3.

En la parte inferior se encuentran los salientes que permitirán el posterior montaje de la Tapadera de este eslabón, así como otros destinados al apoyo del Eslabón 3 sobre el Eslabón 2 cuando el robot se encuentra en su posición de reposo, y a la activación del Final de Carrera 3.

En su extremo final (parte derecha de la figura) el Eslabón 3 cuenta con los orificios que permitirían el anclaje de todo tipo de efectores finales que se diseñen para este robot, así como un hueco central para permitir la entrada y salida del cableado asociado.

2.2.2 Efectores finales

Herramienta puntero

La primera de las herramientas diseñada para este robot, mostrada en la figura 2.10, se trata de un puntero que permitirá comprobar el adecuado control en posición de las coordenadas XYZ que se puede lograr con los tres grados de liberad del mecanismo.

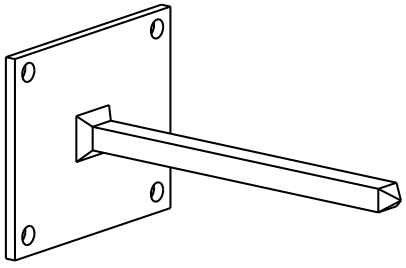

Figura 2.10 Herramienta puntero.

Pinza

Adicionalmente a la Herramienta puntero mostrada, se ha diseñado una pinza como efector final para el robot que, accionada por un Servomotor de 5g, permitirá recoger y soltar objetos de pequeño tamaño, dando así funcionalidades más interesantes al robot.

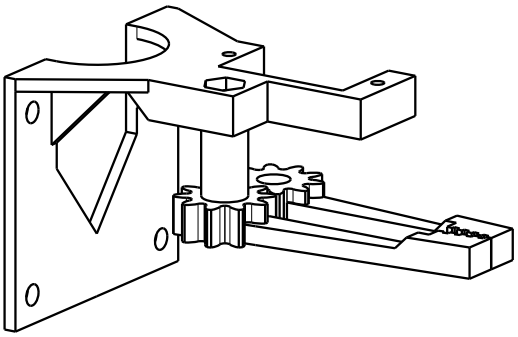

Figura 2.11 Pinzas.

Las partes que componen a esta pinza se muestran en la Figura 2.11, y son tres: el soporte que permite fijar el efector al Eslabón 3 del robot y sujetar al servomotor; la primera mitad de la pinza, que se mueve solidaria con el giro del eje del servomotor; y la segunda mitad de la pinza, que rota respecto al soporte movida por el engranaje que la conecta a la primera de las mitades.

2.2.3 Anclaje de los Finales de Carrera

Como se ha mencionado previamente, en anclaje de los Finales de Carrera al cuerpo de los eslabones requiere de poder ser ajustable durante el montaje del robot. Se han utilizado Finales de Carrera de propósito general, para los que se ha diseñado una carcasa a medida que permite encerrarlos sin holguras y posteriormente sujetarlos, mediante el uso de tornillos, a la altura adecuada de unos carriles dispuestos en el propio cuerpo del eslabón correspondiente. En la Figura 3.3 se muestran las piezas que dan forma a esta idea para el caso concreto del montaje del Final de Carrera 2 en el Eslabón 1.

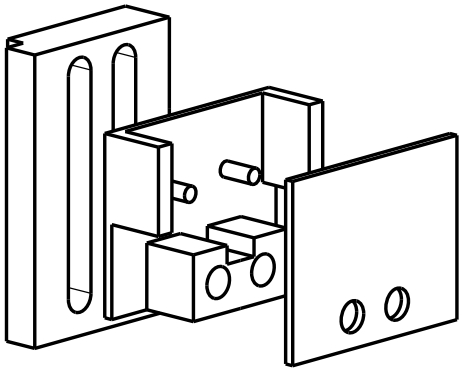

Figura 2.12 Detalle del montaje de los Finales de Carrera.

2.3 Esquema simplificado y dimensiones del robot

En la Figura 2.13 se muestra el esquema simplificado de los eslabones y articulaciones cuyo diseño ha sido descrito anteriormente, así como los parámetros dimensionales que se han definido para caracterizarlo.

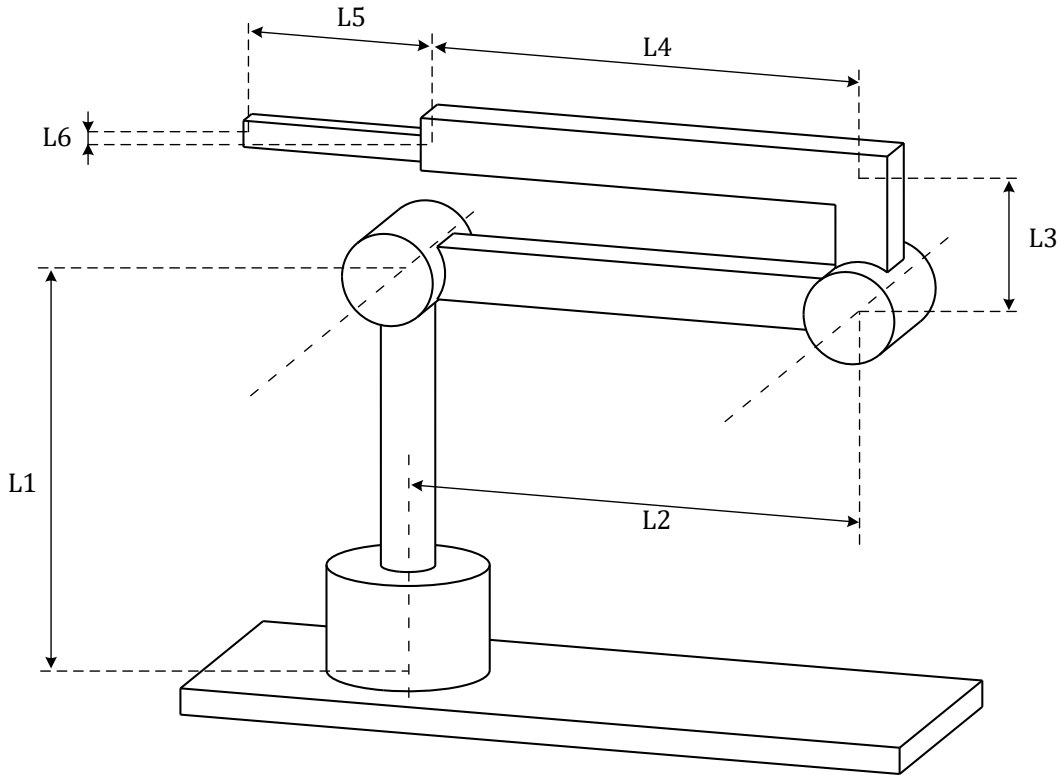

Figura 2.13 Eslabones, articulaciones y dimensiones.

Los valores de estas dimensiones para el diseño concreto de este proyecto se recogen en la Tabla 2.1.

Tabla 2.1 Dimensiones del robot.

L1	L2	L3	L4	L5	L6
154.85mm	178.1mm	48.5mm	170.0mm	70.0mm	1.0mm

2.4 Generación de los mallados tridimensionales

El último paso a realizar con el diseño obtenido en *Catia* consiste en generar de los archivos que contengan la geometría de los objetos tridimensionales para que éstos puedan ser utilizados por los programas de laminación 3D, los cuales proporcionan los ficheros de extensión ".gcode" que utilizan las impresoras 3D. *Catia* trabaja con un formato propio de archivos (de extensión ".CATPart"), por lo que el archivo del diseño realizado no puede ser directamente utilizado. Los archivos deseados son de de extensión ".stl", y consisten en una malla tridimensional de vértices que constituyen un objeto sin color ni textura. Para la exportación a este formato, *Catia* cuenta con una herramienta propia, denominada *"STL Rapid Prototyping"*.

2.4.1 Sumario de piezas y pesos

Las piezas de plástico del robot construido en este proyecto ha sido impresas en material PLA, en una impresora 3D de tecnología FDM (Deposición de Material Fundido) y los consumos de plástico de cada una de ellas con las características con las que han sido impresas en este proyecto concreto, incluyendo los pesos de los soportes que deben ser retirados durante el postprocesado a mano de cada pieza tras finalizar el proceso de impresión, quedan recogidos en la Tabla 2.2.

Tabla 2.2 Pesos de las piezas.

Pieza	**Masa de PLA**
Eslabón 0	109 g
Eslabón 1	92 g
Eslabón 2	106 g
Eslabón 3	105 g
Tapón eslabón 1	2 g
Tapadera eslabón 1	4 g
Tapadera eslabón 2	7 g
Tapadera eslabón 3	8 g
Rueda dentada (x2)	7 g
FdC 1, parte 1	1 g
FdC 1, parte 2	1 g
FdC 2, parte 1	2 g
FdC 2, parte 2	1 g
FdC 3, parte 1	2 g
FdC 3, parte 2	1 g
Herramienta puntero	5 g
Pinzas - soporte	7 g
Pinzas - engranaje 1	2 g
Pinzas - engranaje 2	2 g
Total (aprox.)	**471 g**

3 Hardware del robot

En este capítulo se describiría el hardware total necesario para la puesta en funcionamiento del robot desarrollado en este trabajo, tanto desde el punto de vista mecánico como del control; a excepción del *Escudo* diseñado y los componentes que éste contiene, que se tratarán en el Capítulo 7.

3.1 Estructura completa del brazo

3.1.1 Métrica de tornillería, varillas, rodamientos

A lo largo del Capítulo 2 se han ido mencionando las características de los elementos no impresos en 3D que son necesarios para el montaje del diseño desarrollado. En este apartado, se recogen a modo de sumario, en la Tabla 3.1, las cantidades empleadas de cada uno de ellos.

Tabla 3.1 Tornillería, varillas y rodamientos.

Elemento	Uds
Tornillo M3 10mm	31
Tornillo M3 20mm	6
Tornillo M3 30mm	2
Varilla acero 3mm diam, long $\leqslant$41,15mm	1
Varilla acero 3mm diam, long $\leqslant$55,2mm	1
Varilla acero 3mm diam, long $\leqslant$27mm	1
Rodamiento 608ZZ	2
Rodamiento 623ZZ	8

3.1.2 Motores paso a paso, correas y servo

Motores paso a paso

Para el accionamiento del robot se han utilizado motores paso a paso Nema 17 alimentados a 12 V, de prestaciones similares a los utilizados por las propias impresoras 3D, por ser este tipo de motores habituales en proyectos de CNC en los que no se requieren pares elevados, y por tanto tratarse de motores económicos y para los que existe una amplia documentación disponible.

Concretamente, el motor utilizado en este proyecto es del de la Figura 3.2, caracterizado por tener 40 mm de alto y soportar 1,68 A.

Figura 3.1 Motor Nema 17 utilizado.

Alternativamente, para el accionamiento de la última articulación se ha probado la sustitución del motor correspondiente por otro motor Nema 17 de 0,35 A, debido a que este tercer motor es el que menor par debe ejercer y la sustitución permite reducir el peso en el Eslabón 3 y por consiguiente agilizar el mecanismo. El comportamiento obtenido en este segundo caso, sin embargo, no difiere demasiado del logrado con el motor original dentro de las velocidades habituales con las que trabaja este robot.

En todos los casos, se trata de motores paso a paso bipolares, de 200 pasos/revolución (1,8 grados/paso), con cableado finalizado en conector DuPont de 2,54 mm.

Correas

Las correas empleadas para la transmisión del movimiento en las reductoras se corresponden con correas estandarizadas de goma, de una sola pieza, de 2mm de pitch (correas GT2) y de 6mm de ancho. La longitud que debe tener cada una de las correas que monta el robot diseñado se recoge en la Tabla 3.2.

Tabla 3.2 Longitud de las correas.

Correa reductora 1	280 mm
Correa reductora 2	200 mm
Correa reductora 3	280 mm

Servomotor

Para el accionamiento de la Pinza diseñada como posible efector final del robot se ha hecho uso de un Servomotor SG90 de 9 g como el mostrado en la Figura 3.2.

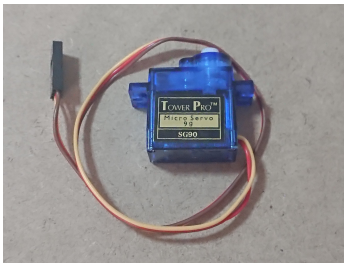

Figura 3.2 Servomotor SG90.

3.1.3 Finales de carrera

Para llevar a los motores paso a paso a una posición concreta conocida se ha hecho uso de finales de carrera de propósito general, de 2 pines, normalmente abiertos, con conector DuPont de 2,54 mm en el extremo, como el mostrado en la Figura 3.3.

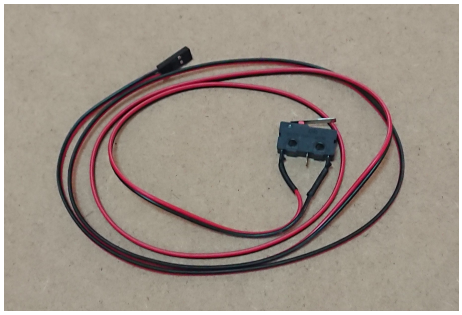

Figura 3.3 Final de Carrera.

3.1.4 Despiece completo de las piezas de plástico diseñadas

Tras haber descrito de forma detallada en el Capítulo 2 las piezas de plástico que componen al robot y sus distintas funciones, y haber recogido en los apartados anteriores las varillas, rodamientos y correas necesarios para el montaje, en la Figura 3.4 se muestra el despiece completo de partes que intervienen en la construcción del robot sin incluir los motores paso a paso ni los finales de carrera.

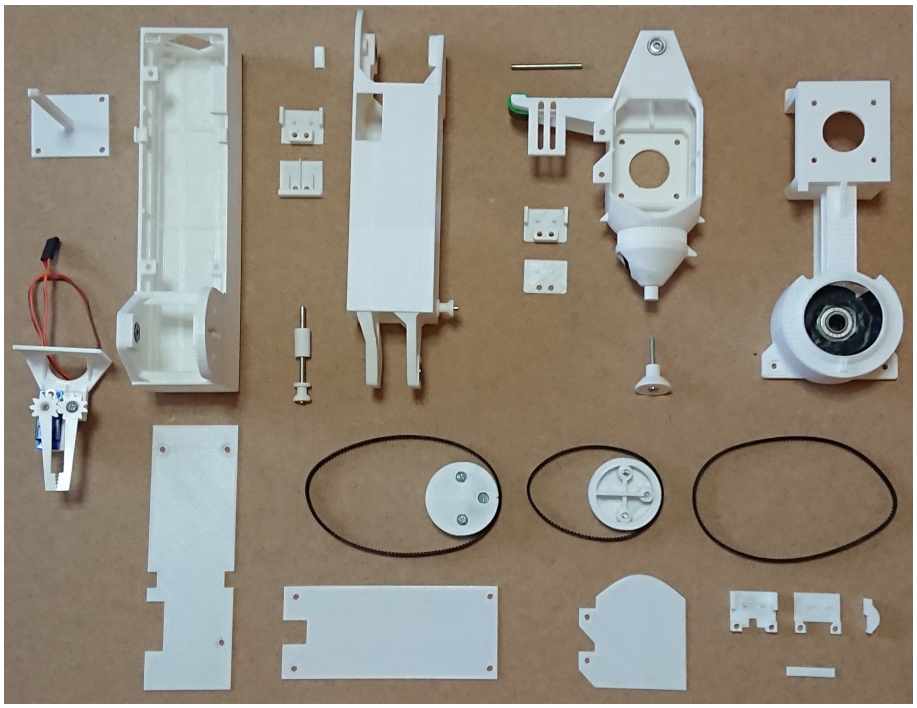

Figura 3.4 Despiece del robot.

3.1.5 Robot montado

Por último, se ha hecho uso de un tablero de madera como mesa de trabajo sobre la que se encuentra el robot, y bajo el que se anclan la Fuente de alimentación y un habitáculo destinado a recoger los cables que salen del robot antes de su llegada al *Escudo*. En la superficie de este tablero se han representado los ejes X e Y asociados al Eslabón 0 que se definirán en el Modelado Cinemático del robot. El montaje completo es el mostrado en la Figura 3.5.

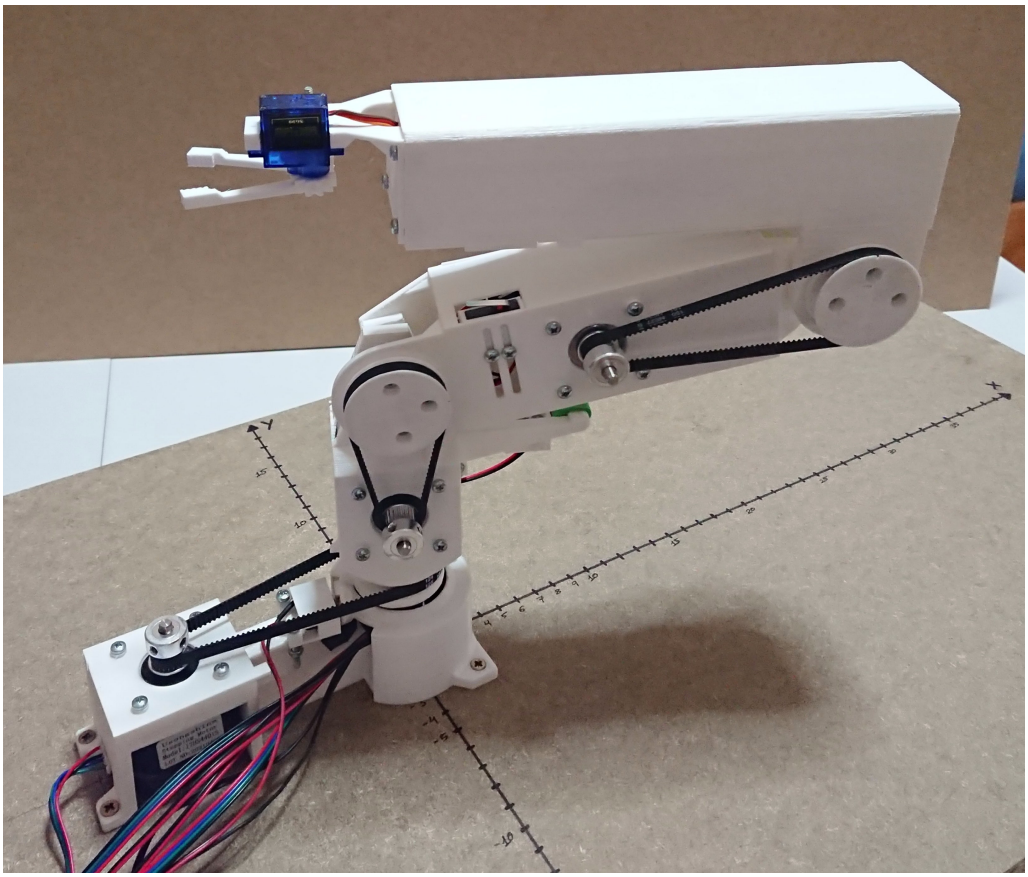

Figura 3.5 Robot montado.

3.2 Hardware de Control

Pasamos, a continuación, a describir los componentes hardware que intervienen en la implementación del control del robot construido.

3.2.1 *Mega 2560 Pro*

Para gestionar el funcionamiento del robot, se ha escogido el *Mega 2560 Pro*. Se trata de una placa electrónica desarrollada por *Robot Dyn®* que monta el microcontrolador *ATmega2560* en una tarjeta de 38 x 55 mm, y que hace uso del adaptador USB-UART de bajo coste *CH340*.

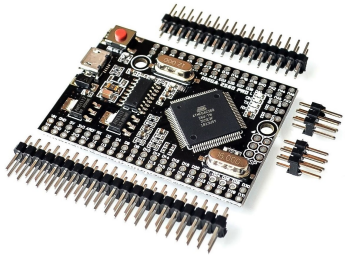

Figura 3.6 *Mega 2560 Pro*.

3.2.2 Drivers *DRV8825*

Para controlar cada motor paso a paso bipolar es necesario el empleo de dos drivers, que deben trabajar con los 12 V de alimentación de dichos motores así como con corrientes más elevadas de lo que puede dar un microcontrolador. Para desempeñar esta tarea, se ha hecho uso del Driver *DRV8825* de *Pololu*, que se trata de un driver capaz de trabajar con hasta 2,5 A que simplifica enormemente el control de un motor paso a paso, encargándose de gestionar esas tensiones y corrientes elevadas, y requiriendo de únicamente dos señales digitales como entradas: la Dirección de giro del motor, y la señal de pulsos asociados a los Pasos. Además, esta versión de driver permite trabajar con un modo de microstepping de 1/32, lo que, unido a los 200 pulsos/revolución de los motores Nema 17 utilizados, hace que se deban dar 6400 pulsos/revolución, proporcionando una magnífica resolución en el control del giro del motor.

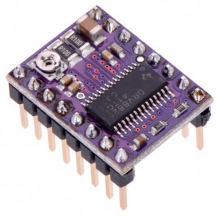

Figura 3.7 Driver *Drv8825*.

En el uso del *DRV8825* es fundamental configurar la corriente máxima del motor al que controla. Para ello, es necesario regular la posición de un potenciómetro ajustando una tensión de referencia que se debe medir con un voltímetro. El fabricante proporciona la fórmula:

$$V_{ref}(V) = I_{max}(A) \cdot 5 \cdot Rs(k\Omega)$$

siendo la Rs del Driver de 100 Ω, por lo que la expresión a usar para obtener la V_{ref} a configurar será:

$$V_{ref}(V) = I_{max}(A) \cdot 5 \cdot 0.1(k\Omega) \Rightarrow$$

$$V_{ref}(V) = \frac{I_{max}(A)}{2}$$

En este proyecto se ha trabajado con dos tipos de motores Nema 17, de 1,68 A y 0,35 A, y las tensiones de referencia configuradas para éstos, guardando cierto margen de seguridad, han sido 0,75 V y 0,17 V respectivamente.

3.2.3 Fuente de alimentación

Para proporcionar los 12 V de alimentación a los tres drivers que intervienen en el funcionamiento del robot, asegurando la capacidad de la fuente de suministrar las corrientes demandables, y teniendo en cuenta lo habitual que resulta encontrarse con fuentes de una hipotética potencia que se deterioran trabajando a valores mucho menores de la capacidad de fábrica, se ha trabajado con una fuente de corriente continua de 12 V y 240 W.

3.2.4 CNC Shield v3

El conexionado de los Drivers al microcontrolador, así como la alimentación de los mismos, y la conexión de éstos a los motores paso a paso quedan muy simplificados mediante el uso un *escudo* como el *CNC Shield v3* de *Protoneer*, diseñado para trabajar con hasta cuatro *Drv8825*, y que incorpora los condensadores que permiten lidiar con los picos de tensión que pueden demandar estos drivers al alimentar a los motores.

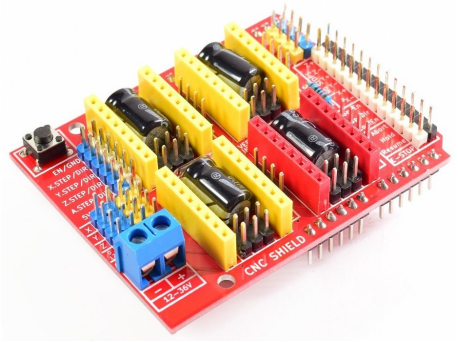

Figura 3.8 CNC Shield v3.

4 Modelado cinemático

En este capítulo se desarrollará la obtención de los Modelos Cinemáticos Directo e Inverso del robot, se definirán los límites del Espacio de Trabajo del mismo, y se analizará la localización de los puntos singulares.

4.1 Grados de libertad y rangos de movimiento

Para realizar el análisis cinemático del robot, se definen en primer lugar los grados de libertad del mismo, sus sentidos de giro y sus respectivos origenes; así como un sistema de referencia solidario a cada uno de los eslabones. En la Figura 4.1 se recoge esta información, habiéndose definido un sistema de referencia adicional para la posición del extremo del efector final, distinto al situado en el extremo del Eslabón 3.

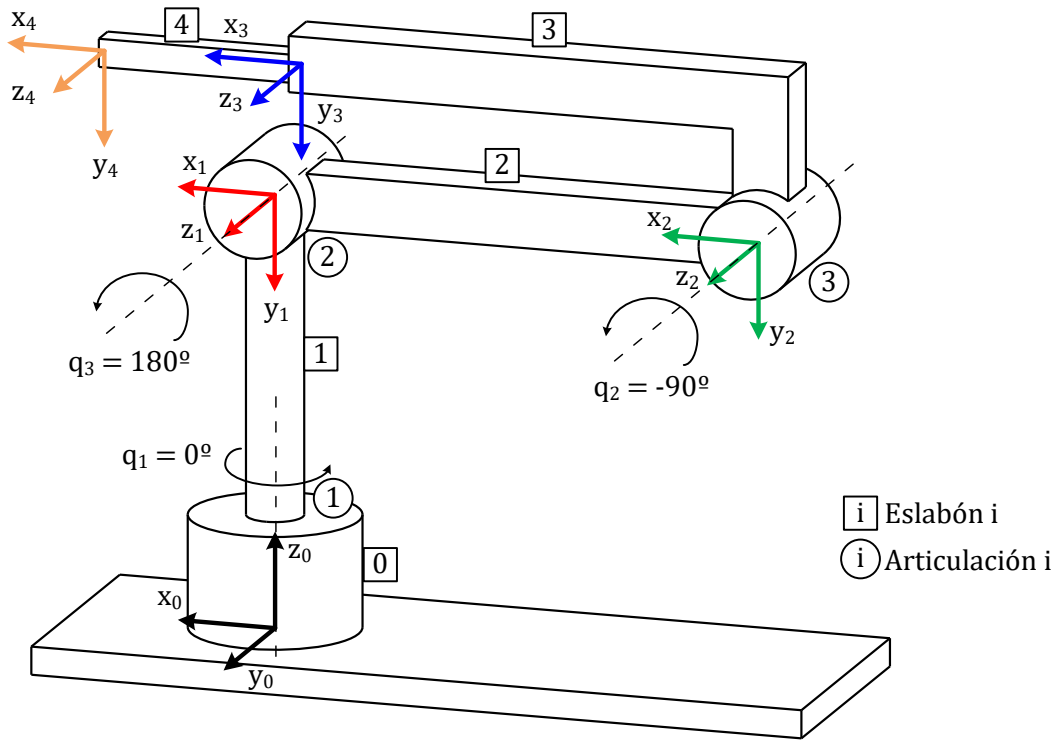

Figura 4.1 Definición de sistemas de referencia y grados de libertad.

De acuerdo con la definición de los grados de libertad realizada, y con las limitaciones físicas del mecanismo que supone la estructura del robot, los rangos de valores que pueden alcanzar físicamente cada una de las coordenadas articulares son los recogidos en la Tabla 4.1.

Tabla 4.1 Rangos de movimiento.

Coordenada articular	Rango
q_1	[-180°, 180°]
q_2	[-90°, 90°]
q_3	[0°, 180°]

4.1.1 Matrices de Transformación Homogéneas

Para caracterizar la localización (posición + orientación) de los sistemas de referencia de los eslabones respecto al sistema de referencia 0 es necesaria la obtención de las Matrices de Transformación Homogénea asociadas.

En general, las MTH pueden funcionar como matrices de cambio de base para vectores de tres componentes $[x,y,z]^T$ que se introduzcan en un vector de cuatro componentes de la forma $[x,y,z,1]^T$, donde la nomenclatura utilizada para denominar a las MTH está asociada a la relación:

$$^0A_1 \equiv P_{\{0\}\leftarrow\{1\}}$$

de manera que, en general, si $r = [x,y,z,1]^T$, entonces:

$$r_{\{0\}} = ^0A_1 \cdot r_{\{1\}}$$

Las MTH básicas son las mostradas a continuación:

$$T_{transl}(x,y,z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{rotx}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\text{sen}(\theta) & 0 \\ 0 & \text{sen}(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{roty}(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \text{sen}(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\text{sen}(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{rotz}(\theta) = \begin{bmatrix} \cos(\theta) & -\text{sen}(\theta) & 0 & 0 \\ \text{sen}(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

En el caso particular de cada robot se pueden obtener las MTH que relacionan a los sistemas de referencia de sus eslabones a partir de la composición de MTH básicas, usando que la postmultiplicación de una MTH a otra equivale a la realización de una transformación sucesiva respecto a ejes móviles.

En la Figura 4.2 se muestran, de manera conjunta: la definición de los parámetros dimensionales realizada en el Capítulo 2 y la definición de las coordenadas articulares aquí realizada.

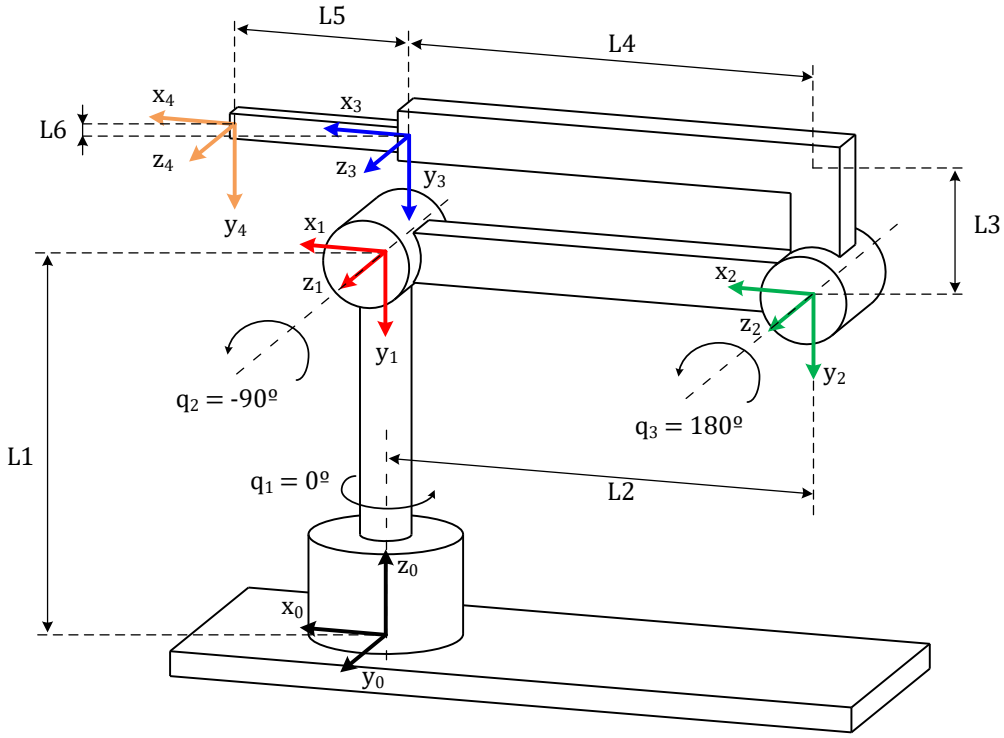

Figura 4.2 Parámetros dimensionales y coordenadas articulares.

De este modo, si se transforma mentalmente cada sistema de referencia {i} hasta colocarlo sobre el sistema {i+1}, se obtienen las matrices:

$$
\begin{aligned}
{}^{0}A_1 &= T_{rotz}(q_1)\cdot T_{transl}(0,0,L_1)\cdot T_{rotx}(-\pi/2)\\
{}^{1}A_2 &= T_{rotz}(\pi/2+q_2)\cdot T_{transl}(-L_2,0,0)\\
{}^{2}A_3 &= T_{rotz}(\pi+q3)\cdot T_{transl}(L_4,-L_3,0)\\
{}^{3}A_4 &= T_{transl}(L_5,L_6,0)
\end{aligned}
$$

cuyas componentes toman los valores mostrados a continuación:

$$
{}^{0}A_1 = \begin{bmatrix} \cos(q_1) & 0 & -\operatorname{sen}(q_1) & 0 \\ \operatorname{sen}(q_1) & 0 & \cos(q_1) & 0 \\ 0 & -1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{1}A_2 = \begin{bmatrix} -\operatorname{sen}(q_2) & -\cos(q_2) & 0 & L_2\cdot\operatorname{sen}(q_2) \\ \cos(q_2) & -\operatorname{sen}(q_2) & 0 & -L_2\cdot\cos(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{2}A_3 = \begin{bmatrix} -\cos(q_3) & \operatorname{sen}(q_3) & 0 & -L_3\cdot\operatorname{sen}(q_3)-L_4\cdot\cos(q_3) \\ -\operatorname{sen}(q_3) & -\cos(q_3) & 0 & L_3\cdot\cos(q_3)-L_4\cdot\operatorname{sen}(q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$ {}^{3}A_4 = \begin{bmatrix} 1 & 0 & 0 & L_5 \\ 0 & 1 & 0 & L_6 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

La obtención de las MTH anteriores permite programar una útil función en *Matlab*® que represente el esquema de alambres del robot para cualquier combinación de valores de $[q_1, q_2, q_3]$, obteniéndose imágenes como las mostradas en la Figura 4.3.

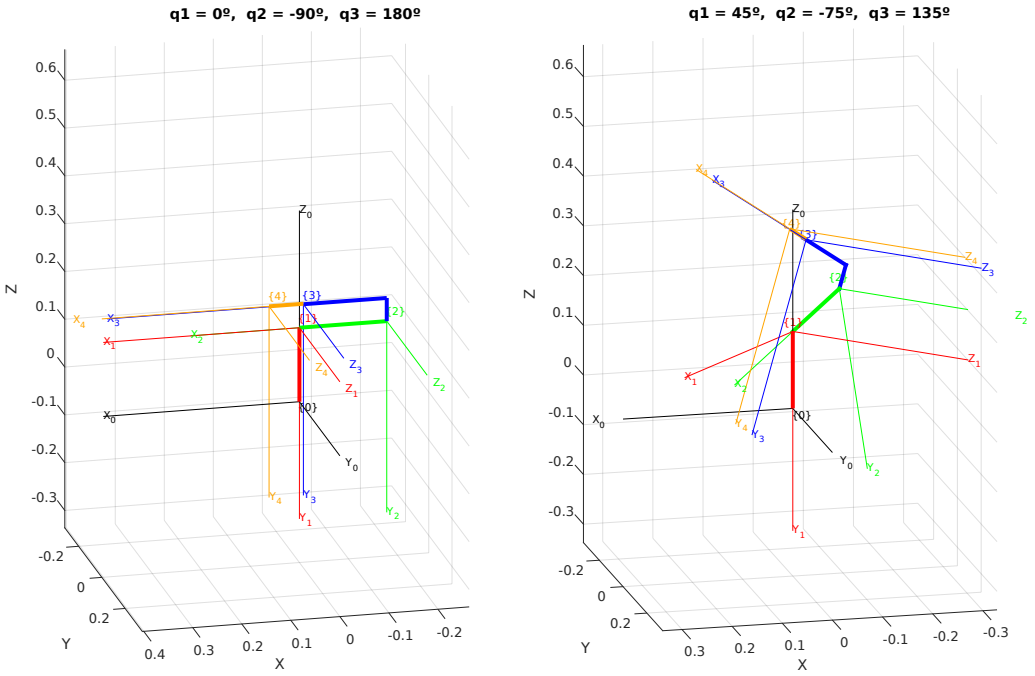

Figura 4.3 Esquema de alambres del robot en distintas posiciones.

4.2 Modelo Cinemático del robot

Se define *cadena cinemática* como un sistema formado por varios cuerpos conectados y con movimiento relativo entre ellos. Y, por otra parte, cuando al menos uno de estos cuerpos no tiene movimiento, denominado entonces *soporte*, a la cadena cinemática se la denomina *mecanismo* [1].

El análisis cinemático de un mecanismo consiste en el estudio del movimiento relativo entre los elementos que lo componen sin considerar las fuerzas que intervienen en dichos movimientos. En esta sección, se desarrollará el estudio de la cinemática directa e inversa del robot construido.

4.2.1 Modelo Cinemático Directo

El Modelo Cinemático Directo de un mecanismo consiste en el conjunto de ecuaciones que permiten obtener la localización (posición + orientación) del extremo final de dicho mecanismo en función de los valores de las coordenadas articulares que determinan la posición de las articulaciones. Este modelo siempre tiene solución única y, en general, se expresa como:

$$ X = f(Q) $$

El robot objeto de estudio cuenta con tres grados de libertad, $[q_1, q_2, q_3]$, por lo que permite controlar tres coordenadas de localización en el espacio, que serán las coordenadas de posición $[x, y, z]$. Por tanto, el Modelo Cinemático Directo buscado responde a la expresión:

$$[x, y, z] = f([q_1, q_2, q_3])$$

Para su obtención, se toma la Matriz de Transformación Homogénea que relaciona al sistema de referencia del efector final con el sistema de referencia asociado a la base del robot:

$${}^0A_4 = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4$$

, y se extraen las tres componentes asociadas a la posición $[x, y, z]$.

Por simplicidad, se trabajará con la nomenclatura:

$s_a \equiv \text{sen}(a)$
$c_a \equiv \cos(a)$
$s_{ab} \equiv \text{sen}(a) + \text{sen}(b)$
$c_{ab} \equiv \cos(a) + \cos(b)$

Entonces, la matriz ${}^0A_4$ viene dada por:

$${}^0A_4 = \begin{bmatrix} s_{23} \cdot c_1 & c_{23} \cdot c_1 & -s_1 & c_1 \cdot (L_6 \cdot c_{23} - L_3 \cdot c_{23} + L_4 \cdot s_{23} + L_5 \cdot s_{23} + L_2 \cdot s_2) \\ s_{23} \cdot s_1 & c_{23} \cdot s_1 & c_1 & s_1 \cdot (L_6 \cdot c_{23} - L_3 \cdot c_{23} + L_4 \cdot s_{23} + L_5 \cdot s_{23} + L_2 \cdot s_2) \\ c_{23} & -s_{23} & 0 & L_1 + L_4 \cdot c_{23} + L_5 \cdot c_{23} + L_3 \cdot s_{23} - L_6 \cdot s_{23} + L_2 \cdot c_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dado que las dimensiones L_3 y L_6 y las dimensiones L_4 y L_5 intervendrán siempre de manera conjunta, se definen los parámetros:

$L_{36} = L_3 - L_6$
$L_{45} = L_4 + L_5$

Con todo esto, la Cinemática Directa (CD) del robot de estudio queda recogida en las expresiones:

$$x = c_1 \cdot (L_2 \cdot s_2 - L_{36} \cdot c_{23} + L_{45} \cdot s_{23}) \tag{4.1}$$

$$y = s_1 \cdot (L_2 \cdot s_2 - L_{36} \cdot c_{23} + L_{45} \cdot s_{23}) \tag{4.2}$$

$$z = L_1 + L_2 \cdot c_2 + L_{36} \cdot s_{23} + L_{45} \cdot c_{23} \tag{4.3}$$

Espacio de Trabajo en coordenadas articulares

Una vez que se conoce una relación entre los valores de las coordenadas articulares y las coordenadas de posición que alcanzará el extremo del robot, conviene hacer uso de ésta para imponer a los valores de $[q_1, q_2, q_3]$ limitaciones más rigurosas que únicamente los rangos físicamente alcanzables por el mecanismo.

De esta forma, se establece que la coordenada z siempre deba ser mayor que 0, y añadimos una restricción que impide la autocolisión del extremo del robot con su propio Eslabón 1. Es inmediato comprobar que el Espacio de Trabajo del robot de estudio presenta simetría de revolución respecto al eje Z_0, por lo que en adelante será conveniente analizar las posibilidades de movimiento del robot trabajando en un plano vertical, válido para cualquier valor de q_1, y en el que los valores de q_2 y q_3 se relacionarán con los de las coordenadas de posición $[r,z]$, donde r se corresponde con la distancia del extremo del robot al eje Z_0:

$$r = \sqrt{x^2 + y^2}$$

Haciendo uso de esta consideración, e imponiendo las restricciones mencionadas, se puede realizar un barrido de los valores de las coordenadas articulares $[q_2,q_3]$ dentro de los rangos que se les asignaron en la Tabla 4.1 y obtener el resultado mostrado en la Figura 4.4.

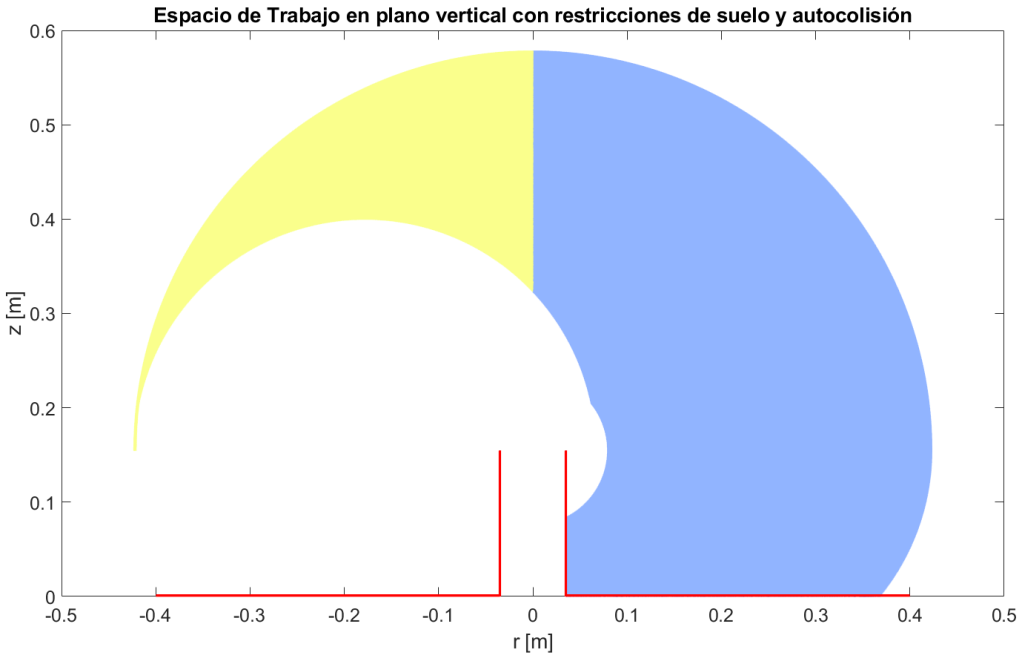

Figura 4.4 Espacio de Trabajo a partir de q_2, q_3 con restricciones de suelo y colisión.

En esta figura se ha hecho distinción entre los casos en que $r \geq 0$ (azul) y $r < 0$ (amarillo) debido a que, por su definición, el modelo Cinemático Inverso desarrollado siempre trabajará con valores de $r \geq 0$, por lo que es conveniente señalar que los puntos representados en amarillo se corresponden con combinaciones de $[q_2,q_3]$ alcanzables dentro de un control en coordenadas articulares pero que nunca se adoptarán en un control en coordenadas cartesianas.

Si se comprueba qué valores de $[q_2,q_3]$ han resultado proporcionar configuraciones del robot que cumplen con las restricciones de suelo y autocolisión impuestas, se obtiene el resultado mostrado en la Figura 4.5, donde el cuadrado rojo encierra a los rangos de estas coordenadas que eran físicamente alcanzables (presentados en la Tabla 4.1); y las regiones azul y amarilla se corresponden con los puntos que sí han resultado alcanzados cumpliendo las restricciones, asociados a las posiciones $[r,z]$ mostradas en la Figura 4.4.

4.2.2 Modelo Cinemático Inverso

El Modelo Cinemático Inverso se corresponde con las relaciones que permiten obtener los valores que deben adoptar las coordenadas articulares de un mecanismo para que su extremo adopte una determinada localización en el espacio. Se trata de un modelo que puede presentar múltiples soluciones, ya que durante su obtención existirán momentos en los que se deba elegir entre distintas vías para calcular el valor de alguna coordenada articular; y, en general, se hace referencia a él como:

$$Q = f(X)$$

En el caso del brazo articulado analizado en este trabajo, se deseará controlar las coordenadas de posición del efector final del robot, por lo que el Modelo Cinemático Inverso buscado responde a la expresión:

$$[q_1,q_2,q_3] = f([x,y,z])$$

Para su obtención, se partirá de las ecuaciones del Modelo Cinemático Directo obtenido en el apartado anterior.

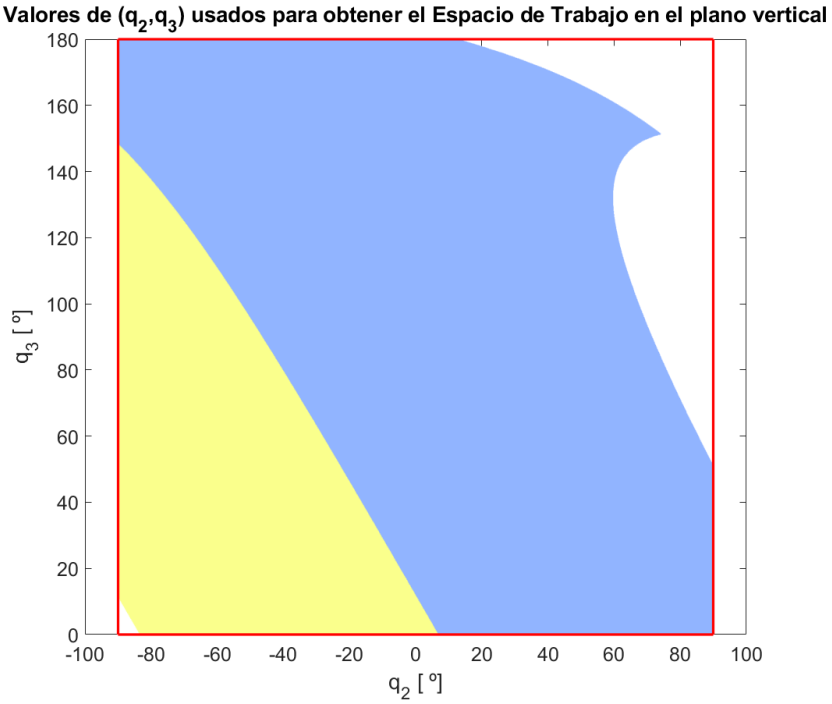

Figura 4.5 Valores de q_2, q_3 alcanzables con restricciones de suelo y colisión.

Obtención de q1

Combinando las ecuaciones (4.1) y (4.2) se obtiene que:

$$\frac{x}{\cos(q_1)} = \frac{y}{\text{sen}(q_1)} \Rightarrow \frac{\text{sen}(q_1)}{\cos(q_1)} = \frac{y}{x} \Rightarrow$$

$$q_1 = \text{atan2}(y,x) \tag{4.4}$$

Análisis de la validez de la expresión de q1

Dado que las coordenadas $[x,y]$ pueden tomar, en principio, cualquier valor $\in \Re$, y que la función atan2 devuelve un ángulo comprendido en [-180°, 180°], el rango de valores que puede devolver la Cinemática Inversa para la coordenada articular q_1 coincide exactamente con el rango de valores alcanzables por esta coordenada, por lo que la expresión de obtención de q_1 es válida.

Obtención de q3

Conocido el valor de q_1, la obtención de q_2 y q_3 se puede plantear como un problema 2-D en el que se conocen los valores de $[r,z]$ asociados a la posición deseada.

Las ecuaciones (4.1) y (4.2) proporcionan de forma equivalente que:

$$\frac{x}{\cos(q_1)} = \frac{y}{\text{sen}(q_1)} = \sqrt{x^2+y^2} = r \Rightarrow$$

$$r = L_2 \cdot s_2 - L_{36} \cdot c_{23} + L_{45} \cdot s_{23} \tag{4.5}$$

Por otra parte, la Ecuación (4.3) se puede reformular como:

$$z - L_1 = L_2 \cdot c_2 + L_{36} \cdot s_{23} + L_{45} \cdot c_{23} \tag{4.6}$$

Sumando las ecuaciones (4.5) y (4.6) elevadas al cuadrado, es decir, $(4.5)^2$ + $(4.6)^2$, se llega a:

$$\frac{r^2 + (z-L_1)^2 - L_2^2 - L_{36}^2 - L_{45}^2}{2 \cdot L_2} = L_{36} \cdot (s_{23} \cdot c_2 - c_{23} \cdot s_2) + L_{45} \cdot (c_{23} \cdot c_2 + s_{23} \cdot s_2)$$

Para una mayor claridad en el procedimiento, se define:

$$K = \frac{r^2 + (z-L_1)^2 - L_2^2 - L_{36}^2 - L_{45}^2}{2 \cdot L_2}$$

Por otra parte, haciendo uso de las expresiones del seno de una suma y del coseno de una suma, se comprueba que:

$$(s_{23} \cdot c_2 - c_{23} \cdot s_2) = \text{sen}(q_2 + q_3 - q_3) = s_3$$
$$(c_{23} \cdot c_2 + s_{23} \cdot s_2) = \cos(q_2 + q_3 - q_2) = c_3$$

Por lo que la ecuación obtenida de hacer $(4.5)^2$ + $(4.6)^2$ equivale a:

$$K = L_{36} \cdot s_3 + L_{45} \cdot c_3$$

Tomamos el cambio de variables:

$$\left.\begin{array}{r} r_3 = \sqrt{L_{36}^2 + L_{45}^2} \\ \gamma_3 = \text{atan2}(L_{36}, L_{45}) \end{array}\right\} \Rightarrow \left\{\begin{array}{l} L_{36} = r_3 \cdot s_{\gamma_3} \\ L_{45} = r_3 \cdot c_{\gamma_3} \end{array}\right.$$

Aplicándolo:

$$\frac{K}{r_3} = c_3 \cdot c_{\gamma_3} + s_3 \cdot s_{\gamma_3} = \cos(q_3 + \gamma_3) \Rightarrow$$

$$\Rightarrow q_3 = \gamma_3 + \text{acos}\left(\frac{K}{r_3}\right)$$

Finalmente, desarrollando la expresión de K y r_3, se obtiene:

$$q_3 = \gamma_3 + \text{acos}\left(\frac{r^2 + (z-L_1)^2 - L_2^2 - L_{36}^2 - L_{45}^2}{2 \cdot L_2 \cdot \sqrt{L_{36}^2 + L_{45}^2}}\right) \tag{4.7}$$

siendo:

$$\gamma_3 = \text{atan2}(L_{36}, L_{45}) \tag{4.8}$$

Análisis de la validez de la expresión de q3

El valor del ángulo γ_3 es una constante, ya que depende únicamente de las dimensiones del robot en cuestión. En el caso del robot construido en este proyecto, se corresponde con $\gamma_3 \approx 11{,}65°$.

El segundo término que interviene en el cálculo de q_3 es un arcocoseno, que devolverá valores comprendidos en [0, 180°], por lo que, en definitiva, los valores de q_3 que devolverá la Cinemática Inversa estarán comprendidos en el rango ≈[11,65 °, 191,65 °].

Dado que el rango de valores en el que podía variar $q3$ era de [0, 180°], el cálculo propuesto para esta coordenada articular es completamente válido y muy adecuado, por aprovechar en gran medida el rango de movimiento de esta articulación.

Obtención de q2

Conocidos los valores de q_1 y q_3, se puede calcular una expresión para q_2 partiendo de las ecuaciones (4.5) o (4.6). Tras haberse estudiado la conveniencia de ambas vías, a continuación se presenta el desarrollo asociado a la que resulta más conveniente que, dado que el valor de r es ≥ 0 por definición, consiste en partir de la Ecuación (4.6). Se tenía:

$$z - L_1 = L_2 \cdot c_2 + L_{36} \cdot s_{23} + L_{45} \cdot c_{23} \Rightarrow$$

Desarrollando el seno y coseno de (q_2 + q_3), y sacando factor común, se obtiene:

$$\Rightarrow z - L_1 = c_2 \cdot (L_2 + L_{36} \cdot s_3 + L_{45} \cdot c_3) + s_2 \cdot (L_{36} \cdot c_3 - L_{45} \cdot s_3)$$

Tomamos:

$$A_2 = L_2 + L_{36} \cdot s_3 + L_{45} \cdot c_3$$
$$B_2 = L_{36} \cdot c_3 - L_{45} \cdot s_3$$

, de forma que:

$$z - L_1 = c_2 \cdot A_2 + s_2 \cdot B_2$$

Haciendo uso del cambio de variables:

$$\left.\begin{array}{r} r_2 = \sqrt{A_2^2 + B_2^2} \\ \gamma_2 = \text{atan2}(B_2, A_2) \end{array}\right\} \Rightarrow \left\{\begin{array}{l} A_2 = r_2 \cdot c_{\gamma_2} \\ B_2 = r_2 \cdot s_{\gamma_2} \end{array}\right.$$

Entonces:

$$\frac{z - L_1}{r_2} = c_2 \cdot c_{\gamma_2} + s_2 \cdot s_{\gamma_2} = \cos(q_2 - \gamma_2)$$

Y finalmente, se obtiene que q_2 puede ser calculada como:

$$q_2 = \gamma_2 + \text{acos}\left(\frac{z - L_1}{\sqrt{A_2^2 + B_2^2}}\right) \tag{4.9}$$

siendo:

$$A_2 = L_2 + L_{36} \cdot s_3 + L_{45} \cdot c_3 \tag{4.10}$$

$$B_2 = L_{36} \cdot c_3 - L_{45} \cdot s_3 \tag{4.11}$$

$$\gamma_2 = \text{atan2}(B_2, A_2) \tag{4.12}$$

Análisis de la validez de la expresión de q2

Para el caso de la expresión obtenida para el cálculo de q_2, no se puede obtener un rango de variación cerrado de la misma manera que se hizo para el caso de q_3 debido a que, en esta ocasión, el valor de γ_2 no es una constante sino que depende del valor que esté tomando en cada caso otra de las variables articulares, q_3. La validez de la expresión nos la proporcionarán, entonces, las pruebas posteriores que se realicen haciendo uso de la misma. Sí es posible adelantar, sin embargo, que el ángulo γ_2 puede tomar en principio cualquier valor, ya que tanto A_2 como B_2 podrían tomar valores positivos y negativos de acuerdo con sus expresiones; y dado

que el segundo de los términos que interviene en el cálculo de q_2 es un arcocoseno, éste devolverá valores comprendidos en [0, 180°]; por lo que en definitiva, el rango de valores que q_2 podía tomar físicamente, que era de [-90°, 90°], estará con seguridad comprendido en los posibles valores que devuelva la Cinemática Inversa para esta coordenada articular.

4.3 Espacio de Trabajo

El conocimiento de un Modelo Cinemático Inverso del robot tiene como objetivo servir de herramienta para la posterior programación de un control al que se le pueda solicitar que el extremo del robot se sitúe en un punto dado por sus coordenadas $[x, y, z]$, y que dicho control sea capaz de determinar la configuración que deben adoptar las coordenadas articulares $[q_1, q_2, q_3]$ para lograr ese objetivo.

Sin embargo, el control en cuestión también requerirá de algún medio para discernir los puntos $[x, y, z]$ que el mecanismo del robot es físicamente capaz de alcanzar. Es por esto que resulta necesario analizar el Espacio de Trabajo del robot, y definir matemáticamente los límites del mismo en función de las coordenadas cartesianas.

Como ya se ha mencionado anteriormente, el Espacio de Trabajo del robot tiene simetría de revolución respecto al eje Z_0, pudiendo además la coordenada q_1, que es la asociada a la rotación en torno a ese eje, tomar cualquier valor dentro de una revolución completa ($q_1 \in$ [-180°, 180°]). Analizando, entonces, el Espacio de Trabajo del robot en un plano vertical $[r,z]$, se deben definir límites para éste teniendo en cuenta las restricciones de: suelo, autocolisión, y extensiones máxima y mínima del brazo.

Esta última restricción cambia en función de la región del plano $[r,z]$ en la que se encuentre el extremo del robot, según pueda estar saturada una u otra coordenada articular (q_2 o q_3) a su valor mínimo ó máximo; lo que hace necesario dividir el Espacio de Trabajo en regiones entre distintas alturas z para poder establecer los límites a imponer a las coordenadas $[r,z]$. La Figura 4.6 recoge la representación de estas regiones, así como la representación de los límites del Espacio de Trabajo que se aplican cuando el robot se encuentra dentro de cada una de ellas.

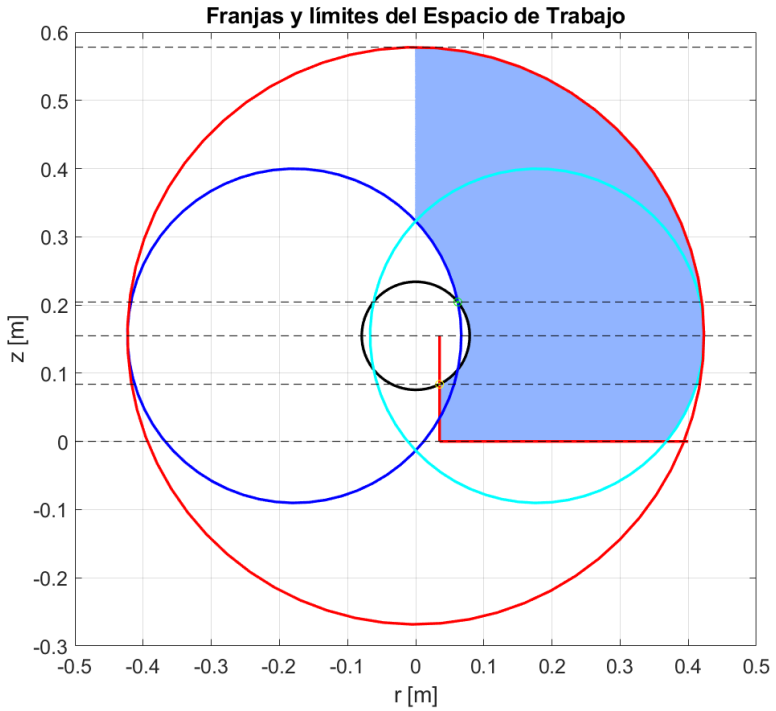

Figura 4.6 Franjas y límites del Espacio de Trabajo en el plano $[r,z]$.

A continuación, se detallan las definiciones de las circunferencias que, junto a la línea horizontal asociada al suelo y la línea vertical asociada a la autocolisión, delimitan el Espacio de Trabajo del robot:

Definición de las circunferencias límites

- **Circunferencia "negra":** q_3 se encuentra saturada a 180° (Eslabón 3 apoyado sobre el Eslabón 2), y q_2 varía. Queda definida por:
  - Centro: $[r,z] = [0, L_1]$
  - Radio: $R = \sqrt{a^2 + b^2}$, con:
  - $a = L_{45} - L_2$
  - $b = L_{36}$
- **Circunferencia "azul":** q_2 se encuentra saturada a -90°, y q_3 varía. Queda definida por:
  - Centro: $[r,z] = [-L_2, L_1]$
  - Radio: $R = \sqrt{a^2 + b^2}$, con:
  - $a = L_{45}$
  - $b = L_{36}$
- **Circunferencia "cian":** q_2 se encuentra saturada a 90°, y q_3 varía. Queda definida por:
  - Centro: $[r,z] = [L_2, L_1]$
  - Radio: $R = \sqrt{a^2 + b^2}$, con:
  - $a = L_{45}$
  - $b = L_{36}$
- **Circunferencia "roja":** q_2 y q_3 se combinan para lograr en todo momento la máxima extensión del brazo, es decir, la máxima separación del efector final respecto al eje Z_0. Queda definida por:
  - Centro: $[r,z] = [0, L_1]$
  - Radio: $R = L_2 + \sqrt{L_{45}^2 + L_{36}^2}$

Se presentan, a continuación, las expresiones matemáticas de los valores de altura z que delimitan las franjas que se distinguen dentro del Espacio de Trabajo, así como las restricciones que se deben imponer a los valores de $[r,z]$ para que el punto dado en coordenadas cartesianas esté dentro del Espacio de Trabajo. Conviene definir, previamente, los siguientes parámetros:

- $Z_1 = L_1 - \sqrt{(L_{45} - L_2)^2 + L_{36}^2 - colision^2}$: altura a la que se produce la intersección entre la circunferencia negra y la vertical de autocolisión.
- $Z_2 = L_1 + L_{36}$: altura a la que se produce la intersección entre la circunferencia negra y la circunferencia azul.
- $z_{max} = L_1 + L_2 + \sqrt{L_{45}^2 + L_{36}^2}$: valor máximo que puede alcanzar la coordenada z del extremo del robot.

Alturas de cambio de franja y límites a aplicar

- $0 \leq z \leq Z_1$
  - $r > colision$
  - $[r,z]$ dentro de la circunferencia cian
- $Z_1 < z \leq L_1$
  - $[r,z]$ fuera de la circunferencia negra
  - $[r,z]$ dentro de la circunferencia cian

- $L_1 < z \leq Z_2$
  - [r,z] fuera de la circunferencia negra
  - [r,z] dentro de la circunferencia roja
- $Z_2 < z \leq z_{max}$
  - [r,z] fuera de la circunferencia azul
  - [r,z] dentro de la circunferencia roja

Estas restricciones serán posteriormente traducidas a código en la programación del control del robot, para poder evitar intentar alcanzar puntos que queden fuera del Espacio de Trabajo del robot.

4.4 Análisis de los puntos singulares

Tras obtener los Modelos Cinemáticos Directo e Inverso de un brazo manipulador, conviene siempre obtener las expresiones de sus Matrices Jacobianas Directa e Inversa, así como realizar un estudio de los posibles puntos singulares.

4.4.1 Jacobiana Directa

La matriz Jacobiana Directa es aquella que permite obtener las velocidades de las coordenadas cartesianas del extremo del robot a partir de las velocidades de las coordenadas articulares. En general, esta relación se expresa como:

$$\dot{X} = J \cdot \dot{Q}$$

En el caso del robot de estudio:

$$\dot{Q} = \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \\ \dot{q_3} \end{bmatrix} \qquad \dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$$

Y las componentes del Jacobiano Directo se corresponden con:

$$J = \begin{bmatrix} \frac{\partial f_x}{\partial q_1} & \frac{\partial f_x}{\partial q_2} & \frac{\partial f_x}{\partial q_3} \\ \frac{\partial f_y}{\partial q_1} & \frac{\partial f_y}{\partial q_2} & \frac{\partial f_y}{\partial q_3} \\ \frac{\partial f_z}{\partial q_1} & \frac{\partial f_z}{\partial q_2} & \frac{\partial f_z}{\partial q_3} \end{bmatrix}$$

, donde f_x, f_y y f_z son expresiones que definen el Modelo Cinemático Directo del robot:

$$x = f_x(q_1, q_2, q_3) = c_1 \cdot (L_2 \cdot s_2 - L_{36} \cdot c_{23} + L_{45} \cdot s_{23})$$

$$y = f_y(q_1, q_2, q_3) = s_1 \cdot (L_2 \cdot s_2 - L_{36} \cdot c_{23} + L_{45} \cdot s_{23})$$

$$z = f_z(q_1, q_2, q_3) = L_1 + L_2 \cdot c_2 + L_{36} \cdot s_{23} + L_{45} \cdot c_{23}$$

Realizando las derivadas parciales correspondientes, se obtiene la matriz Jacobiana Directa:

$$J = \begin{bmatrix} -s_1 \cdot (L_2 \cdot s_2 - L_{36} \cdot c_{23} + L_{45} \cdot s_{23}) & c_1 \cdot (L_2 \cdot c_2 + L_{36} \cdot s_{23} + L_{45} \cdot c_{23}) & c_1 \cdot (L_{36} \cdot s_2 3 + L_{45} \cdot c_{23}) \\ c_1 \cdot (L_2 \cdot s_2 - L_{36} \cdot c_{23} + L_{45} \cdot s_{23}) & s_1 \cdot (L_2 \cdot c_2 + L_{36} \cdot s_{23} + L_{45} \cdot c_{23}) & s_1 \cdot (L_{36} \cdot s_2 3 + L_{45} \cdot c_{23}) \\ 0 & -L_2 \cdot s_2 + L_{36} \cdot c_{23} - L_{45} \cdot s_{23} & L_{36} \cdot c_2 3 - L_{45} \cdot s_{23} \end{bmatrix}$$

4.4.2 Jacobiana Inversa

La matriz Jacobiana Inversa proporciona la relación contraria a la Jacobiana Directa, de manera que permite obtener las velocidades de las coordenadas articulares a partir de las velocidades de las coordenadas cartesianas del extremo del robot. En general, esta relación se expresa como:

$$\dot{Q} = J^{-1} \cdot \dot{X}$$

, siendo:

$$J^{-1} = \frac{1}{\det(J)} \cdot [\text{Adj}(J)]^T$$

Los puntos singulares son aquellos en los que se produce que $\det(J) = 0$, y en ellos incrementos infinitesimales de los valores de las coordenadas cartesianas $[x,y,z]$ implican incrementos infinitos de los valores de las coordenadas articulares $[q_1, q_2, q_3]$, ya que si:

$$\left.\begin{array}{l} \dot{Q} = J^{-1} \cdot \dot{X} \\ \\ \det(J) = 0 \end{array}\right\} \quad \Rightarrow \quad J^{-1} \to \infty \quad \Rightarrow \quad \dot{Q} \to \infty$$

Para el caso del robot construido, el determinante del Jacobiano Directo viene dado por:

$$\det(J) = \frac{1}{2} \cdot \left[L_2 \cdot L_{36}^2 \cdot c_2 - L_2^2 \cdot L_{36} \cdot s_{23} + L_2 \cdot L_{45}^2 \cdot c_2 + L_2 \cdot L_{36}^2 \cdot \cos(q_2 + 2 \cdot q_2) + L_2^2 \cdot L_{45} \cdot \cos(q_2 - q_3) - \right.$$
$$\left. - L_2 \cdot L_{45}^2 \cdot \cos(q_2 + 2 \cdot q_3) - L_2^2 \cdot L_{36} \cdot \text{sen}(q_2 - q_3) - L_2^2 \cdot L_{45} \cdot c_{23}\right] - L_2 \cdot L_{36} \cdot L_{45} \cdot \text{sen}(q_2 + 2 \cdot q_3) \quad (4.13)$$

Se observa, en primer lugar, que su expresión no depende de la coordenada articular q_1, reafirmando el hecho de que el Espacio de Trabajo del robot presentaba simetría de revolución en torno al eje de giro de la Articulación 1.

Dada la complejidad de la expresión obtenida, no es posible resolver analíticamente la ecuación $\det(J) = 0$, por lo que se ha realizado un barrido en *Matlab*® del Espacio de Trabajo del robot detectando los puntos en los que $\det(J)$ se hace 0 o muy próximo a 0, obteniéndose la el resultado mostrado en la Figura 4.7.

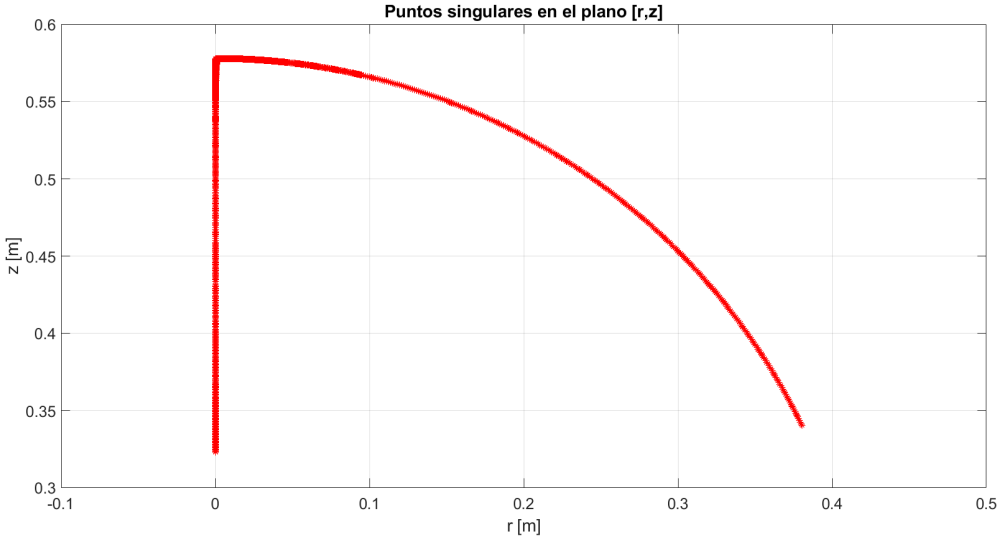

Figura 4.7 Puntos singulares en el plano $[r,z]$.

La Figura 4.7 ha sido obtenida realizando un barrido de las coordenadas articulares q_2 y q_3 con paso de 0.003 radianes, y considerando que el $\det(J)$ es suficientemente próximo a 0 según un $\varepsilon = 10^{-5}$. Para valores menores de estos parámetros, la obtención de los resultados se vuelve demasiado lenta, pero la imagen aquí mostrada permite extraer las siguientes conclusiones que, por otra parte, resultaban intuibles:

Los puntos singulares del mecanismo se dan, para cualquier valor de q_1, en:

- **Todos los puntos con r=0**: puntos que se encuentran sobre el eje Z_0, ya que si por ejemplo el robot se está moviendo a lo largo de una línea de puntos con coordenada x=0, cuando se pase de y>0 a y<0, el robot necesitaría dar en un instante una semivuelta completa sobre su Articulación 1 para poder continuar la línea.
- **La bóveda descrita por el robot cuando está completamente extendido**: donde el robot no puede estirarse más para moverse de una posición a otra porque ya está al límite de sus dimensiones.

Ambas posibilidades serán tratadas en la programación del control del robot para evitar las posibles consecuencias indeseadas que provocaría el hecho de que los motores intentasen imprimir velocidades tendientes a infinito a las coordenadas articulares del robot.

5 Control del robot

En este capítulo se explicarán los fundamentos y el funcionamiento del código que hace posible, desde el control de bajo nivel de los motores paso a paso del robot, hasta la ejecución de movimientos de distintas características coordinando a los tres motores durante el trayecto.

A lo largo de este capítulo, se hará frecuentemente referencia a un agente cuyo papel es conveniente aclarar antes de comenzar a dirigirse a él, y es el denominado:
·***Usuario***: término con el que se hará referencia a la persona hipotética que hace uso del robot desarrollado en este trabajo una vez que éste cuenta con todo el código generado en este proyecto, sin tener acceso al mismo, sino solo utilizando el modo Manual y las funciones aquí preparadas para poder programar las tareas del robot. En una situación real, se correspondería con el individuo encargado de programar un robot industrial que hubiera sido adquirido por una empresa para utilizarlo en su cadena de producción.

5.1 Entorno de desarrollo

Como se ha introducido en el Capítulo 3, para la programación del funcionamiento del robot se ha hecho uso del *Mega 2560 Pro* de *Robot Dyn®*. Esta tarjeta, aunque no está desarrollada por *Arduino*, es compatible con el entorno de desarrollo de esta empresa, por lo que para su programación se hará uso del *Open-source Arduino Software (Arduino IDE)*, y por consiguiente el código generado se encuentra en lenguaje C++.

Figura 5.1 Logo de Arduino.

5.2 Control de bajo nivel de motores paso a paso

El control de bajo nivel que se hace de los motores paso a paso consiste en un control en velocidad, que se logra mediante la gestión del intervalo de tiempo que se deja transcurrir entre cada pulso que se envía al motor y el siguiente, pudiendo controlarse la frecuencia con la que se producen los pasos. Aún por debajo de este nivel de control se encontraría la gestión que realiza el *Drv8825* utilizado en este proyecto, que se encarga de energizar convenientemente las bobinas del motor para materializar los micropasos que se desea ejecutar.

Es importante tener en cuenta que la relación entre la velocidad de giro del motor y ese intervalo de tiempo entre pulsos no es lineal, sino inversamente proporcional:

$$v[rad/s] = \frac{1}{\Delta t_{pulso}[s]} \cdot RPP$$

, siendo:

·*RPP*: "Radianes Por Paso": constante que relaciona el ángulo de giro del eje de la articulación con el giro del motor por cada pulso recibido, teniendo en cuenta que los motores utilizados presentan 200 pasos/revolución y que se trabaja con reductoras de 1:3 y una configuración de micropasos de 1/32.

5.2.1 Gestión de las aceleraciones

En el control de los motores paso a paso es fundamental hacer una adecuada gestión de las aceleraciones del eje del motor, ya que los cambios bruscos de velocidades provocan la pérdida de pasos en el movimiento y el consiguiente mal funcionamiento del motor, que puede presentar comportamientos extraños e incluso llegar a bloquearse si la frecuencia de los pulsos que se entrega al Driver aumenta de forma repentina. Es por eso que en este trabajo se va a hacer uso de perfiles de velocidades trapezoidales, es decir, se va a trabajar con movimientos en los que los cambios de velocidad serán tales que la velocidad evolucionará linealmente, lo que implica aceleraciones constantes y evoluciones de las coordenadas articulares parabólicas.

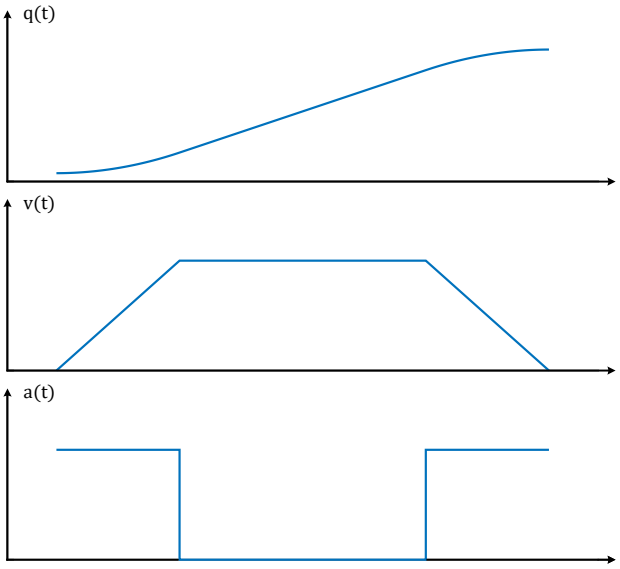

Figura 5.2 Evolución con perfil de velocidad trapezoidal.

Como se verá a lo largo de este capítulo, la tipología de movimiento anterior será la utilizada para el diseño de las trayectorias de referencia que permitan el paso suavizado por los puntos proporcionados por el usuario, las cuales deberán ser seguidas por los motores en sus movimientos haciendo uso de un bucle de control.

5.2.2 Control en posición: Controlador Proporcional con actuación en velocidad

Ha de tenerse en cuenta que el control en velocidad de los motores paso a paso presenta las siguientes limitaciones:

- Introducción de error acumulativo debido a la pérdida de precisión en los cálculos: el diferencial de tiempo, dT, que se deja transcurrir entre dos pulsos es un número entero dado en microsegundos al

microcontrolador, por lo que en general nunca será exactamente el que debiera ser para cumplir con la velocidad instantánea deseada en cada momento.

- Introducción de error acumulativo debido a la naturaleza de la variación de la velocidad: en los tramos del movimiento con aceleraciones constantes, la velocidad no varía realmente de forma lineal sino escalonada, ya que el dT aplicado debe cumplir con su duración concreta antes de volver a aumentarse o reducirse, de manera que se introduce error en el movimiento y se produce una pérdida o ganancia de pasos indeseada constantemente.
- Introducción de error acumulativo debido al funcionamiento del microconstrolador: los pequeños retrasos que se puedan producir entre el instante ideal en que se debería calcular un nuevo valor del dT y comenzar a contar el transcurso de éste, y el momento real en que el microcontrolador sea capaz de empezar a realizar esos cálculos, sumado al período de tiempo necesario para realizarlos, provoca la introducción de retrasos en el movimiento y la consiguiente pérdida de pasos respecto a la posición ideal que debería tener el motor.

Para garantizar alcanzar los puntos de destino en los movimientos y tener un control total sobre la posición del robot en todo momento, se realizará un control en posición de las coordenadas articulares asociadas a cada articulación del mecanismo. Se hace, entonces, necesario introducir un mecanismo que sea capaz de corregir la aparición de estos errores durante la propia ejecución del movimiento, ya que, de lo contrario, las diferencias entre las posiciones alcanzadas y las posiciones deseadas al final de cada trayecto se hacen completamente imprevisibles, y las discrepancias entre los instantes de finalización del giro de los distintos motores se vuelven mayores cuanto más distintas sean las velocidades que estos adoptan durante el desplazamiento.

Debido a su rápida dinámica tras recibir un pulso, un motor paso a paso puede tratarse como un sistema a controlar que funciona como un integrador, siendo la señal de entrada la velocidad y la de salida la posición:

$$\frac{q(s)}{v(s)} = \frac{1}{s}$$

Por tanto, si se le aplica un Control proporcional $C(s) = P$ en bucle cerrado se obtiene un sistema de primer orden cuya dinámica se puede ajustar mediante la sintonización de la ganancia P, y que nos permite controlar al motor en posición usando: las velocidades como acción de control y el conteo que realiza el microcontrolador de los pulsos dados como señal de realimentación.

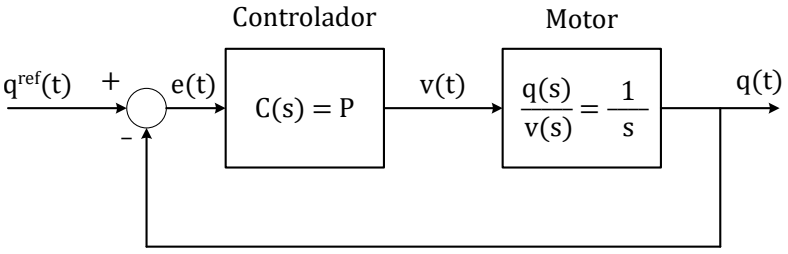

Figura 5.3 Bucle de control de los motores paso a paso.

Al tratarse de un control en bucle cerrado, el propio sistema (motor controlado) es capaz de corregir durante la ejecución del movimiento la posible pérdida de pasos que se pueda ocasionar por cualquier causa, y se consigue una gran sincronía entre los movimientos de los distintos motores tanto durante la ejecución de los movimientos como en el alcance del punto final de destino, logrando que éstos completen sus trayectos de forma prácticamente simultánea.

Con esta estrategia de control, las trayectorias con perfiles de velocidades trapezoidales anteriormente mencionadas pasarán a ser, por tanto, la señal de referencia que reciba el controlador de cada motor que, debido a su rápida dinámica, conseguirá adaptarse a las curvas deseadas con suficiente precisión.

Sintonización de la ganancia P

Para llevar a cabo la sintonización del valor de P en base a técnicas de control, se parte del siguiente desarrollo:

$$G(s) = \frac{1}{s} \qquad C(s) = P$$

$$G_{bc}(s) = \frac{C(s)\cdot G(s)}{1 + C(s)\cdot G(s)} = \frac{1}{1+\frac{s}{P}} = \frac{1}{1+\tau\cdot s} \quad \Rightarrow \quad \tau = \frac{1}{P}$$

, donde τ es la constante de tiempo del sistema en bucle cerrado.

Dado que no se disponía a priori de un valor concreto deseado para imponer como constante de tiempo, se realizaron pruebas experimentando con los valores de P y dando referencias en rampa a la posición deseada de los motores, pudiendo comprobarse que:

- Valores muy pequeños de P provocan una respuesta demasiado lenta en los cambios de velocidad, dejando de cumplirse con las aceleraciones deseadas que presentan las curvas de las señales de referencia para las coordenadas articulares.
- Valores demasiados grandes de P provocan que la actuación sea demasiado brusca y, consecuentemente, el valor del error esté permanentemente conmutando entre valores grandes y pequeños, provocando que el dT calculado por el control también haga lo propio, y obteniéndose un comportamiento indeseado del motor, que presenta un giro lleno de "traqueteos".

Por estas razones, se decidió realizar una sintonización empírica del valor de la P, basada en la calidad del sonido generado por el motor durante un movimiento de los motores a velocidad controlada y constante. De esta forma:

- Se preparó un pequeño programa que, sin hacer uso de esta estrategia de control, imprimiese un tren de pulsos a una frecuencia conocida y constante, asociada a una velocidad de giro deseada.
- Se preparó otro pequeño programa que, haciendo uso de la codificación del control proporcional, recibiese como referencia una señal en rampa cuya pendiente estuviese asociada a la velocidad de giro utilizada en el primer programa.
- Se ajustó el valor de la P hasta lograr que el característico sonido generado por el motor en ambos movimientos fuese el mismo, comprobándose además que, para ese valor de P, el proceso de aceleración inicial al arranque del movimiento fuese progresivo.

El valor de la ganancia P resulta ser de 0,0001 μs^{-1}. Por tanto, la constante de tiempo que presenta el sistema para la P escogida está asociada a P = 100 s^{-1}, y vale: $\tau = \frac{1}{100} = 0{,}01\,s$

Bucle de control

Conocida la ganancia P, los cálculos que se deben realizar para controlar un motor paso a paso son, en bucle:

- Evaluar el valor instantáneo de la referencia, q_i^{ref}.
- Calcular el error en posición: $e_i = q_i^{ref} - q_i$.
- Calcular la velocidad a adoptar por el motor: $v_i = P\cdot e_i$
- Obtener el intervalo de tiempo asociado a la velocidad calculada: $dT_i = \frac{1}{v_i}\cdot RPP$.
- Dar un pulso al motor (un paso), y volver a esperar hasta que se cumpla el dT_i.

5.3 Control del movimiento coordinado de los motores

En esta sección se explicarán los puntos más importantes del código que permite controlar el funcionamiento del robot.

5.3.1 Unidades utilizadas en el código

Antes de comenzar la explicación, es conveniente listar las unidades en las que se encuentran los distintos parámetros y variables del código generado. En todos los casos, éstas unidades se han escogido tales que permitan minimizar la cantidad de operaciones a realizar por el microcontrolador, así como que el rango de valores que vayan a tomar para el caso del robot de estudio sea adecuado teniendo en cuenta el tipo de la variable en cuestión:

- Coordenadas articulares: en [pasos].
- Referencia generada para las coordenadas articulares: en [radianes].
- Velocidades de giro de los motores y otros parámetros de velocidad: en [radianes/s].
- dT y otros parámetros de tiempo: en [μs].

5.3.2 Esquema de relación entre las funciones

La cantidad de funciones programadas y la complejidad de las relaciones entre éstas ha ido aumentando a medida que se generaba el código de control del robot, por lo que antes de comenzar con la explicación del funcionamiento concreto de las principales funciones, conviene incluir un esquema explicativo de las relaciones entre éstas, que pueda servir de ayuda, especialmente, a quien acuda al código para consultarlo. Éste esquema es el mostrado en la Figura 5.4.

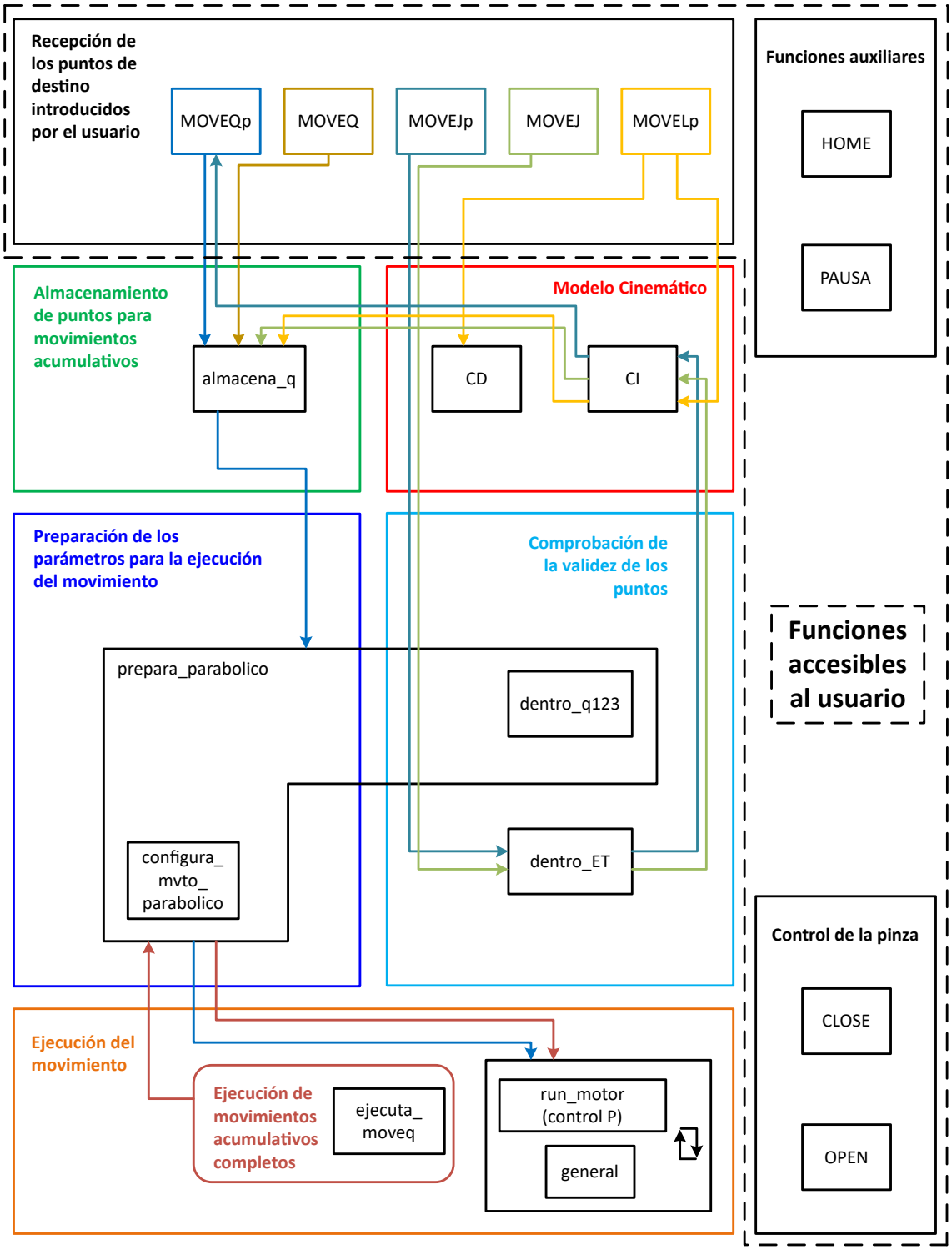

Figura 5.4 Esquema de relación entre las funciones del control.

5.3.3 Ejecución del movimiento

Como se puede observar en el esquema de la Figura 5.4, el uso de todas las funciones de tipo MOVEx presenta un bloque final común, que consiste en la ejecución en bucle de la función ***run_motor*** junto a la función ***general***, lo cual se realiza hasta que los tres motores hayan completado su movimiento.

La función ***general*** simplemente se encarga de comprobar la posible pulsación de los botones que permiten gestionar el cambio entre los modos de funcionamiento del robot, de los que se hablará en el Capítulo 6; mientras que la función ***run_motor*** es llamada una vez para cada motor, y se encarga de ejecutar el bucle de control ya explicado, donde la evaluación del valor instantáneo de la referencia se hace a partir de los parámetros generados y almacenados por la función ***configura_mvto_parabolico*** para el movimiento de la coordenada articular q_i correspondiente al motor en cuestión.

5.3.4 Gestión del paso del tiempo

Todas las funciones, tanto internas como accesibles al usuario, programadas en este proyecto son tales que en ningún caso deben bloquear ni acaparar al microcontrolador. Esto cobra especial importancia en el caso de la función que controla los pulsos enviados a cada motor durante los movimientos del robot, ***run_motor***, que además debe trabajar con unos intervalos de tiempo dT_i que resultan ser variables.

Dicha función trabaja con la medida del paso del tiempo que hace el propio *ATMega 2560*, de manera que durante la realización de cualquier trayecto, ***run_motor*** está siendo llamada permanentemente para los tres motores, realizando los siguientes pasos:

- Leer el valor actual del tiempo con la función *micros()*.
- Comprobar si ha transcurrido el dT_i que se debía esperar desde el último pulso que recibió este motor.
  - En caso negativo, se sale directamente de la función para liberar al microcontrolador.
  - En caso afirmativo, se ejecuta el bucle de control para ese motor, actualizando el valor de dT_i y dando un nuevo pulso, para finalmente volver a salir de la función y liberar al microcontrolador.

Aunque la función ***run_motor*** es llamada para cada motor con una alta frecuencia, la estrategia aquí seguida provocará que habitualmente exista un pequeño retraso entre el instante ideal en que se debían realizar los nuevos cálculos del bucle de control de cada motor y el instante real en que esto se realicen. Sin embargo, como ya se explicó en la Sección 5.2, los errores que introducirían estos retrasos son automáticamente corregidos durante la ejecución del movimiento gracias a la estrategia de control en bucle cerrado seguida.

5.3.5 Fundamentos del movimiento parabólico

La generación de referencias parabólicas (es decir, con perfiles de velocidades trapezoidales) utilizada en este proyecto se basa en las ideas presentadas en [2].
Desde el punto de vista conceptual, el generador de trayectorias parabólico define la trayectoria que une a tres puntos $(t_j, q_{i,j})$ de una coordenada articular q_i como la curva formada por: dos tramos rectilíneos que parten de y llegan a los puntos primero y tercero, unidos por un tramo parabólico intermedio cuya curvatura está limitada por una aceleración fija a que se establece que puede experimentar el eje de giro del motor. De esta forma, la trayectoria diseñada pasará más cerca del punto intermedio cuanto mayor sea esa aceleración permitida. La Figura 5.5 ilustra la curva descrita, así como el papel de las variables involucradas en el cálculo de cada tramo de la trayectoria.

Las funciones que definirían a cada tramo de la curva de la Figura 5.5 son las siguientes:

$$q(t) = \begin{cases} q_0 + \frac{q_1 - q_0}{T_1} t & 0 \leq t < T_1 - \tau \\ q_1 + \frac{(q_1 - q_0)}{T_1}(t - T_1) + \frac{a}{2}(t - T_1 + \tau)^2 & T_1 - \tau \leq t < T_1 + \tau \\ q_1 + \frac{q_2 - q_1}{T_2}(t - T_1) & T_1 + \tau \leq t \leq T_1 + T_2 \end{cases}$$

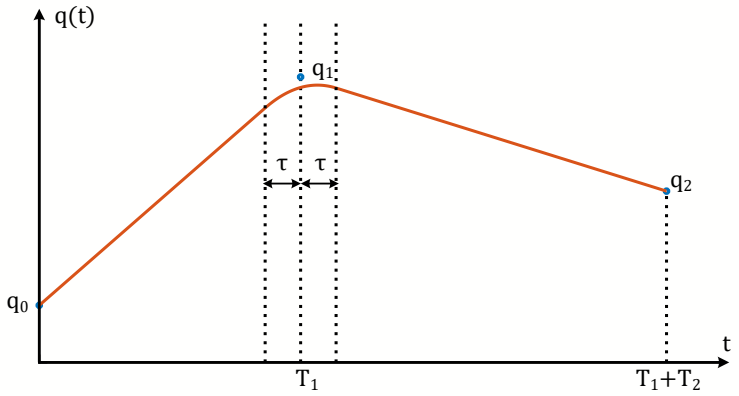

Figura 5.5 Interpolador con ajuste parabólico.

En las expresiones anteriores de la función a trozos analizada, el parámetro τ se corresponde con la mitad del tiempo durante el cual se debe extender la fase de aceleración (el tramo parabólico) para poder enlazar los dos tramos rectilíneos de acuerdo con la limitación impuesta por la aceleración a. El valor de a se corresponde con la aceleración constante que se impondrá en los tramos curvos de la trayectoria, es decir, en los que está cambiando la velocidad de la coordenada articular. La relación entre τ y a viene dada por la siguiente expresión:

$$\tau = \frac{T_1(q_2 - q_1) - T_2(q_1 - q_0)}{2T_1T_2a}$$

Para extrapolar esto al caso de una cantidad indefinida de puntos, es necesario generalizar el cálculo de los parámetros del movimiento (a, T_1, T_2, τ) que posteriormente permiten reproducir la curva de $q_i^{ref}(t)$.

La idea aquí adoptada consiste en, dados N puntos (t_j, $q_{i,j}$) a enlazar, tomar secuencias de puntos de tres en tres y calcular, para la sub-trayectoria que los uniría, los valores de los dichos parámetros, y almacenarlos para poder posteriormente hacer uso de ellos durante el control del movimiento.
Para los substramos de aceleración ha de tenerse en cuenta, además, que la aceleración a alcanzable por los motores está dada en valor absoluto. Para determinar su signo en cada tramo parabólico, basta con analizar las pendientes de los tramos rectilíneos anterior y posterior al tramo parabólico en cuestión, y comprobar si la velocidad de la coordenada articular debe aumentar ($a > 0$) o disminuir ($a < 0$). La a con signo (y no la dada en valor absoluto) debe ser la usada en el cálculo del tercer coeficiente que aparece en la expresión de $q(t)$ que define al tramo parabólico.

De la trayectoria resultante asociada a cada trío de puntos consecutivos, se tomarán únicamente los parámetros de movimiento asociados a: el primer tramo rectilíneo, y el tramo parabólico de paso por el punto intermedio; teniéndose en cuenta que ese primer tramo rectilíneo deberá ser tomado solo a partir del instante en que finaliza el tramo parabólico de paso por el punto anterior.

Es evidente que, con este procedimiento, ninguno de los puntos intermedios de la secuencia de puntos (t_j, $q_{i,j}$) es realmente alcanzado, sino que se realiza un paso suavizado por las proximidades de todos ellos. Sí serán alcanzados, lógicamente, el punto inicial y final, para lo que es necesario tomar el primer tramo rectilíneo completo de la primera sub-trayectoria, así como el segundo tramo rectilíneo de la última sub-trayectoria.

Arranque y frenada

Las ideas anteriores se aprovecharán para lograr que se realicen de forma suavizada, no solo el paso por todos los puntos que se desea alcanzar, sino también el arranque inicial y la frenada final que se producen en la salida del punto de partida del movimiento y la llegada al último punto de destino. Para ello, se hace necesario que la curva de cada $q_i(t)$ tenga los tramos rectilíneos inicial y final horizontales, es decir, que el movimiento comience y finalice con velocidades nulas.

De garantizar esto se encargará la función ***prepara_parabolico*** que, antes de que se proceda al cálculo de los parámetros (a, T_1, T_2, τ) para cada tramo, introducirá a la secuencia de puntos de paso dada por el usuario

para cada coordenada articular tres puntos adicionales:

- El valor de la coordenada articular en el punto actual en que se encuentra el efector final del robot, repetido dos veces, introducidas al principio de la secuencia.
- El valor de la coordenada articular en el punto final repetido una vez, adicional a la ya dada por el usuario, al final de la secuencia.

En la Figura 5.6 se muestra el que sería el resultado de aplicar este procedimiento para una situación en la que el usuario habría proporcionado dos puntos por los que pasar (representados en azul), con las correspondientes velocidades máximas v_1 y v_2 con las que se desearía que se realizase el movimiento hasta alcanzarlos; y en la que el control habría introducido los tres puntos adicionales representados en rojo para permitir que el arranque inicial y la frenada de finalización del movimiento respetasen la misma aceleración que se aplica en los tramos parabólicos de los puntos de paso.

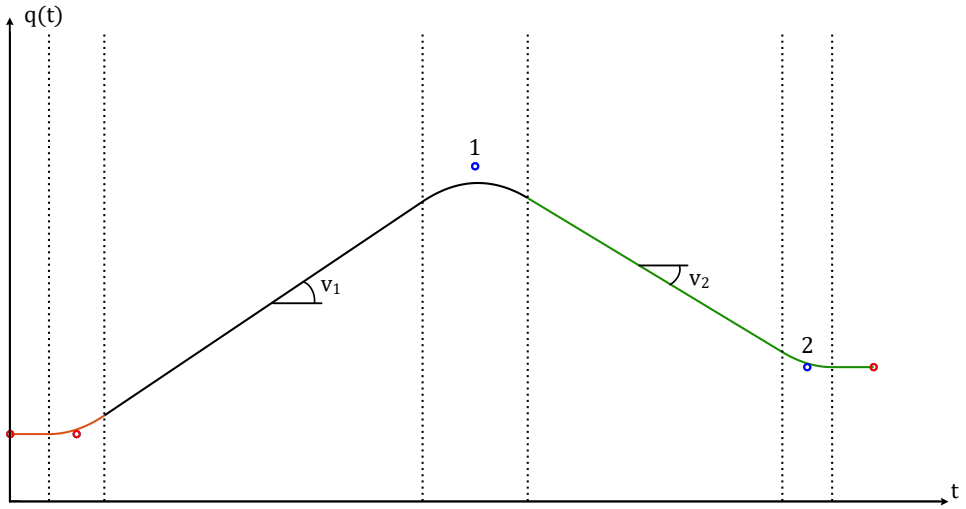

Figura 5.6 Trayectoria parabólica con varios puntos de paso, frenada y arranque.

Estrategia de obtención de los parámetros del movimiento

De acuerdo con las expresiones presentadas en la Subsección 5.3.5, la obtención de los parámetros que permiten definir la función a trozos que describe cada tramo del movimiento requiere de conocer los instantes t_j en los que las posiciones $q_{i,j}$ deben ser alcanzadas, ya que de estos instantes t_j se extraen directamente los valores de los T_1 y T_2. Sin embargo, en el caso de la programación de las tareas a ejecutar por un robot, el control no conocerá esos datos, sino los valores de $q_{i,j}$ acompañados de las velocidades máximas v_j con las que deben ser alcanzados, como se muestra en la Figura 5.6.

Se hace, entonces, necesario establecer una estrategia que permita obtener los valores de T_1, T_2 para cada tramo a partir de la información disponible, y poder entonces calcular el valor de τ con la expresión presentada anteriormente. A esta estrategia se le debe imponer el requisito de que esos valores de T_1 y T_2 sean comunes, tramo a tramo, para las tres coordenadas articulares q_i del robot, ya que es así como se garantiza lograr la sincronía en los movimientos de los motores.

Para realizar el cálculo de estos parámetros, partimos de la nomenclatura seguida en la Figura 5.5, y realizamos las siguientes definiciones adicionales por comodidad: dadas tres posiciones $q_{i,0}$, $q_{i,1}$, $q_{i,2}$ de una coordenada articulara a alcanzar de forma consecutiva, tomamos:

$$\Delta q_{i,10} = q_{i,1} - q_{i,0}$$
$$\Delta q_{i,21} = q_{i,2} - q_{i,1}$$

En base a lo anterior, el cálculo de los T_1, T_2 comunes a los tres motores para el caso del tramo de trayectoria que comprende a los dos puntos dados por el usuario constará de los pasos:

1. Obtención de Δq_{10}^{max}: búsqueda del motor que realiza el mayor movimiento en el tramo 10.
2. Cálculo de T_1: dado que, si existiese una sola coordenada articular, el valor de T_1 cumpliría corresponderse con: $T_1 = \frac{\Delta q_{10}}{v_1}$. Entonces, al tener trabajar con varias coordenadas articulares, el valor de T_1 deberá satisfacer que la coordenada articular que realiza el mayor movimiento sea capaz de realizarlo en el transcurso de ese tiempo T_1 haciendo uso de la velocidad máxima v_1 permitida para ese tramo de movimiento, por lo que se realizará el cálculo:

$$T_1 = \frac{|\Delta q_{10}^{max}|}{v_1}$$

3. Obtención de Δq_{21}^{max}.
4. Cálculo de T_2: análogamente:

$$T_2 = \frac{|\Delta q_{21}^{max}|}{v_2}$$

El procedimiento anterior se realiza punto a punto para la secuencia de posiciones por las que debe pasar el robot, y en cada ocasión, una vez conocidos los valores de T_1 y T_2, se realizan el resto de cálculo para los tramos rectilíneo y parabólico que suceden al punto en cuestión. Para ello, se establece que todos los motores aprovechen la aceleración máxima permitida en su tramo parabólico, de manera que serán los valores de la velocidad del tramo rectilíneo y de τ los que se adapten a cumplir con esos T_1 y T_2, de forma que solo el motor que realiza el mayor desplazamiento sea el que aproveche la velocidad máxima permitida para el movimiento. Por tanto, los pasos que siguen a la obtención de T_1 y T_2 son:

- Para cada motor (es decir, para cada coordenada articular q_i):
  1. Calcular la velocidad del tramo rectilíneo como:

$$v_{i,1} = \frac{\Delta q_{i,10}}{T_1}$$

  2. Determinando el signo de la aceleración del tramo parabólico:
     - Si $\Delta q_{i,21} > \Delta q_{i,10} \rightarrow$ aceleración positiva.
     - Si $\Delta q_{i,21} < \Delta q_{i,10} \rightarrow$ aceleración negativa.
     - Si $\Delta q_{i,21} = \Delta q_{i,10} \rightarrow$ aceleración nula.
  3. Calcular el valor del τ para el tramo parabólico:

$$\tau_i = \frac{T_1(q_{i,2} - q_{i,1}) - T_2(q_{i,1} - q_{i,0})}{2T_1T_2a}$$

Todos estos parámetros son almacenados para las tres coordenadas q_i para cada punto j a alcanzar, para poder ser utilizados posteriormente por la función ***run_motor*** durante el movimiento y permitirle calcular los valores de $q_i^{ref}(t)$ con la menor cantidad de cálculos computacionales posible.

5.4 Funcionamiento completo del control

El funcionamiento global de todas las ideas presentadas en el apartado anterior para llevar a cabo el control de los motores queda esquematizado en la Figura 5.7, donde queda de manifiesto que:

- Se hace uso de curvas de posición parabólicas, asociadas a perfiles de velocidades trapezoidales, para la generación de referencias que soliciten a los motores aceleraciones controladas.

- Antes de comenzar el movimiento, con el fin de reducir la carga computacional durante la ejecución del mismo, es necesario realizar el cálculo de los parámetros que permitirán evaluar el valor de la referencia q_i^{ref} de cada motor en cada instante.
- En cada ciclo de control, de forma previa a la ejecución de los cálculos propios del controlador, es necesario evaluar el valor de dicha referencia.
- Para que el bucle de control sea posible, es necesario: por un lado, realizar la conversión de la velocidad de giro a adoptar por el motor al tiempo de espera dT entre el pulso actual y el siguiente; y por otro lado llevar un conteo de los pasos dados por cada motor que sirva de realimentación del bucle.

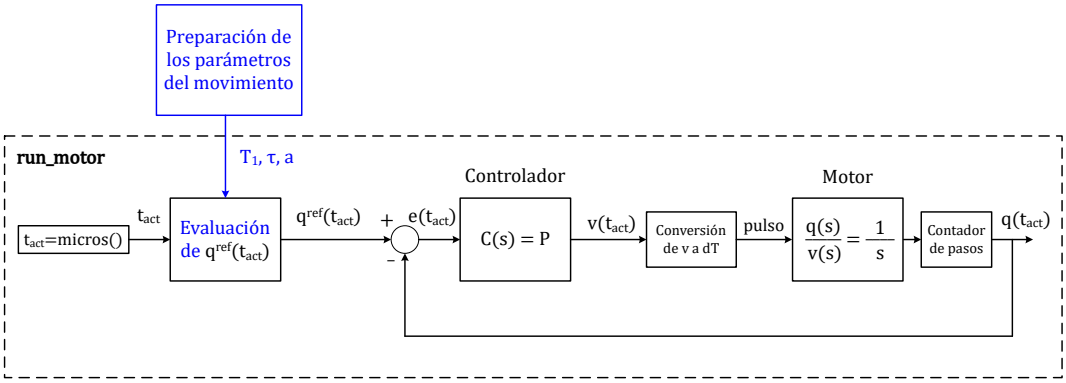

Figura 5.7 Control completo de los motores paso a paso.

De esta forma, queda combinada la generación de trayectorias que respetan un perfil de velocidades trapezoidal, con la utilización de un control en bucle cerrado que logra que los motores realicen el seguimiento de esas trayectorias.

Resultado del seguimiento de referencia

A continuación, se muestra el resultado de la traducción a código del diagrama de bloques anterior, particularizado para un caso concreto en el que se reciben por parte del usuario dos posiciones de destino $[q_{1,j}, q_{2,j}, q_{3,j}]$ (con $j = 1,2$), con las velocidades máximas v_j que se deben respetar en el movimiento hacia cada una de ellas. Tomando uno cualquiera de los motores, por ejemplo el Motor 1, el resultado de su control para un caso concreto similar al de la Figura 5.6 antes mostrada daría una evolución como la recogida en la Figura 5.8.

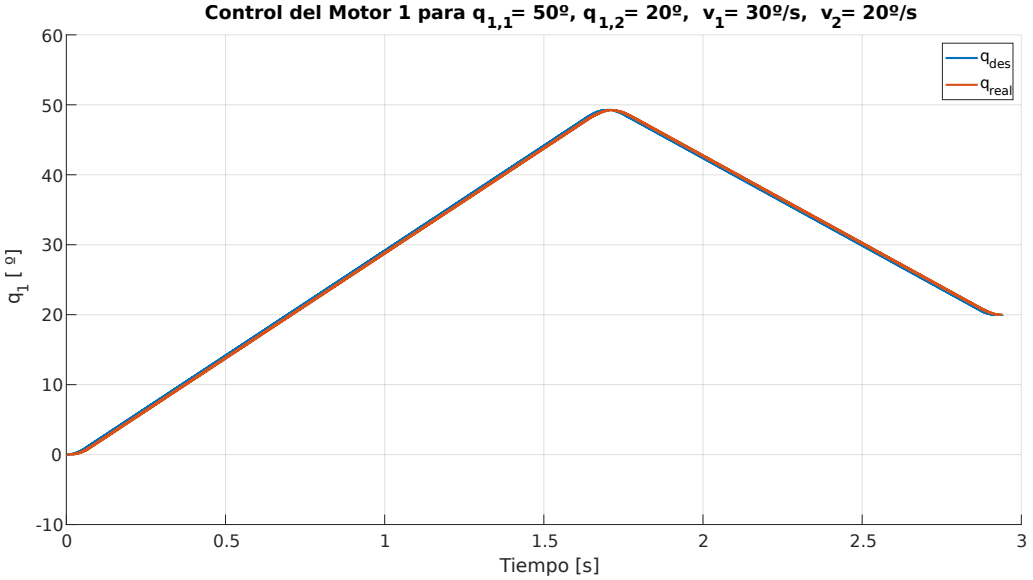

Figura 5.8 Trayectoria resultante del control completo de un motor.

Es conveniente señalar que esta gráfica ha sido obtenida artificialmente con *Matlab*®, utilizando el mismo código cargado en el *ATMega* para el control de los motores (pero traducido a lenguaje *Matlab*®), y generando el transcurso del tiempo. La razón de este hecho reside en que se ha comprobado que el microcontrolador no es capaz de enviar la información del movimiento al PC y respetar los dT que requieren los motores simultáneamente, sin que ésto perturbe la calidad del movimiento. Sin embargo, la gráfica generada resulta válida y representativa, ya que a que se ha impuesto que, para su generación, el bucle de control de los motores sea llamado cada $300\mu s$, lo cual implica que la actualización de la señal de control (del dT) nunca se llevará a cabo en el instante exacto en que se debería producir el nuevo pulso del motor, implicando la aparición temporal de errores de posición que no se acumulan gracias al papel del controlador proporcional.

5.5 Funciones de usuario MOVEx

En los apartados anteriores se han explicado todos los procedimientos necesarios para poder lograr el control sincronizado de los tres motores del robot en el paso por una cantidad N de puntos dados en coordenadas articulares $[q_1, q_2, q_3]$, respetando unas velocidades máximas proporcionadas por el usuario y logrando la corrección automática de la pérdida de pasos durante la ejecución del movimiento.

En este apartado, se explicará el funcionamiento interno de las funciones MOVEx preparadas para ser usadas por el usuario en la programación de tareas para el robot. Estas funciones se pueden clasificar según estén asociadas a la ejecución de movimientos punto a punto o movimientos continuos que comprenden una secuencia de puntos, así como según trabajen con puntos dados en coordenadas articulares o en coordenadas cartesianas. En todos los casos, son funciones que hacen uso de herramientas comunes que se encuentran programadas en funciones internas del control del robot, y la simplificación de cómo se relaciona cada función MOVEx con esas funciones internas se encuentra recogida en la Figura 5.4 ya presentada. En lo que sigue se describen por separado:

5.5.1 *MOVEQp*

Función que recibe un punto de destino dado en coordenadas articulares $[q_1, q_2, q_3]$ en unidades de grados, así como la velocidad máxima que debe presentar cualquiera de las articulaciones en ese movimiento en unidades de grados por segundo; y ejecuta un movimiento punto a punto desde la posición actual hasta esa posición de destino, sin imponer mayores requisitos al movimiento que el hecho de que los motores tengan un comportamiento sincronizado durante todo el trayecto, de manera que el robot realiza la trayectoria que le resulta más "cómoda" y directa.

De acuerdo con las ideas expuestas en este capítulo, la trayectoria a seguir por cada coordenada articular en este movimiento constará realmente de cuatro puntos de paso: primero el punto actual repetido dos veces, para lograr un arranque con aceleración controlada, y luego el punto de destino repetido también dos veces, para garantizar que la frenada al alcanzarlo también sea suavizada.

La sintáxis de uso de esta función es la siguiente:

Código 5.1 Sintáxis de la función *MOVEQp*.

```
MOVEQp(q_d1, q_d2, q_d3, v_max);
```

Un ejemplo de uso sería:

Código 5.2 Ejemplo de uso de la función *MOVEQp*.

```
MOVEQp(0.0, -90.0, 180.0, 30.0);
```

5.5.2 *MOVEQ*

Función que recibe un punto de destino, dado en coordenadas articulares, que se desea que forme parte de una trayectoria multipunto que realice de manera suavizada el paso por el conjunto de posiciones a alcanzar, idea a la que en adelante se hará referencia como "movimiento acumulativo".

Se trata, por tanto, de una función que no ejecuta ningún movimiento cuando es llamada, sino que se encarga únicamente de almacenar la información recibida en un vector de puntos que será posteriormente procesado por otras funciones. La sintáxis de la función es:

Código 5.3 Sintáxis de la función *MOVEQ*.

```
MOVEQ(q_d1, q_d2, q_d3, v_max);
```

Ejecución de los movimientos acumulativos

Para permitir el adecuado funcionamiento de las funciones asociadas a "movimientos acumulativos", es decir, movimientos pasando por una secuencia de puntos proporcionada por el usuario, como son las funciones ***MOVEQ*** y ***MOVEJ***, se ha seguido la siguiente estrategia:

- Cada función de movimiento acumulativo, cuando es llamada, almacena el punto recibido en un vector, y activa un *flag* para indicar al resto de funciones que existen puntos acumulados pendientes de recorrer.
- Todas las funciones, al ser llamadas, comprueban si existen puntos acumulados pendientes, y en caso de que existan, antes de realizar sus propias tareas, llaman a la función interna ***ejecuta_moveq*** que se encargará de ejecutar el movimiento acumulativo completo.
- Si la función llamada es una función de movimiento acumulativo, y los puntos que existían acumulados habían sido almacenados por otras llamadas a ella misma, no se ejecuta movimiento, sino que se continúa almacenando puntos hasta que en el código del usuario se llame a otra función distinta a ésta, para entonces sí realizar el movimiento acumulativo completo del tipo en cuestión.

5.5.3 *MOVEJp*

Función que recibe un punto de destino dado en coordenadas cartesianas $[x, y, z]$ en unidades de metros, así como la velocidad máxima que debe presentar cualquiera de las articulaciones en el trayecto; y ejecuta un movimiento punto a punto desde el punto actual hasta ese punto de destino, siguiendo la trayectoria más "cómoda" y directa para el robot.

Se trata, por tanto, de el equivalente a la función ***MOVEQp*** pero que trabaja, esta vez, con coordenadas cartesianas. Su código consta, consecuentemente, de un único paso adicional que consiste en la conversión del punto dado por el usuario a coordenadas articulares, haciendo uso del Modelo Cinemático Inverso del robot codificado en la función ***CI***.

La sintáxis de uso de esta función es la siguiente:

Código 5.4 Sintáxis de la función *MOVEJp*.

```
MOVEJp(x_d, y_d, z_d, v_max);
```

En la llamada a esta función, el usuario también puede hacer uso de los parámetros dimensionales que definen la geometría de la estructura del robot. Un ejemplo de esto sería:

Código 5.5 Ejemplo de uso de la función *MOVEJp*.

```
MOVEJp(0.1, -0.05, L1, 30.0);
```

5.5.4 *MOVEJ*

Función que recibe un punto de destino, dado en coordenadas cartesianas, que se desea que forme parte de una trayectoria multipunto que realice de forma suavizada el paso por el conjunto de puntos a alcanzar. Se trata, por tanto, de una función de movimiento acumulativo, al igual que ***MOVEQ***, y que difiere de ésta en las unidades en las que se encuentra el punto recibido (coordenadas cartesianas en lugar de coordenadas articulares). En su código, se encarga de almacenar dicho punto tras haberlo convertido a coordenadas articulares usando la función ***CI***.

La sintáxis de uso de esta función es la siguiente:

Código 5.6 Sintáxis de la función *MOVEJ*.

```
MOVEJ(x_d, y_d, z_d, v_max);
```

5.5.5 *MOVELp*

La función ***MOVELp*** es la última y más completa de las funciones de usuario que permiten programar el movimiento del robot, y recoge todas las ideas utilizadas por las funciones MOVEx previas, ya que recibe una única posición de destino, pero construye su propio movimiento acumulativo para alcanzarla.

Se trata de una función que recibe un punto de destino a alcanzar, en coordenadas cartesianas, al que se desea que el robot se desplace desde su posición actual siguiendo una trayectoria rectilínea en el espacio durante su movimiento. Hace, por tanto, necesario introducir un proceso de interpolación, de manera que su codificación consta de los siguientes pasos:

- Cálculo de la distancia a recorrer, y obtención del *paso* de interpolación en función de un número máximo de puntos almacenables en memoria.
- Interpolación de la trayectoria rectilínea, calculando puntos de ésta equiespaciados por el *paso* de interpolación calculado, y obteniendo las coordenadas articulares asociadas a cada uno de ellos con la función ***CI***, almacenándolos como si de una función de movimiento acumulativo se tratase. Serán puntos que se encuentren a distancias muy pequeñas en el espacio, logrando así que la línea recta deseada se pueda reproducir con suficiente fidelidad.
- Ejecución del movimiento pasando por la secuencia de puntos almacenados siguiendo el mismo funcionamiento que presentan ***MOVEQ*** y ***MOVEJ***, pero donde en este caso los puntos intermedios no han sido proporcionados por el usuario, sino generados por el control.

La sintáxis de uso de la función ***MOVELp*** es la siguiente:

Código 5.7 Sintáxis de la función *MOVELp*.

```
MOVELp(x_d, y_d, z_d, v_max);
```

5.6 Funciones para el control de la pinza

Como se ha mostrado en el Capítulo 2, el robot construido ha sido dotado con una pinza como posible efector final que permite el agarre de objetos de pequeño peso, sirviéndose de un servomotor como actuador. La apertura de esta pinza es controlada haciendo uso de la librería *Servo.h*, habiéndose realizado pruebas experimentales para determinar los rangos de movimiento adecuados para el servomotor una vez montado en el mecanismo. Tras esto, se han programado dos funciones que permiten al usuario trabajar con la pinza cómodamente sin preocuparse de cuestiones más allá de proporcionar la apertura deseada.

5.6.1 *CLOSE*

La función ***CLOSE*** supone una función muy sencilla cuya única tarea es realizar un cierre rápido de la pinza, llevando al servomotor a la orientación que está asociada con esa posición. No recibe ningún argumento en su llamada, y su sintáxis es la siguiente:

Código 5.8 Sintáxis y ejemplo de uso de la función *CLOSE*.

```
CLOSE();
```

5.6.2 *OPEN*

La función ***OPEN*** es la que permiten verdaderamente el control adecuado de la apertura de la pinza. Dicha apertura está limitada a 90° en cada mandíbula de la pinza, por lo que esta función recibe como argumento el valor entero del ángulo de apertura que se desea que adopte la pinza, en unidades de grados y medido respecto a la línea que pasa por la posición de la pinza cerrada, como se muestra en la Figura 5.9.

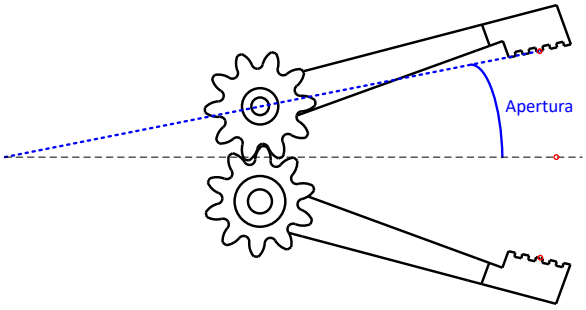

Figura 5.9 Ángulo de apertura de la pinza.

La función ***OPEN*** imprime sobre la pinza un movimiento suavizado, de manera que no llama a la función *servo.write* directamente con la orientación final deseada, sino que la apertura o cierre se realiza a una velocidad constante que se ha considerado como adecuada partiendo de la posición previa que presentase la pinza.

La sintaxis de uso de esta función es la siguiente:

Código 5.9 Sintáxis y ejemplo de uso de la función *OPEN*.

```
OPEN(20);
```

5.7 Funciones de usuario auxiliares

Adicionalmente a las funciones que permiten al usuario programar el movimiento del robot o la apertura y cierre de la pinza, existen otras dos funciones preparadas para ser usadas por éste que implementan comportamientos de gran utilizad:

5.7.1 *HOME*

La función ***HOME*** puede ser usada por el usuario pero también es utilizada por el control interno del robot y, en definitiva, permite resetear los valores internos de las coordenadas articulares conocidas por el robot (medidas en [pasos]), llevando al brazo articulado a una posición conocida sirviéndose de los Finales de

Carrera que éste tiene.

La función ***HOME*** presenta una codificación similar a la de las funciones que permiten ejecutar los movimientos normales del robot, pero con la diferencia de que, en ocasiones, esta función no necesita ejecutar un movimiento hasta que dicho movimiento se complete, sino hasta que se active un Final de Carrera. La secuencia de tareas que realiza es la siguiente:

- Para cada motor:
  1. Movimiento rápido en el sentido de aproximación al Final de Carrera hasta tocarlo.
  2. Alejamiento del Final de Carrera 10 grados a velocidad baja.
  3. Movimiento lento de aproximación al Final de Carrera hasta volver a tocarlo, logrando un ajuste fino de su posición.
- Tras completar los pasos anteriores:
  - Movimiento lento hasta la posición de reposo del robot.

La sintáxis de uso de ***HOME*** es la mostrada a continuación:

Código 5.10 Sintáxis y ejemplo de uso de la función *HOME*.

```
HOME();
```

5.7.2 *PAUSA*

La última de las funciones accesible al usuario es la función ***PAUSA***, que permite ser intercalada entre cualesquiera de las funciones anteriores y realizar una espera durante un período de tiempo determinado que recibe como argumento, medido en segundos.

Esta función, al igual que ocurría con la función ***run_motor***, no debe bloquear al microcontrolador, por lo que se servirá de la función *micros()* para ir comprobando si se ha cumplido el período de espera mientras permite la detección simultánea de la pulsación de los botones asociados a los cambios entre modos de funcionamiento. Su sintáxis es la siguiente:

Código 5.11 Sintáxis y ejemplo de uso de la función *PAUSA*.

```
PAUSA(1.5); // 1.5 segundos
```

5.8 Control Manual

El robot desarrollado en este proyecto, además de permitir la programación de tareas mediante código, presenta un Modo Manual que permite al usuario controlar su movimiento a partir del accionamiento de unos joysticks, así como conocer información sobre la situación de éste mediante la pulsación de botones.

5.8.1 Uso de los controles en el Modo Manual

En la Figura 5.10 se resumen las funciones de los controles del robot cuando se está haciendo uso del Modo Manual. Como se puede observar, el *Escudo* de control del robot cuenta con dos joysticks y dos botones, aunque dentro de este modo no se hará uso de uno de esos botones.
Los controles que se contemplan son:

- **Botón de la derecha:** conmutación entre el Modo Manual en coordenadas articulares y el Modo Manual en coordenadas cartesianas.

- **Dentro del modo Manual articular:**
  - **Joystick 1 a derecha e izquierda:** movimiento de la coordenada q_1.
  - **Joystick 1 arriba y abajo:** movimiento de la coordenada q_2.
  - **Joystick 2 arriba y abajo:** movimiento de la coordenada q_3.
- **Dentro del modo Manual cartesiano:**
  - **Joystick 1 a derecha e izquierda:** movimiento de la coordenada y.
  - **Joystick 1 arriba y abajo:** movimiento de la coordenada x.
  - **Joystick 2 arriba y abajo:** movimiento de la coordenada z.
- **Joystick 2 a derecha e izquierda (en cualquiera de los modos):** apertura y cierre de la pinza.
- **Pulsador del Joystick 1:** mostrar por el monitor serie la información sobre la posición actual del efector final del robot, en coordenadas articulares $[q_1, q_2, q_3]$ y en coordenadas cartesianas $[x,y,z]$.
- **Pulsador del Joystick 2:** mostrar por el monitor serie la información sobre el ángulo de apertura actual de la pinza.

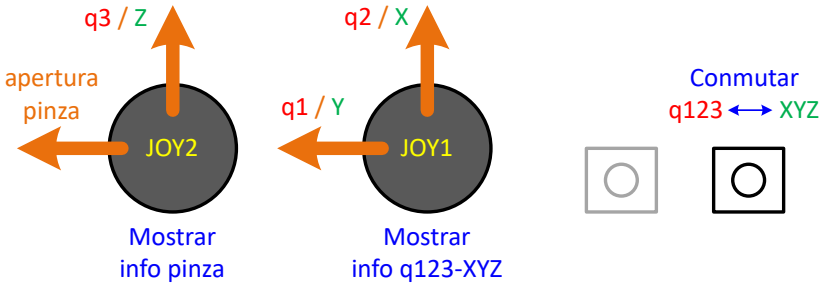

Figura 5.10 Controles durante el Modo Manual.

Este Modo Manual ha sido desarrollado con el objetivo de servir de ayuda al hipotético usuario encargado de programar tareas para el robot, que le permitiría llevar al brazo manualmente a distintas posiciones del Espacio de Trabajo teniendo realimentación visual de dónde se encuentra en cada momento, para posteriormente poder conocer los datos de estas posiciones y así usarlos como argumentos de las funciones MOVEx de las que haga uso en su código. Análogamente, la apertura y cierre manuales de la pinza permiten comprobar cual es la posición adecuada de ésta en el momento de coger un objeto concreto que se deba manipular.

A continuación, se explican las ideas en las que se basa funcionamiento de cada uno de los dos submodos dentro del Modo Manual:

5.8.2 Modo Manual en coordenadas articulares

El funcionamiento del Modo Manual en coordenadas articulares, o "Modo Manual q123", se basa en comprobar la posición de la palanca de cada uno de los joysticks de manera que, mientras que el joystick de una determinada coordenada articular se mantenga desplazado en un sentido, dicha coordenada articular se incremente o reduzca a razón de una velocidad que se establece como proporcional al grado de desplazamiento de dicha palanca. La velocidad en cuestión estará acotada entre cero y una velocidad máxima, que se ha establecido en 20 °/s, de manera que resulta siempre lo suficientemente baja como para que no sea necesario introducir procesos de aceleración ni desaceleración al comenzar y terminar los movimientos del Modo Manual, lo que hace posible, a su vez, que el robot comience a moverse y se pare inmediatamente en cuanto el usuario desplaza o suelta los joysticks, y que la carga de cálculos computacionales para el *ATMega* dentro de este submodo sea muy reducida, permitiendo que su funcionamiento sea muy fluido.

Como se puede intuir de la explicación anterior, en el Modo Manual articular se hace uso del valor de la señal analógica que proporcionan los potenciómetros de los joysticks para graduar la velocidad de movimiento

de las articulaciones. Ha de tenerse en cuenta que la realización de pruebas experimentales mostró que la calidad de los dispositivos era reducida, saturándose rápidamente el valor de la señal a su máximo o mínimo en cuanto se desplaza al joystick de su posición de reposo, y presentando esta señal cierto ruido cuando el joystick se encuentra quieto. Es por esto que se hace necesario establecer un valor umbral a partir del cual puede considerarse que el usuario está desplazando cada joystick en una u otra dirección, siendo, solo entonces, cuando se aplica la proporcionalidad mencionada para provocar el movimiento de los motores.

5.8.3 Modo Manual en coordenadas cartesianas

El funcionamiento del Modo Manual cartesiano, o "Modo Manual XYZ", se basa, al igual que el anterior, en la comprobación del desplazamiento de los joysticks para modificar consecuentemente los valores de las coordenadas $[x,y,z]$ de referencia, aplicándoles un incremento proporcional al ángulo que presente la palanca del joystick. Sin embargo, la carga computacional de este submodo es considerablemente mayor que la del anterior, ya que al tratarse de un control cartesiano, es necesario que el microcrontrolador realice paralelamente las tareas de:

- Lectura de los joysticks.
- Actualización de los valores de las coordenadas $[x,y,z]$ a alcanzar de acuerdo con esa pulsación de los joysticks.
- Comprobación de si la nueva posición $[x,y,z]$ queda dentro del Espacio de Trabajo del robot, haciendo uso de la función ***dentro_ET***. En caso de que la nueva posición de referencia quede fuera de los límites alcanzables por el robot, se deberá restablecer la referencia anterior.
- Cálculo de las coordenadas articulares $[q_1, q_2, q_3]$ asociadas a la posición $[x,y,z]$ de referencia, haciendo uso de la función ***CI***.
- Implementación del bucle control proporcional de los motores para lograr el movimiento de estos hacia la posición $[q_1, q_2, q_3]$ de referencia conforme ésta va variando, dando los pulsos correspondientes y calculando los valores de dT_i que se deben dejar transcurrir.

La alta carga de tareas que debe ejecutar el microcontrolador durante este Modo Manual XYZ, que requiere de preparar y controlar el movimiento simultáneamente, hace que se deba limitar la velocidad con la que pueden evolucionar las coordenadas $[x,y,z]$ a valores muy bajos, ya que de lo contrario surge el problema de que el tiempo que transcurre entre cada dos ciclos en los que se ejecuta el bucle de control de un motor comienza a ser mayor que el dT_i con el que éste requeriría funcionar. De darse esta situación, comenzaría a producirse acumulación del error de posición del robot, conllevando que cuando el usuario soltase los joysticks, el robot aún permanecería desplazándose durante un breve período de tiempo hasta alcanzar la posición que tiene almacenada como referencia.

6 Programación del brazo como robot industrial

En este capítulo se describen las herramientas que han hecho posible añadir una capa adicional en la jerarquía del control del brazo articulado para permitir al usuario trabajar con el sistema construido como si se tratara de un robot industrial convencional, pudiendo hacer uso de todas las funciones explicadas en el Capítulo 5 para la programación de tareas mediante código, y además conmutar entre distintos modos de funcionamiento haciendo uso del *Escudo* cuya fabricación se tratará en el Capítulo 7.

6.1 Máquina de Estados del control

El funcionamiento completo del robot construido está gestionado, en el nivel más alto de la jerarquía de control, por una Máquina de Estados que contempla los siguientes modos de funcionamiento:

- **DES:** el robot se encuentra con los motores desenergizados, no por acción del microcontrolador sino porque el usuario ha interrumpido físicamente la llegada de la señal de Eneable a los drivers. A este estado se puede acceder en cualquier momento mediante el uso de un interruptor dedicado, de manera que consistiría en una medida de seguridad del sistema que permite apagar los motores del robot en cualquier situación de emergencia.
- **OFF:** los motores del robot se mantienen apagados, pero en esta ocasión por estar deshabilitada la señal de Eneable que envía el microcontrolador a los drivers, existiendo una conexión física ininterrumpida entre ambas partes.
- **INICIO:** los motores del robot quedan energizados, y se mantienen bloqueados en la posición en la que se encontrasen, sin movimiento.
- **PREPARADO:** el robot, tras pasar por el procedimiento adecuado para ello, pasa a tener conocimiento de en qué posición se encuentran sus eslabones, quedando preparado para iniciar cualquier tipo de movimiento controlado.
- **CODIGO:** el robot ejecuta el código de la tarea que el usuario ha programado y cargado en el microcontrolador.
- **BLOQUEADO:** el robot queda inmediatamente bloqueado en la posición en la que se encontrase tras solicitarse que se aborte la tarea de usuario durante su ejecución; o al completarse ésta.
- **MANUAL:** ejecución del Modo Manual del brazo articulado, con las características que se han detallado en el Capítulo 5.

La Máquina de Estados en cuestión es, por tanto, capaz de trabajar de manera combinada con: el Modo Manual del robot, la ejecución de tareas programadas por el usuario, y la conmutación adecuada y segura entre los distintos modos de no actividad por los que se ha de pasar. Para ello, será necesario acudir con frecuencia a la realización de movimientos predefinidos, que dan lugar a estados adicionales de transición de los que se sale automáticamente cuando concluyen. Dichos estados de transición serán:

- **HOME:** proceso de ejecución de la función ***HOME*** para resetear las variables asociadas a las coordenadas articulares del robot. Su ejecución es necesaria al arranque del sistema, debido a que se desconoce en qué posición se encuentran los eslabones; y no volverá a ser necesaria mientras los motores no pasen por un estado desenergizado.
- **NO_HOME:** la estructura del robot presenta soportes para autosustentarse en su posición de reposo, de manera que es sencillo lograr que el robot se encuentre, antes de encenderlo, en aproximadamente la misma posición que alcanzaría tras ejecutarse ***HOME***. Es por eso que, fundamentalmente para agilizar la realización de pruebas con el sistema y ahorrar tiempo, existe una función ***NO_HOME*** que permite realizar el mismo reseteo de las variables q_i que realiza la función ***HOME*** pero omitiendo los movimientos asociados.
- **Movimiento a posición de reposo:** tras finalizar cualquier tarea de usuario, se ha establecido que sea necesario, por seguridad, que el robot vuelva a su posición de reposo cuando se le indique desde el *Escudo* antes de poder volver a ejecutar dicha tarea, de manera que los programas codificados por el usuario podrán asumir que el robot siempre parte de esa posición.

Los movimientos de los estados de transición pueden, además, ser interrumpidos en cualquier momento en mitad de su ejecución si el usuario solicita que sean abortados, pasando entonces el robot a quedar bloqueado.

Para indicar al exterior el estado en el que se encuentra el sistema en cada momento se hará uso de tres leds, de colores rojo, verde y azul. El funcionamiento completo de la Máquina de Estados queda recogido, a modo de grafo de estados, en la Figura 6.1, donde se muestran adicionalmente las acciones que debe realizar el usuario desde el *Escudo* para lograr la conmutación entre los estados, y los leds que quedan encendidos o parpadeando dentro de cada uno.

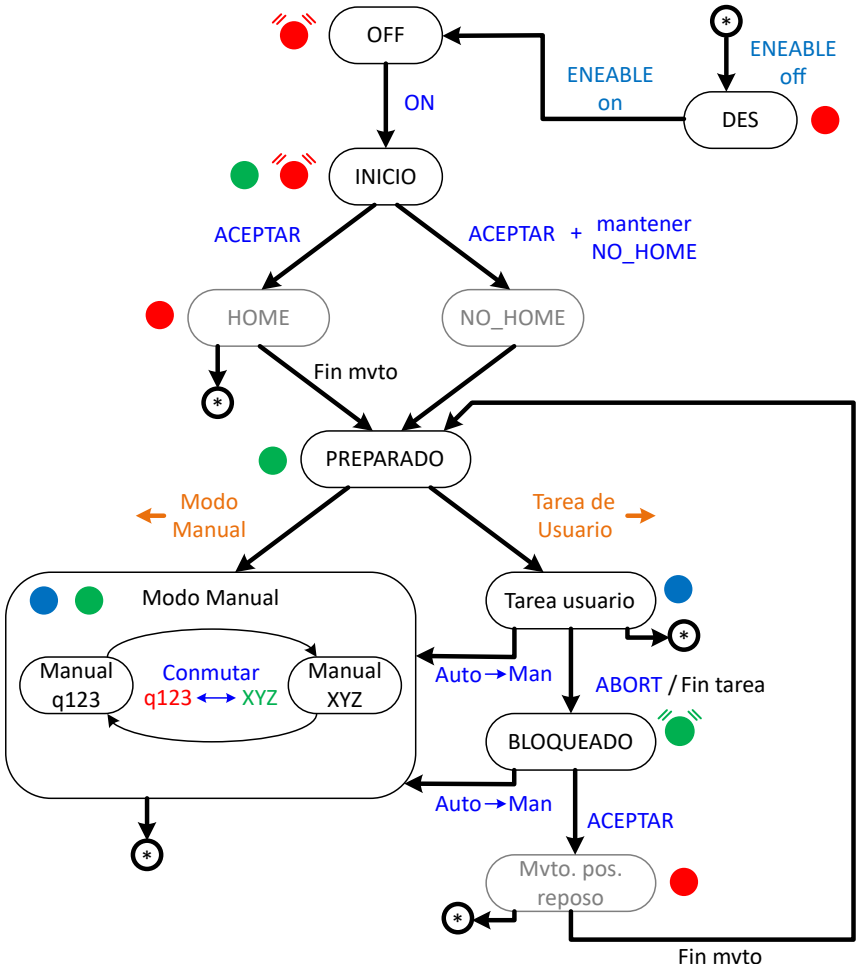

Figura 6.1 Máquina de Estados del control.

6.1.1 Instrumentos para gestionar el control

Como se puede comprobar en la Figura 6.1, para la gestión de todas las conmutaciones contempladas serían necesarias herramientas que permitan al usuario la interacción con el microcontrolador para indicar el siguiente listado acciones: poner la señal de Eneable que llega a los drivers a ON/OFF manualmente, encender el robot (botón de ON), ACEPTAR, ABORTAR cualquier movimiento, conmutar de Auto->Man, conmutar entre los modos manuales q123 y XYZ, arrancar la ejecución de la tarea del usuario, y arrancar la ejecución del Modo Manual desde el estado de PREPARADO.

A las acciones anteriores, se les suma la restricción de disponer de al menos dos joysticks y un pulsador para satisfacer las necesidades del Modo Manual. Tras estudiar la manera óptima de recoger todas estas funcionalidades haciendo uso de un número reducido de instrumentos y garantizando que no exista cabida para la confusión en cuanto a la utilidad que presentan en cada instante los mandos con funciones multiplexadas, se concluyó en disponer los siguientes componentes para la gestión del control del robot:

- **Dos joysticks:** cuyo uso ya ha sido explicado en el Capítulo 5, y a los que se le añade una funcionalidad adicional: cuando el sistema se encuentre en el estado PREPARADO, se indicará con el joystick derecho si se desea que se inicie la ejecución de la tarea de usuario, desplazándolo hacia la derecha; o que el robot entre en el Modo Manual, desplazándolo hacia la izquierda.
- **Dos interruptores:**
  - Eneable: interruptor que permite al usuario desconectar físicamente la llegada de la señal de Eneable que el microcontrolador envía a los drivers.
  - Alimentación del *Mega 2560 Pro*: interruptor que permite cortar la alimentación que, como se explicará en el Capítulo 7, llega de la fuente que alimenta a los motores al microcontrolador.
- **Tres leds:** de colores rojo, verde y azul, cuya utilidad ya se ha mostrado en la Figura 6.1.
- **Dos pulsadores:** el conjunto de funcinalidades que requieren de la pulsación de botones queda recogida en dos pulsadores de usos multiplexados:
  - MULTI1: destinado a:
    - ON
    - ACEPTAR
    - ABORT
    - Conmutación entre los modos manuales q123 y XYZ
  - MULTI2: destinado a:
    - Conmutar de la ejecución de la tarea del usuario al Modo Manual en cualquier momento
    - Poder omitir la ejecución de ***HOME*** ejecutando ***NO_HOME*** si es mantenido mientras se pulsa ACEPTAR

Señal de Eneable con realimentación

Como se viene mencionando a lo largo de este capítulo, el usuario tiene la posibilidad de desconectar físicamente la señal de Eneable que llega a los drivers que controlan a los motores, sin depender de la intervención del microcontrolador como intermediario en esta acción. Sin embargo, el control necesita, además, tener conocimiento de si la conexión entre su pin de salida destinado a dar esa señal de Eneable y la entrada para esta señal del *CNC Shied V3* está o no interrumpida, para que nunca pueda iniciar un movimiento estando los motores desenergizados.

Es aquí donde entra en juego el pin del microcontrolador destinado a recibir la realimentación de la señal de Eneable que envía el propio *ATMega*, cuya existencia ya se mencionó en el Capítulo 5. La estrategia seguida para poder cortar físicamente la llegada de esta señal a los drivers y además recibir realimentación de dicha información es la mostrada en el esquema de conexionado de la Figura 6.2.

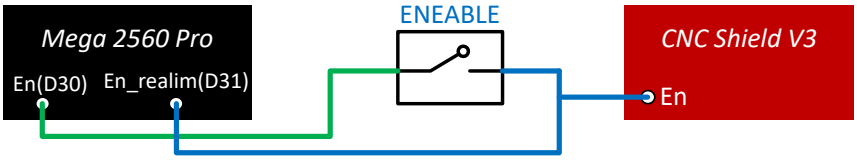

Figura 6.2 Realimentación de la señal de Eneable con paso por interruptor.

Requisitos para el *Escudo*

El conjunto de instrumentos anteriormente listado necesario para gestionar el control del robot deberá ser tenido en cuenta durante el diseño del *Escudo* que se llevará a cabo en el Capítulo 7, donde éstos se dispondrán según se muestra en la Figura 6.3.

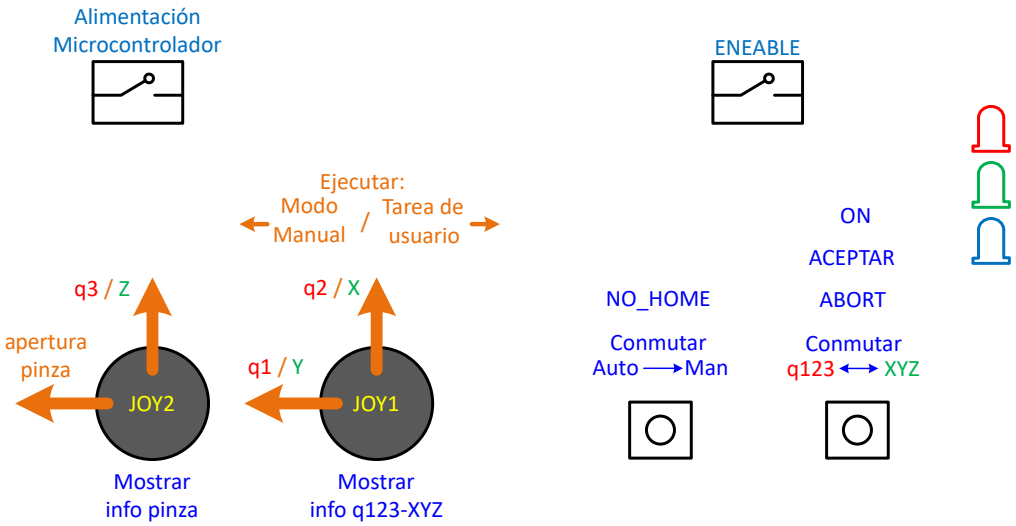

Figura 6.3 Disposición de los instrumentos en el *Escudo*.

6.2 Tareas de usuario

Una vez se establece la gestión del funcionamiento del robot mediante una Máquina de Estados, la tarea programada por el usuario pasa, desde el punto de vista de la programación, a tener que estar encapsulada en una función a la que esta máquina de estados pueda llamar cuando corresponda.

Dicha función es la denominada ***codigo_usuario***, y dentro de ella es donde se hará uso de las funciones ***MOVEx***, ***PAUSA*** y ***OPEN*** o ***CLOSE*** que se explicaron en el Capítulo 5. De esta forma, se logra que la programación de tareas para el robot sea muy limpia y tenga la misma sintaxis y apariencia que presentan los lenguajes de programación desarrollados por empresas dedicadas a la producción de robots industriales, como puede ser el caso del lenguaje *RAPID* de *ABB*.

Un ejemplo de lo que podría ser el código de programación de una tarea a cargar en el microcontrolador es el siguiente:

Código 6.1 Ejemplo de programación de una tarea para el robot.

```
#include "mra1.h"
void codigo_usuario(void)
{
  OPEN(30);
  PAUSA(1);
  MOVEQ(0.0, -45.0, 135.0, 30.0);
  MOVEJ(0.2, -0.06, 0.08, 25.0);
  MOVEJ(0.2, -0.12, 0.01, 10.0);
  OPEN(10);
  MOVEJ(0.2, 0.3, 0.01, 30.0);
  MOVELp(0.2, 0.0, L1, 30.0);
  ...
}
```

6.3 Librería del control

El paso final del desarrollo del código de control del sistema construido en este trabajo consiste en la creación de una librería que encierra la programación de todas las funciones mencionadas en este documento, a excepción de la función ***codigo_usuario***, que sería la única función presente en el archivo de extensión ".ino" con el que tuviese que trabajar el usuario programador de las tareas del robot, logrando así la máxima limpieza y simplicidad posible.

A esta librería se la ha denominado *mra1.h*, y un ejemplo de su uso es el mostrado en el Código 6.1 anterior. Su existencia hace posible que las líneas presentes en dicho código sean las únicas que contendría el fichero a cargar en el microcontrolador, y con esto se cargaría en el *Mega 2560 Pro* no solo la tarea de usuario sino la Máquina de Estados completa explicada en este capítulo.

7 Diseño del Escudo

Para facilitar el control del robot de forma ordenada y compacta se ha llevado a cabo el diseño y la fabricación de un *Escudo* que permite recoger todo el interconexionado necesario entre el *Mega 2560 Pro* y el *CNC Shield V3*, así como hacer de soporte a los botones y otros elementos necesarios para gestionar los modos de funcionamiento del sistema. En este capítulo se exponen los requisitos impuestos a su diseño y las posibilidades que éste ofrece, y se muestran los resultados obtenidos.

7.1 Diseño del *Escudo*

7.1.1 Requisitos de diseño del *Escudo*

A continuación se recogen los requisitos que se han tenido presentes durante la planificación y realización del diseño del *Escudo*:

- **Diseño compacto:** el *Escudo* deberá llevar conectados por su parte inferior tanto al *Mega 2560 Pro* como al *CNC Shield V3*, y para que el diseño sea lo más compacto posible, la PCB presenta un tamaño igual al de la superficie conjunta del *Mega* y del *Shield* más el espacio adicional estrictamente necesario para hacer posible la soldadura de los pines para el conexionado de los cables de los elementos a controlar (motores, finales de carrera, etc).
- **Alimentación del microcontrolador sin necesidad de conexión USB**: aunque el robot está destinado a ser programado desde un ordenador con las tareas codificadas por el usuario, resulta también interesante que el sistema pueda funcionar de forma independiente una vez tiene un programa cargado, sin la necesidad de disponer de una conexión permanente a un ordenador. El *Mega 2560 Pro* incorpora un regulador de tensión que puede ser alimentado con una tensión de entrada V_{in} de hasta 12 V, de manera que se ha incorporado en el *Escudo* un conector que permitirá hacer llegar a ese pin de entrada del *Mega* la tensión que proporciona la misma fuente utilizada para alimentar a los drivers que controlan a los motores. Esto se hace, además, incorporando en el *Escudo* un interruptor para poder desconectar al *Mega* de la fuente de 12 V en caso de que fuese necesario; aunque conviene señalar que la subida de un nuevo programa en el microcontrolador desde la conexión USB se puede realizar sin necesidad de desconectar la alimentación que le llega al pin de V_{in}.
- **Funcionalidades ampliables:** la conexión del *Mega 2560 Pro* al *Escudo* es tal que dificulta en gran medida el acceso a pines de la placa que no se encuentren conectados al propio *Escudo*. Dado el atractivo de este sistema para servir como punto de partida para otros proyectos o ampliaciones de éste, con posibles expansiones de los grados de libertad del robot o incorporación de otras funcionalidades, se ha diseñado el *Escudo* de forma que no se limite a satisfacer únicamente las necesidades del robot construido, sino que permitiera trabajar con una mayor cantidad de elementos. De esta forma, se ha intentado aprovechar al máximo los pines del *Mega 2560 Pro* de los que no se hace uso en este trabajo, incorporando en el *Escudo* pines hembra adicionales que estarían destinados a controlar: motores paso a paso, servomotores (con señal PWM) y finales de carrera adicionales, así como un módulo de comunicación I2C.
- **Contenido de las serigrafías:** en cuanto a las serigrafías impresas sobre la PCB, se ha dejado en la cara superior solo la información que sería necesaria para el hipotético usuario que tuviese que

programar y trabajar con el robot una vez que el sistema se encuentra ya montado y terminado; y se ha utilizado la cara inferior, por la que se realizan los conexionados, para incluir todo tipo de etiquetas que permitan identificar la funcionalidad de las distintas conexiones y seguir el curso de las pistas.

- **Espesor de las pistas:** para la determinación del ancho adecuado de las pistas en base a la corriente que deban soportar se ha tomado como referencia la información disponible en [7] y en [5], prestando especial cuidado a la elección de los espesores de las líneas asociadas a la alimentación de los motores paso a paso y del servomotor.
- **Anclaje del *Escudo* a la mesa de trabajo:** por último, se han incluido en la PCB orificios destinados a la introducción de tornillos de la misma métrica utilizada para el montaje del robot, M3, con el objetivo de permitir su anclaje al tablero que hace de mesa de trabajo del sistema.

7.1.2 Sumario de conexiones utilizadas en el proyecto

Conexiones al *Mega 2560 Pro*

Los pines del *Mega 2560 Pro* necesarios para el funcionamiento del robot, y que por tanto se traducen en las conexiones mínimas necesarias entre esta tarjeta y el *Escudo*, son los listados a continuación:

- **Alimentación:** pin de entrada V_{in}, alimentado a 12V, y pines de salida VCC y GND para alimentar a botones, leds, finales de carrera y servo.
- **Control de los motores paso a paso:** tres pines digitales para el control de la *Dirección* de giro de los motores, y tres para proporcionar los *Pulsos*.
- **Control del servomotor:** un pin digital con salida PWM para el control del servomotor que acciona a la pinza.
- **Finales de carrera:** tres pines digitales para la lectura de las señales provenientes de los finales de carrera.
- **Botones:** cuatro pines digitales para la lectura de las señales provenientes de los dos pulsadores de los joysticks y los dos pulsadores adicionales para la gestión de los modos de funcionamiento.
- **Leds:** tres pines digitales para el encendido de los tres leds que hacen de indicadores del modo de funcionamiento en que se encuentra el robot.
- **Eneable:** dos pines digitales, uno para dar la señal de Enable y otro para servir de realimentación y comprobar si ésta está llegando a los drivers.
- **Señales analógicas de los joysticks:** cuatro pines analógicos para la recepción de las señales provenientes de los potenciometros de los dos joysticks.

Conexiones al *CNC Shield V3*

Los pines del *CNC Shield V3* necesarios para el funcionamiento del robot son los listados a continuación:

- **Alimentación de los drivers:** los *DRV8825*, aunque se encargan de gestionar la alimentación a 12 V de las bobinas de los motores paso a paso, funcionan alimentados a una tensión de 5 V que es proporcionada por el *Mega*, siendo necesarios dos pines, de VCC y GND, para la entrada de esta tensión.
- **Control de los motores paso a paso:** seis pines (dos por motor) para la recepción de las señales digitales de *Dirección* y *Pulsos* que permiten controlar a los tres motores del robot.
- **Eneable:** pin de entrada para recibir la señal de Enable que el *Shield* hace llegar a todos los drivers que se monten sobre él.

En la Figura 7.1 se muestran los pines concretos de los se que hace uso para materializar las conexiones indicadas en este apartado.

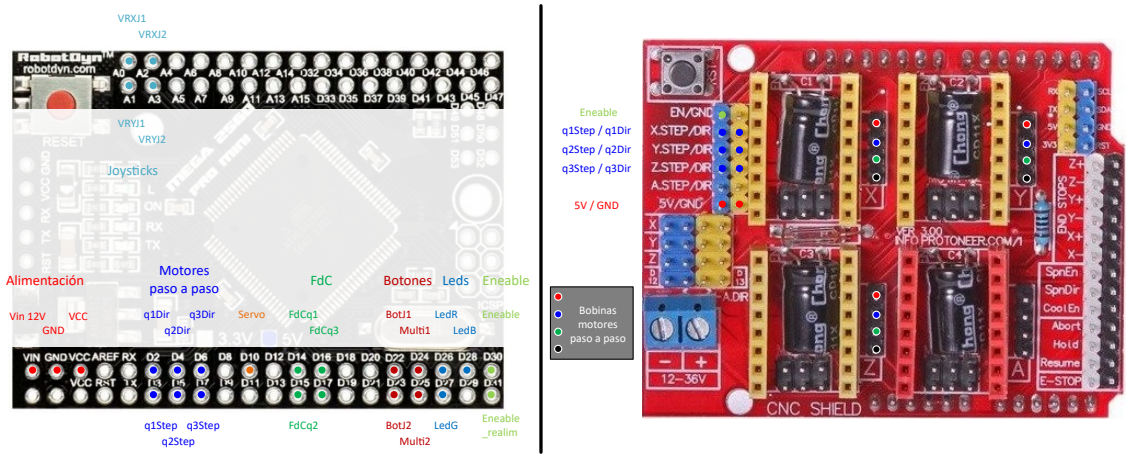

Figura 7.1 Conexiones utilizadas del *Mega 2560 Pro* y *CNC Shield V3*.

7.1.3 Conexiones disponibles en el *Escudo*

Como se ha indicado en la Subsección 7.1.1, el *Escudo* se ha diseñado de forma que no se limite a satisfacer únicamente las necesidades del robot construido, sino que permita trabajar con una mayor cantidad de elementos.

Tras un análisis de los pines utilizados para el control del sistema aquí estudiado y de los pines restantes que quedaban disponibles en el *Mega 2560 Pro*, se decidió aprovechar los pines en desuso, especialmente los pines digitales, según el reparto que se muestra en la Figura 7.2.

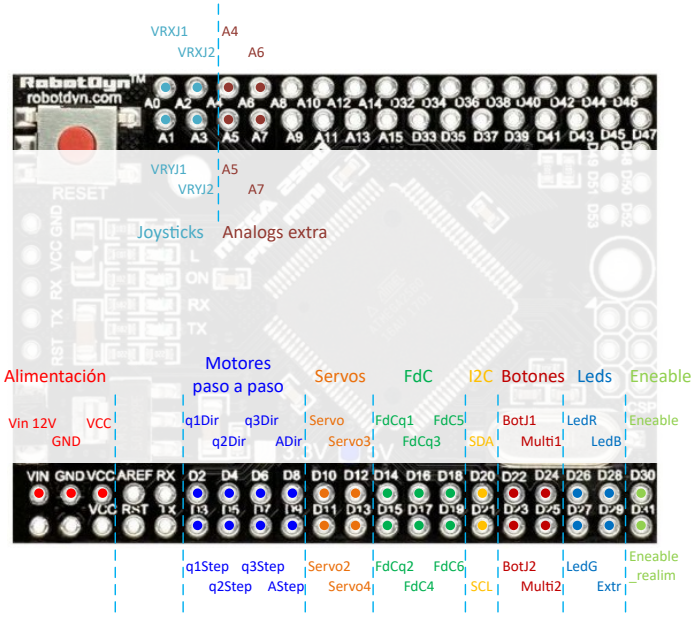

Figura 7.2 Conexiones aprovechadas del *Mega 2560 Pro*.

Con este reparto de la utilización de los pines, el *Escudo* diseñado podría servir para que el *Mega* trabajase con los siguientes elementos adicionales:

- **Un motor paso a paso:** dado que el *CNC Shield V3* está preparado para la conexión de un driver y un motor paso a paso adicionales a los tres utilizados en este robot, se utilizaron dichas conexiones asignándoles los pines necesarios del *Mega 2560 Pro*.

- **Tres servomotores:** los tres pines restantes del *ATMega* que cuentan con salida PWM estarían destinados a poder controlar tres servomotores adicionales.
- **Tres finales de carrera.**
- **Un módulo de comuniación I2C.**
- **Un botón o led extra:** dado que quedaba sin uso un pin restante en la zona de pines dedicada al trabajo con los pulsadores y los leds, se ha aprovechado el hecho de que estos dos componentes necesitan ambos una conexión a GND y otra a un pin del *ATMega*, preparando el *Escudo* de forma que podría admitir la conexión de uno cualquiera de estos dos elementos indistintamente.
- **Cuatro señales analógicas:** se ha dejado preparado el acceso a otros cuatro pines de entrada analógica del microcontrolador.
- **Señales de VCC y GND extras:** con el fin de facilitar la fase de ensayos durante la posible extensión de las funcionalidades de este sistema, se disponen en el *Escudo* pines destinados a proporcionar las tensiones de VCC y GND del *Mega* al exterior.

7.1.4 Elementos a soldar al *Escudo*

La preparación del *Escudo* para trabajar con los elementos citados no consiste simplemente en disponer sobre él las conexiones al *Mega 2560 Pro* y al *CNC Shield V3*, sino también los pines de salida que se requieren para trabajar con dichos elementos de forma práctica y sin necesitar cables adicionales ni protoboards. Las conexiones con la PCB que utiliza cada tipo de elemento son:

Elementos no definidos en librerías de componentes:

Los principales elementos no se encuentran definidos en librerías debido a que no van soldados a la PCB sino que requerirán de la disposición de pines para su conexión, y son:

- ***Mega 2560 Pro* y *CNC Shield V3***: todos los pines que presentan ambas tarjetas son tipo macho, por lo que requieren de un pin hembra en el *Escudo* por cada pin de éstas a utilizar.
- **Motores paso a paso:** cuatro pines macho, asociados a los cuatro cables que provienen de estos motores para energizar sus bobinas, que cuentan con un conector tipo hembra.
- **Servomotor:** un pin de señal PWM, y dos pines para la alimentación, VCC y GND; todos tipo macho.
- **Final de carrera:** dos pines macho, uno puesto a tensión de GND, y otro para la recepción de la señal digital cuando se cierra el contacto del final de carrera, el cuál deberá estar configurado en el *ATMega* con una resistencia de pull-up.
- **Joystick:** cinco pines hembra, que además deberán presentar un ángulo de 90° para facilitar la utilización de estos módulos.

Elementos definidos en librerías de componentes:

Para el resto de elementos utilizados, sus conexiones requeridas son generadas en la PCB automáticamente por el software al insertarlos en el diseño, y son:

- **Botón:** dos conexiones en la PCB (realmente cuatro, pero dos se dejan en desuso), una puesta a GND, y la otra conectada a un pin del *Mega* configurándose en el microcontrolador con una resistencia de pull-up.
- **Led:** dos conexiones en la PCB, una puesta a GND y la otra asociada a un pin del microcontrolador, desde el que se podrá encender aplicando un nivel alto de tensión. En el caso de los leds es, además, fundamental disponer una resistencia en serie con éstos que limite la corriente que los atraviesa.
- **Interruptor:** tres conexiones en la PCB asociadas a sus tres pines de entrada.
- **Conector de 12 V:** dos conexiones en la PCB.

7.1.5 Diseño del circuito con *EasyEDA*

El diseño del circuito que será impreso en la PCB (*Printed Circuit Board*) que da forma al *Escudo* ha sido realizado en el software en linea de *EasyEDA*, debido a que se trata de un software gratuito, de sencilla utilización, y que garantiza una correcta generación final de los ficheros necesarios para encargar la fabricación de la PCB.

El proceso de diseño ha constado de los siguientes pasos:

- Búsqueda en las librerías de los componentes disponibles.
- Diseño del conector de 12 V: el único componente no encontrado en las librerías de *EasyEDA* es el conector KF370 2P, con una separación entre pines de 7,5mm, para el que fue necesario realizar el diseño.
- Estudio de la posición más conveniente de los distintos componentes en base a: facilitar el conexionado de todos los cables al *Escudo* solo por un lateral de éste, y la colocación de los elementos de interacción con el usuario en posiciones ergonómicas.
- Determinación de las reglas de enrutado a respetar por las pistas.
- Utilización de la herramienta de auto-route que proporciona el propio software para generar los esquemáticos.

Los esquemáticos resultantes, con sus correspondientes serigrafías, son los mostrados en la Figura 7.3.

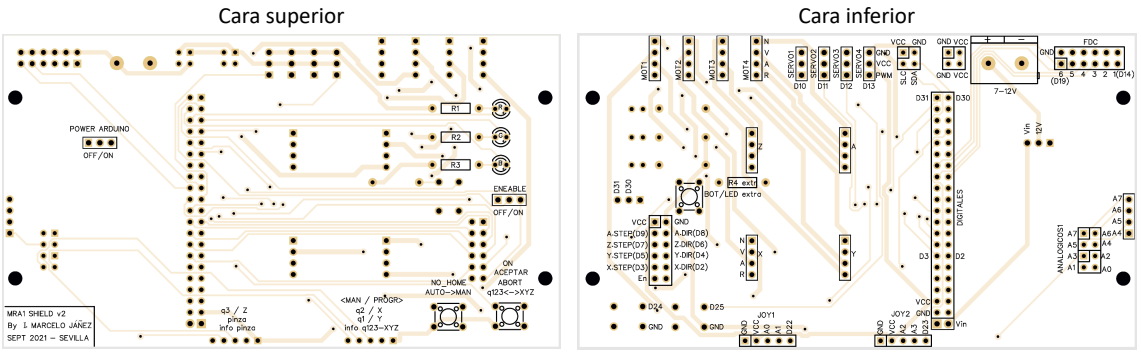

Figura 7.3 Esquemáticos del *Escudo*.

7.2 Fabricación del *Escudo*

La fabricación del *Escudo* cuenta de dos fases claramente diferenciadas: en primer lugar, la obtención de la PCB, y en segundo lugar, la ejecución del proceso de soldadura de los distintos componentes.

Para la fabricación de la PCB se ha hecho uso del servicio ofrecido por *JLCPCB.com*. Tras adquirirla, el conjunto de componentes a soldar para satisfacer todas las conexiones expuestas en los apartados anteriores son los mostrados en la Figura 7.4.

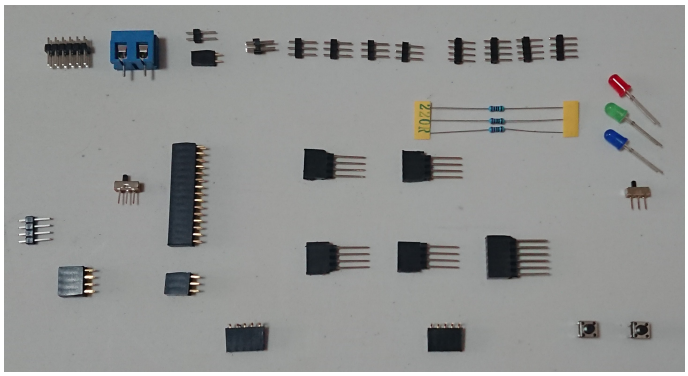

Figura 7.4 Componentes a soldar en la PCB.

7.2.1 Resultado obtenido

Tras concluir el proceso de soldadura, el *Escudo* finalizado es el mostrado en las siguientes imágenes. En la Figura 7.5 se muestran las dos caras de la PCB con los componentes; y en la Figura 7.6 se presenta una vista de cómo queda el acoplamiento del *Mega 2560 Pro* y del *CNC Shied V3* al *Escudo*.

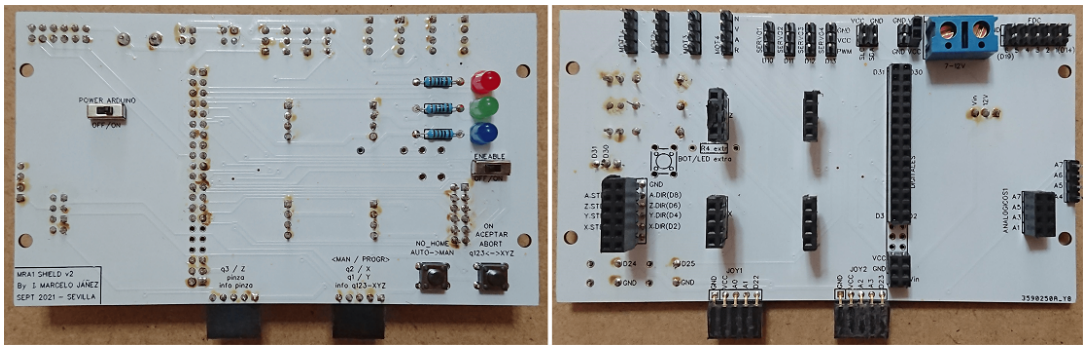

Figura 7.5 PCB con los componentes soldados.

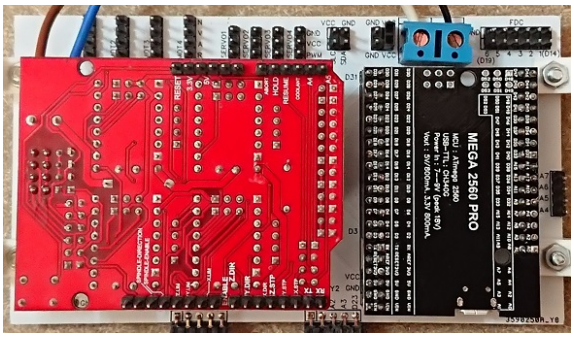

Figura 7.6 Acomplamiento del *Mega 2560 Pro* y del *CNC Shied V3* al *Escudo*.

Por último, en la Figura 7.7 se encuentra una comparativa del cableado necesario para el funcionamiento del robot durante la fase de prototipado del mismo, con el resultado final del *Escudo* tras conectarle todos los cables y los joysticks necesarios para controlar al sistema.

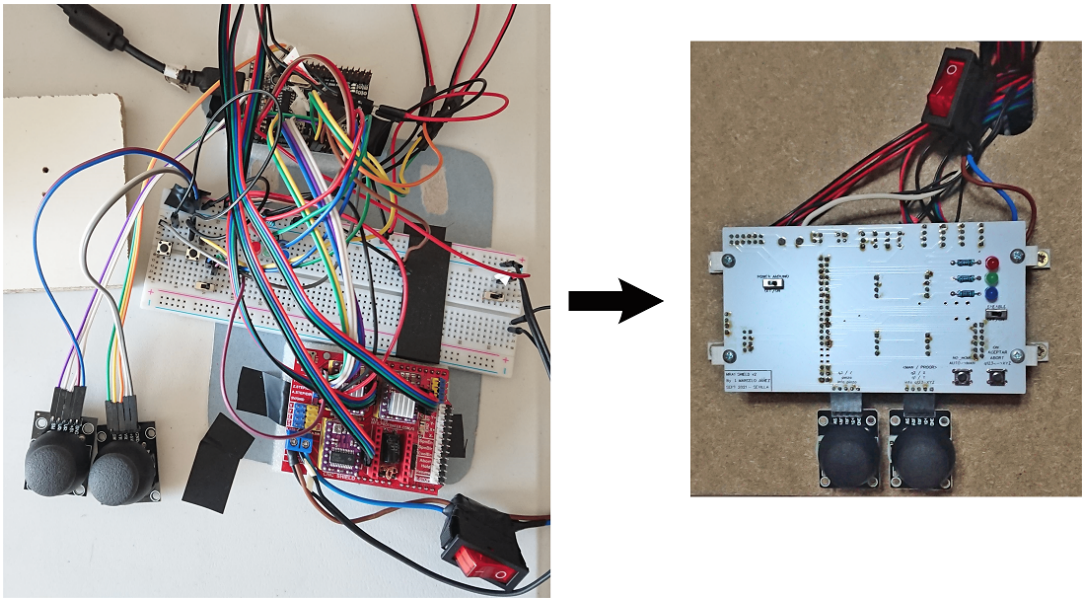

Figura 7.7 Comparativa del prototipado con el *Escudo*.

8 Conclusión

Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

Winston Churchill, 1942

8.1 Conclusiones

A lo largo de este documento se ha presentado el proceso de diseño, construcción, modelado y programación de un brazo articulado de tres grados de libertad, con la posibilidad de hacer uso de una pinza como efector final.

En el proceso de obtención del diseño utilizado, destaca la dificultad para lograr una versión definitiva y factible de las piezas que haga posible su construcción y permita un posterior funcionamiento adecuado del sistema. Para alcanzar este punto, fue necesaria una extensa fase de prototipado y realización de pruebas tanto dimensionales como de resistencia de las piezas, que hizo posible la obtención de la estructura hoy utilizada.

Desde el punto de vista del control, es conveniente señalar la efectividad que presenta la metodología utilizada para el control en bucle cerrado de la posición de los motores paso a paso, logrando de manera satisfactoria la sincronización del conjunto de actuadores durante la ejecución de los movimientos.

Por último, hacer mención al importante papel que puede presentar el tema del presupuesto en cualquier proceso de fabricación de una máquina desde cero, y la gran influencia que puede tener la existencia de horizontes temporales exigentes en el precio de adquisición de los productos necesarios. En el caso del presente proyecto, todos los materiales y dispositivos utilizados resultan tener un coste reducido y que ha sido asequible para el alumno; pero este hecho no quita que el proceso de prototipado haya requerido de invertir en numerosas ocasiones en la compra de algún elemento por duplicado, así como en la adquisición de numerosos componentes en cantidades mucho mayores a las que se necesitarían para la construcción del robot.

8.2 Trabajos futuros

Tras la finalización de este proyecto queda abierto un gran abanico de posibilidades para el desarrollo de nuevas aplicaciones del sistema obtenido. A continuación, se presentan las ideas que se consideran más interesantes y relevantes:

- Ampliación de los grados de libertad del brazo articulado mediante la introducción de eslabones y actuadores adicionales. Esta propuesta permitiría el control, no solo de la posición del efector final del

robot, sino también de su orientación, y estaría considerablemente propulsada por las posibilidades de expansión para las que está preparado el *Escudo* desarrollado para controlar el robot.

- Diseño y fabricación de efectores finales adicionales compatibles con su acoplamiento al último eslabón del robot, que permitan quizás el aprovechamiento de los nuevos grados de libertad introducidos, y la realización tareas que imiten a procesos tales como la colocación de componentes electrónicos con la orientación adecuada, o la ejecución de procesos de soldadura.
- Introducción de otros dispositivos en el sistema, como podrían ser una cinta transportadora o incluso una segunda versión de este robot, que permitan construir una pequeña célula de fabricación para la que programar tareas en las que se deba coordinar el funcionamiento de los agentes presentes ideando un sistema de comunicación entre éstos.
- Obtención de una versión en metal de la estructura del robot, en colaboración con el Departamento de Ingeniería Mecánica y Fabricación de la Escuela, a partir de procesos de mecanizado, para obtener un brazo articulado que deje de estar condicionado por la flexibilidad del plástico y presente mayores prestaciones en cuanto a precisión en los movimientos y resistencia a los esfuerzos externos.
- Desarrollo de un simulador del funcionamiento del brazo articulado, como el que presentan muchos robots utilizados en la industria, que permita la previsualización de los movimientos a ejecutar de acuerdo con el código desarrollado y facilite la depuración de las tareas programadas.

Apéndice A
Librería del control

A continuación se muestra el contenido completo de la librería ***mra1.h***.

Código A.1 Librería *mra1.h*.

```
#include <Math.h>
#include <Servo.h>

// DIMENSIONES DEL ROBOT ////////////////////////////////////////
const float L1 = 154.85 *1e-3;
const float L2 = 178.1  *1e-3;
const float L3 = 48.5   *1e-3;
const float L4 = 170.0  *1e-3;
const float L5 = 70.0   *1e-3;
const float L6 = -1.0   *1e-3;

const float L36 = L3 - L6;
const float L45 = L4 + L5;

const float xmax = L2 + sqrt(pow(L45,2) + pow(L36,2));
const float zmax = L1 + L2 + sqrt(pow(L45,2) + pow(L36,2));

const float colision = 3.5*1e-2;

const float pi = 3.14159265358979323846;



// PARAMETROS ///////////////////////////////////////////

const int PPR = 200;          // Pasos por revolución del Nema 17
const int MP = 32;            // Micropasos con ls que se configura el Drv8825
const int R = 3;              // Reductora
const int MPR = MP*R;         // Micropasos y Reductora
const int PPREV = PPR * MPR;    // Pasos para una vuelta completa del eje de la articulacion
//const float GPP = 360.0/PPREV;   // [grados/pulso]: Grados que se mueve el eje de la articulacion por pulso que recibe el drv
//const float PPG = PPREV/360.0;   // [pulsos/grados]
const float RadPP = (2*pi)/PPREV;   // [radianes/pulso]
const float PPRad = PPREV/(2*pi);   // [pulsos/radian]


// PINES ////////////////////////////////////////////
const int ENEABLE = 30;
const int ENEABLE_realim = 31;

const int DIR[] =   {2, 4, 6};
const int STEP[] =  {3, 5, 7};

const int FDC[] =   {14, 15, 16};

const int PINZA = 10;

int JOY[4] =   {1, 0, 2, 3}; // VRX1, VRY1, VRX2, VRY2. No es "const" porque se modifica al cambiar del modo manual q123 al xyz y viceversa
const int BOTZ1 = 22;
const int BOTZ2 = 23;

// Botón multiusos1
const int MULTI1 = 24;
const int ON = 0;
const int ACEPTAR = 0;
const int TIPO_MAN = 0;
const int ABORT = 0;

// Botón multiusos2
const int MULTI2 = 25;
const int BNO_HOME = 1;
const int MAN_AUTO = 1;


// LEDs
const int LEDR = 26;
const int LEDG = 27;
const int LEDB = 28;



// VARIABLES Y PARÁMETROS GENERALES ////////////////////////////////////////////
int i_aux;
```

```
const bool CW = false;
const bool CCW = true;
bool sentido_positivo[] = {CCW, CCW, CCW};   // Sentido de giro que se considera como positivo en cada motor
int sentido_p[3];                            // Sentido de giro actual del motor, en pasos, para modificar la posicion del motor

const bool FDCA = true;    // Final de Carrera Activo

// Posición actual de cada motor en pasos
int q_p[3]={0, 0, 0};           // (con signo) Posicion actual en pasos

// Coordenadas articulares del robot cuando se encuentra en su posición de reposo
// float qreposo[3] = {0.0, −90.0, 180.0};
const float qreposo[3] = {0.0, −pi/2.0, pi};

// Límites de las coordenadas articulares q123
const float q123max[3] = { pi,  pi/2.0,  pi};
const float q123min[3] = {−pi, −pi/2.0, 0.0};


////// Variables para la función "configura_mvto_parabolico"/////////////////////////////////////////
// Velocidad máxima de movimiento del robot, saturada desde la programación
const float vmax_sat = 60.0*pi/180.0;  // de [grados/s] a [rad/s]. Por encima de este valor, los movimientos son demasiado bruscos, y además la velocidad del micro empieza a quedarse corta

// Máximo número de puntos considerado en una misma trayectoria (ya sea interpolada o no)
# define NPTOS_DIV 60

// Array global para almacenar la secuencia de puntos a recorrer
float qdestinos[3][NPTOS_DIV];
float vmax_destinos[NPTOS_DIV];
int npuntos;
int npuntos_acumulados;

// Parámetros límites del robot
const float a_abs[] = {500.0*pi/180.0, 500.0*pi/180.0, 500.0*pi/180.0};   // de [grados/s2] a [rad/s2]: aceleraciones máximas en valor absoluto


// Variables auxiliares para almacenar la información de cada movimiento parabólico
const float T12min = 0.1;   // Valor mínimo que se asignará a T1 y T2 cuando, de acuerdo con los puntos a recorrer, resulte que deban ser nulos

float v1_pud[3][NPTOS_DIV];         // (con signo) velocidad constante del tramo lineal en [pasos/us] (la _d es de "double" (o float))

float a_2d[3][NPTOS_DIV];           // (0.5*aceleración), en [pasos/us^2]. Va asociada a las pendientes de las curvas de velocidad con variación lineal

long taus[3][NPTOS_DIV];    // tau en [us]
long T1s[NPTOS_DIV];     // T1 en [us]


// Variables auxiliares a usar cada vez que se llame la función "configura_mvto_parabolico"
int pto_i;

float Dq10[3];
float Dq10_max;
float Dq21[3];
float Dq21_max;

int motor_Dq10_max;
int motor_Dq21_max;

float T1;
float T2;
float tau;
float amax;

float pv1_d;
float v2_d;
float pac_d;

bool flag_repetir_pto = false;
float T1min;
float T1min_mot;




////// Variables para la función "run_motor"/////////////////////////////////////////
// Variables de tiempo para la aplicación de pulsos
long tini_tramo[3];   // Instante de inicio de la aceleración o desaceleración que estén en curso
long tlast[3];   // Ultimo instante en que se dió un pulso al motor

float vact_pud[3];              // (con signo) velocidad actual (instantanea) en [pasos/us]
long dT_p[3];                  // delta de tiempo en [us] que debe transcurrir hasta el proximo pulso del motor

const float vmin = 10.0*pi/180.0;  // de [grados/s] a [rad/s]: velocidad considerada como mínima para los motores.
                  // Experimentalmente se obtiene que velocidades inferiores a 0.1grados/s no tienen sentido en esta aplicación
                  // Pero además, de acuerdo con la estrategia de control utilizada, para P=0.0001, la vact(mínima) es 1.875grados/s (explicado en Prog.11)
                  // por lo que tomamos este valor auxiliar para poder establecer un valor de dT alto pero no infinito cuando el e_p resulte e_p=0.
                  // Originalmente se tomó vmin=1grado/s, por ser suficientemente pequeña y menor que la vact(minima)=1.875grados/s. Pero esta vmin nunva es aplicada,
                  // sino solo usada como parámetro auxiliar para calcular dT cuando NO se da pulso porque vact es muy baja. Dado que vact solo pasa por valores bajos
                  // cuando se están produciendo las aceleraciones, y estas son las zonas donde la curva de qdes(t) cambia con mayor rapidez, establecemos una vmin mayor
                  // que la originalmente escogida de 1grado/s, para que cuando se pase por vact~0, la qdes(t) aún así se siga evaluando con mayor frecuencia.
const float vmin_pud = PPRad*vmin/1e6;   // [radianes/us]

int pto_motor[3];          // Contador de para qué punto se están trazando el tramo rectílineo y curvo

bool finalizado[] = {true, true, true};   // Movimiento en curso de motor en cuestión finalizado. Es una variable interna del movimiento total del motor
bool tramo_finalizado[] = {false, false, false};

// Variables de tiempo
long t_act;  // Valor actual de micros()
long t_dif;  // Tiempo transcurrido desde el inicio del tramo actual del movimiento de un motor

// Controlador
int e_p = 0;   // error de posición actual en [pasos]   Se corresponde con: e_p = qdeseada[pasos] − qreal[pasos];
const float P = 0.0001;   // Constante de proporcionalidad del controlador C(s) = P
            // P está en unidades de [us^(−1)], ya que v_up está en [pasos/us], y e_p en [pasos]



////// Variables para la función "MOVEQp"/////////////////////////////////////////
bool completado_mot[3];

////// Variables para la gestión de las funciones acumulativas /////////////////////////////////////////
bool moveq_pte = false;
```

```
bool movej_pte = false;



////// Variables para la función "HOME"/////////////////////////////////////////
// Velocidades de los movimientos durante el HOME
const float v_home_r = 30.0*pi/180.0;  // Velocidad rápida de aproximación al FDC hasta tocarlo por primera vez
const float v_home_l = 5.0*pi/180.0;   // Velocidad lenta para ajuste fino de la posición del FDC

const float alejamiento = 10.0*pi/180.0;    // de [grados] a [radianes] que cada brazo se separará de su FDC tras tocarlo y dejar de tocarlo, antes de hacer el ajuste fino

bool en_home = false; // Variable a activar durante la realización del HOME para permitir exceder los limites de q123


////// Variables para la función "CD"/////////////////////////////////////////
float q123[3];
float xyz[3];



////// Variables para la función "MOVEL"/////////////////////////////////////////
const float paso_inter = 0.005; // Paso de interpolación en [m]
float xdes, ydes, zdes;
float distancia;
float dist[3];
float dist_tot;
int npuntos_inter;
float paso_xyz[3];
int eje_mayor;
float xyz_mayor;



////// Variables para "Manual()"/////////////////////////////////////////
// Variables de lecturas
int joy[4]; // Lecturas de los Joysticks

// Calibrado de los Joystics
const int calibre_joy[4] = {512, 512, 512, 512};
const int delta_joy = 80; // Variación a partir de la cual se considera que los joysticks están activos

// Control del servo de la pinza
const int pinza_sup = 168;
const int apertura_max_pinza = 90;
Servo pinza;
int apertura_act_pinza = 0;
long tlast_pinza = 0;
const long dT_pinza_man = 30*1e3; // Asociado a la velocicad de apertura y cierre de la pinza en el modo Manual
const long dT_pinza_auto = 15*1e3;

// Mostrar información por pantalla al pulsar los botones de los joysticks
bool botz1 = true;
bool botz2 = true;
bool botz1_ant = true;
bool botz2_ant = true;

// Control de los motores
const long dT_man = (1e6)/(20.0*pi/180.0*PPRad);  //dT en [us] asociado a la velocidad máxima dentro del Modo Manual, en [grados/s]
long dT_manual[3] = {dT_man, dT_man, dT_man};
bool tipo_manual = false;

// Parámetros para el movimiento en el modo Manual a 1mm/s
// Hacemos que los valores de xyz puedan variar cada de 250 us (El dT asociado a una velocidad articular de 60 grados/s era de 312 us)
const long dT_xyz = 200;
const float inc_xyz = 100*(1e-3)*dT_xyz/1e6;  // Para variar con el dT usado
long tlast_xyz = 0;

// Variables para el control trabajando con xyz
float xyz_des[3];



//////  Variabls para gestionar la Máquina de estados /////////////////////////////////////////
bool flag_ejecuta_funcion = true; // Flag que permite parar de ejecutar el código del usuario si mientras se ejecuta se pasa a los modos Abort/Manual/Desenergizado

# define OFF       0
# define INICIO    1
# define PREPARADO  2
# define CODIGO    3
# define BLOQUEADO  4
# define MANUAL    5
# define DES       6
const char nombre_estado[][10] = {"OFF", "INICIO", "PREPARADO", "CODIGO", "BLOQUEADO", "MANUAL", "DES"};
int estado = 0;
int estado_ant = -1;

long tlast_parpadeo = 0;
long dT_parpadeo1 = 1.0*1e6;
bool ledr = false;
bool ledg = false;


bool botones[2];
bool botones_ant[2];




////////////////////// Cabeceras de las funciones ////////////////////////////////////////////////////////////////
void codigo_usuario(void);
void general(void);
void Manual(void);
void CLOSE(void);
void OPEN(int apertura_deseada);
void prepara_parabolico(void);
void configura_mvto_parabolico(void);
bool run_motor(int motor);
void almacena_q(float q_d1, float q_d2, float q_d3, float v_max);
void MOVEQp(float q_d1, float q_d2, float q_d3, float v_max);
void MOVEJp(float x, float y, float z, float v_max);
void MOVEQ(float q_d1, float q_d2, float q_d3, float v_max);
```

```
void MOVEJ(float x, float y, float z, float v_max);
void fin_programa(void);
void ejecuta_moveq(void);
void PAUSA(float duracion);
void HOME(void);
void NO_HOME(void);
void CD(void);
void CI(void);
void MOVELp(float xdes, float ydes, float zdes, float vmax);
bool dentro_ET(void);
bool dentro_q123(void);









////////////////////// setup() //////////////////////////////////////////////////////////////

void setup()
{
 Serial.begin(115200);
 pinMode(LED_BUILTIN, OUTPUT);


 // Detectamos si los drivers son habilitados físicamente
 pinMode(ENEABLE, OUTPUT);
 digitalWrite(ENEABLE, HIGH); // Motores inicialmente apagados

 pinMode(ENEABLE_realim, INPUT_PULLUP); // Detección de si el Eneable ha sido apagado físicamente con el interruptor


 for(i_aux=0; i_aux<3; i_aux++)
 {
  pinMode(DIR[i_aux], OUTPUT);
  pinMode(STEP[i_aux], OUTPUT);

  pinMode(FDC[i_aux], INPUT_PULLUP);

  digitalWrite(DIR[i_aux], sentido_positivo[i_aux]);
  sentido_p[i_aux] = 1;
 }

 pinMode(BOTZ1, INPUT_PULLUP);
 pinMode(BOTZ2, INPUT_PULLUP);


 pinza.attach(PINZA);
 pinza.write(pinza_sup);

 pinMode(MULTI1, INPUT_PULLUP);
 pinMode(MULTI2, INPUT_PULLUP);


 // LEDs inicialmente apagados
 pinMode(LEDR, OUTPUT);
 digitalWrite(LEDR, LOW);
 pinMode(LEDG, OUTPUT);
 digitalWrite(LEDG, LOW);
 pinMode(LEDB, OUTPUT);
 digitalWrite(LEDB, LOW);


 Serial.println("ROBOT ENCENDIDO");
 //while(!Serial);
 //Serial.println("qdes q_p");
}




////////////////////// loop() //////////////////////////////////////////////////////////////

void loop()
{
 botones[0] = !digitalRead(MULTI1);
 botones[1] = !digitalRead(MULTI2);


 //NOTA: es necesario detecar el valor de "ENEABLE_realim" dentro de cada "case" y no fuera del switch porque dentro del estado OFF NO queremos conmutar a DES, ya que
 // si no el programa entraríe en un bucle de Activar–Desactivar el Eneable
 // Es decir, del estado OFF nunca se puede pasar a DES, porque en el estado DES se activa el Eneable para poder detectar su realimentación física

 // Máquina de estados que gestiona el funcionamiento del robot
 switch (estado)
 {
  case OFF: // El robot se ha enchufado pero aún no se le ha dado a encender
    // ENEABLE apagado
    digitalWrite(ENEABLE, HIGH);

    // Parpadeo del LEDR
    digitalWrite(LEDG, LOW);
    digitalWrite(LEDB, LOW);

    t_act = micros();
    if((t_act − tlast_parpadeo) >= dT_parpadeo1)
    {
     ledr = !ledr;
     digitalWrite(LEDR, ledr);
     tlast_parpadeo = t_act;
    }

    if(botones[ON] && !botones_ant[ON])
    {
     // Esperamos a que se suelte el boton y guardamos que ya no esta pulsado
     while(botones[ON])
```

```
    {
      botones[ON] = !digitalRead(MULTI1);
    }
    delay(20);
    estado = INICIO;
  }

  break;


case INICIO:
  // Activamos el ENEABLE. Si el Eneable esta cortado fisicamente, nos iremos a DES
  digitalWrite(ENEABLE, LOW);

  // Parpadeo del LEDR
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, LOW);

  t_act = micros();
  if((t_act − tlast_parpadeo) >= dT_parpadeo1)
  {
    ledr = !ledr;
    digitalWrite(LEDR, ledr);
    tlast_parpadeo = t_act;
  }

  // Si los motores no están energizados, nos vamos directamente al estado DES
  if(digitalRead(ENEABLE_realim))
  {
    estado = DES;
  }
  else
  {
    // Paso a PREPARADO sin pasar por HOME
    if((botones[BNO_HOME]) && (botones[ACEPTAR] && !botones_ant[ACEPTAR]))
    {
      digitalWrite(LEDR, HIGH);
      digitalWrite(LEDG, LOW);
      digitalWrite(LEDB, LOW);

      // Esperamos a que se suelte el boton y guardamos que ya no esta pulsado
      while(botones[ACEPTAR])
      {
        botones[ACEPTAR] = !digitalRead(MULTI1);
      }
      delay(20);

      NO_HOME();
      estado = PREPARADO;
    }

    // Paso a PREPARADO haciendo HOME
    else if((!botones[BNO_HOME]) && (botones[ACEPTAR] && !botones_ant[ACEPTAR]))
    {
      digitalWrite(LEDR, HIGH);
      digitalWrite(LEDG, LOW);
      digitalWrite(LEDB, LOW);

      // Esperamos a que se suelte el boton y guardamos que ya no esta pulsado
      while(botones[ACEPTAR])
      {
        botones[ACEPTAR] = !digitalRead(MULTI1);
      }
      delay(20);

      // Ejecutamos HOME
      flag_ejecuta_funcion = true;
      HOME();

      // Si volvemos del HOME estando el eneable fisicamente activo, pasamos a PREPARADO
      if(!digitalRead(ENEABLE_realim))
      {
        estado = PREPARADO;
      }
      else
      {
        estado = DES;
      }
    }
  }
  break;


case PREPARADO:
  // Leds
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDB, LOW);

  // Si se desconecta eneable
  if(digitalRead(ENEABLE_realim))
  {
    estado = DES;
  }
  else
  {
    // Leemos el Joystick de y
    joy[1] = analogRead(1);

    // Si se pulsa a la Derecha, ejecutamos el código del usuario
    if(joy[1] < delta_joy)
    {
      estado = CODIGO;

      // Esperamos a que se suelte el joystick
      while(joy[1] < delta_joy)
      {
        joy[1] = analogRead(1);
      }
    }
    else if(joy[1] > (1023−delta_joy))
    {
```

```
      // El modo Manual siempre empieza en q123
      tipo_manual = false;
      JOY[0] = 1; // q1 en el joystick de y
      JOY[1] = 0; // q2 en el joystick de x
      estado = MANUAL;

      // Esperamos a que se suelte el joystick del todo
      while(joy[1] > (calibre_joy[1]+delta_joy))
      {
        joy[1] = analogRead(1);
      }
    }

  }

  break;



case CODIGO:
  // Si los motores no están energizados, nos vamos directamente al estado DES
  if(digitalRead(ENEABLE_realim))
  {
    estado = DES;
  }
  else
  {
    // Leds
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDG, LOW);
    digitalWrite(LEDB, HIGH);

    // Ejecutamos el código del usuario
    flag_ejecuta_funcion = true;
    codigo_usuario();
    // Al terminar el código, hay que ejecutar "fin_programa()" para ejecutar los movimientos acumulativos pendientes
    fin_programa();

    // Tras completar los movimientos, pasamos al estado BLOQUEADO
    estado = BLOQUEADO;
  }
  break;



case BLOQUEADO:
  // Si los motores no están energizados, nos vamos directamente al estado DES
  if(digitalRead(ENEABLE_realim))
  {
    estado = DES;
  }
  else
  {
    // Leds
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDB, LOW);

    // Parpadeo del Led verde
    t_act = micros();
    if((t_act − tlast_parpadeo) >= dT_parpadeo1)
    {
      ledg = !ledg;
      digitalWrite(LEDG, ledg);
      tlast_parpadeo = t_act;
    }

    // Conmutación al Modo Manual
    if(botones[MAN_AUTO] && !botones_ant[MAN_AUTO])
    {
      // El modo Manual siempre empieza en q123
      tipo_manual = false;
      JOY[0] = 1; // q1 en el joystick de y
      JOY[1] = 0; // q2 en el joystick de x
      estado = MANUAL;

      // Esperamos a que se suelte el boton
      while(botones[MAN_AUTO])
      {
        botones[MAN_AUTO] = !digitalRead(MULTI2);
      }
      delay(20);
    }

    // Conmutación a movimiento al punto de reposo y vuelta a PREPARADO
    if(botones[ACEPTAR] && !botones_ant[ACEPTAR])
    {
      digitalWrite(LEDR, LOW);
      digitalWrite(LEDG, HIGH);
      digitalWrite(LEDB, LOW);

      // Para que "general()" no vuelva a detectar que se acaba de pulsar el boton
      botones_ant[ACEPTAR] = botones[ACEPTAR];
      delay(20);
      // Ejecutamos un movimiento a la posición de reposo
      flag_ejecuta_funcion = true;
      MOVEQp(180.0/pi*qreposo[0], 180.0/pi*qreposo[1], 180.0/pi*qreposo[2], 20.0);

      // Si volvemos del movimiento porque los motores se han apagado:
      if(digitalRead(ENEABLE_realim))
      {
        estado = DES;
      }
      // Si volvemos porque se aborto el movimiento hacia qreposos, volvemos a quedarnos en estado BLOQUEADO
      else if(flag_ejecuta_funcion==false)
      {
        estado = BLOQUEADO;
        //Evitamos redetectar el flanco
        botones_ant[ABORT] = botones[ABORT];
      }
      // Si se ha completado el movimiento hasta qreposos, pasamos a PREPARADO
      else
      {
        estado = PREPARADO;
```

```
        }
       }

      }
      break;


    case MANUAL:
      // Si los motores no están energizados, nos vamos directamente al estado DES
      if(digitalRead(ENEABLE_realim))
      {
       estado = DES;
      }
      else
      {
       // Leds
       digitalWrite(LEDR, LOW);
       digitalWrite(LEDG, HIGH);
       digitalWrite(LEDB, HIGH);

       // Ejecutamos el modo Manual
       Manual();

       // Detectamos la salida del Modo Manual
       if(botones[MAN_AUTO] && !botones_ant[MAN_AUTO])
       {
        // Esperamos a que se suelte el boton
        while(botones[MAN_AUTO])
        {
         botones[MAN_AUTO] = !digitalRead(MULTI2);
        }
        delay(20);

        // Cambiamos a BLOQUEADO
        estado = BLOQUEADO;
       }
      }
      break;


    case DES:
      // Dejamos ENEABLE Activo, ya que será la forma de comprobar que el usuario lo ha activado físicamente
      digitalWrite(ENEABLE, LOW);

      // Leds
      digitalWrite(LEDR, HIGH);
      digitalWrite(LEDG, LOW);
      digitalWrite(LEDB, LOW);

      if(!digitalRead(ENEABLE_realim))
      {
       estado = OFF;
      }

      break;


   }

   // Imprimimos el cambio de estado
   if(estado!=estado_ant)
   {
    Serial.print("\n\nESTADO ");
    Serial.print(estado);
    Serial.print(": ");
    Serial.println(nombre_estado[estado]);
   }
   estado_ant = estado;

   // Guardamos el valor actual de los botones
   botones_ant[0] = botones[0];
   botones_ant[1] = botones[1];
  }




 ////////////////////// general ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
 // Â·general: función que permite la detección de la pulsación de los botones durante la ejecución del código del usuario, permitiendo detectar si se pasa a los modos:
 //  –Abort
 //  –Manual
 //  –Desenergizado

 void general(void)
 {
  botones[0] = !digitalRead(MULTI1);
  botones[1] = !digitalRead(MULTI2);

  if((botones[ABORT] && !botones_ant[ABORT]) || (botones[MAN_AUTO] && botones_ant[MAN_AUTO]) || digitalRead(ENEABLE_realim))
  {
   // Indicamos que no se debe seguir ejecutando el movimiento
   flag_ejecuta_funcion = false;

   // Reseteamos las principales variables del movimiento
   moveq_pte = false;
   movej_pte = false;
   npuntos_acumulados = 0;

   // Guardamos el modo al que se va a pasar
   if(digitalRead(ENEABLE_realim))
   {
    estado = DES;
   }
   else if(botones[ABORT])
   {
    estado = ABORT;
   }
   else if(botones[MAN_AUTO])
   {
```

```
    // El modo Manual siempre empieza en q123
    tipo_manual = false;
    JOY[0] = 1; // q1 en el joystick de y
    JOY[1] = 0; // q2 en el joystick de x
    estado = MANUAL;
   }

 }


 // Guardamos el valor actual de los botones
 botones_ant[0] = botones[0];
 botones_ant[1] = botones[1];

}




////////////////////// Manual ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·Manual: código del control manual

void Manual(void)
{
 int motor;


 // Lectura del joystick asociado a la pinza
 joy[3] = analogRead(JOY[3]);


 t_act = micros();


 // PINZA: control de la pinza
 if((t_act−tlast_pinza)>dT_pinza_man)
 {
  tlast_pinza = t_act;

  // Joystick hacia 1023, valor del servo crece, valor de "apertura_act_pinza" decrece
  if(joy[3] < delta_joy && apertura_act_pinza < apertura_max_pinza)
  {
   apertura_act_pinza++;
   pinza.write(pinza_sup − apertura_act_pinza);
  }
  else if(joy[3] > (1023−delta_joy) && apertura_act_pinza > 0)
  {
   apertura_act_pinza−−;
   pinza.write(pinza_sup − apertura_act_pinza);
  }

 }


 // Mostrar información por pantalla cuando se pulsen los botones de los joystics
 botz1 = digitalRead(BOTZ1);
 botz2 = digitalRead(BOTZ2);
 // q123 xyz
 if(!botz1 && botz1_ant)
 {
  Serial.println("−Manual: información de la posición actual del robot:");
  q123[0] = RadPP*q_p[0];
  q123[1] = RadPP*q_p[1];
  q123[2] = RadPP*q_p[2];
  CD();
  Serial.print("\t(q1, q2, q3) =");
  Serial.print(" ("); Serial.print(q123[0]*180.0/pi, 3);
  Serial.print(", "); Serial.print(q123[1]*180.0/pi, 3);
  Serial.print(", "); Serial.print(q123[2]*180.0/pi, 3); Serial.println(")");

  Serial.print("\t(x, y, z) =");
  Serial.print(" ("); Serial.print(xyz[0], 5);
  Serial.print(", "); Serial.print(xyz[1], 5);
  Serial.print(", "); Serial.print(xyz[2], 5); Serial.println(")");

  Serial.println("");
 }
 botz1_ant = botz1;

 // pinza
 if(!botz2 && botz2_ant)
 {
  Serial.println("−Manual: información de la posición actual de la pinza:");
  Serial.print("\tApertura = "); Serial.print(apertura_act_pinza); Serial.println("grados");
  Serial.println("");
 }
 botz2_ant = botz2;



 // Conmutación entre el Modo Manual q123 y el xyz
 botones[TIPO_MAN] = !digitalRead(MULTI1);
 if(botones[TIPO_MAN] && !botones_ant[TIPO_MAN])
 {
  // Si se pretende cambiar del modo q123 al xyz, hay que comprobar que el punto actual esté dentro, y establecer el valor inicial de xyz_des
  if(!tipo_manual)
  {
   q123[0] = RadPP*q_p[0];
   q123[1] = RadPP*q_p[1];
   q123[2] = RadPP*q_p[2];
   CD();

   if(dentro_ET())
   {
    xyz_des[0] = xyz[0];
    xyz_des[1] = xyz[1];
    xyz_des[2] = xyz[2];

    tipo_manual = true;
    // En el modo Manual XYZ cambiamos el orden de los valores analógicos respecto al q123
```

```
      JOY[0] = 0; // x
      JOY[1] = 1; // y

      Serial.println("-Manual: Cambiando de modo q123 al modo xyz");
    }

  }
  // Del modo xyz al modo q123 se cambia siempre sin problemas
  else
  {
    tipo_manual = false;
    JOY[0] = 1; // q1 en el joystick de y
    JOY[1] = 0; // q2 en el joystick de x
    Serial.println("-Manual: Cambiando de modo xyz al modo q123");
  }
 }
 botones_ant[TIPO_MAN] = botones[TIPO_MAN];




 // Control articular de los motores en modo Manual
 // Lectura de los joystics asociados a los motores
 for(motor=0; motor<3; motor++)
 {
   joy[motor] = analogRead(JOY[motor]);
 }

 // Volvemos a leer el sintante actual
 t_act = micros();




 // Modo Manual q123
 if(!tipo_manual)
 {
  // Introducimos los valores de la posición actual en q123
  q123[0] = RadPP*q_p[0];
  q123[1] = RadPP*q_p[1];
  q123[2] = RadPP*q_p[2];

  // Para cada motor:
  for(motor=0; motor<3; motor++)
  {
    // Comprobamos si ha pasado el dT_manual desde la última pulsación
    if((t_act - tlast[motor]) > dT_manual[motor])
    {

      // Modificacion del joystick de q1 para que su control sea intuitivo
      if(motor==0)
      {
        joy[motor] = 1023 - joy[motor];
      }



      // Si el joystick está hacia su nivel minimo, y la coordenada articular no está saturada
      if( (joy[motor] < (calibre_joy[motor]-delta_joy)) && (q123[motor] < q123max[motor]) )
      {
        // El motor se mueve en su sentido positivo
        sentido_p[motor] = 1;
        digitalWrite(DIR[motor], sentido_positivo[motor]);

        // Damos el pulso
        digitalWrite(STEP[motor], HIGH);
        digitalWrite(STEP[motor], LOW);
        q_p[motor] += sentido_p[motor];
        tlast[motor] = t_act;

        // Modificacion de dT_manual de acuerdo con la posición del joystick
        // Cuando el joystick puede valer entre [0, calibre_joy[motor]-delta_joy)]
        dT_manual[motor] = dT_man * calibre_joy[motor]/(calibre_joy[motor]-joy[motor]);

      }
      else if( (joy[motor] > (calibre_joy[motor]+delta_joy)) && (q123[motor] > q123min[motor]) )
      {
        // El motor se mueve en su sentido negativo
        sentido_p[motor] = -1;
        digitalWrite(DIR[motor], !sentido_positivo[motor]);

        // Damos el pulso
        digitalWrite(STEP[motor], HIGH);
        digitalWrite(STEP[motor], LOW);
        q_p[motor] += sentido_p[motor];
        tlast[motor] = t_act;

        // Modificacion de dT_manual de acuerdo con la posición del joystick
        // Cuando el joystick puede valer entre [calibre_joy[motor]+delta_joy), 1023]
        dT_manual[motor] = dT_man * calibre_joy[motor]/(joy[motor]-calibre_joy[motor]);
      }
    }

  }
 }

 // Modo Manual xyz
 else // if(tipo_manual)
 {
  // Si ha transcurrido dT_xyz, revisamos los joysticks para modificar las referencias de xyz
  if((t_act - tlast_xyz) > dT_xyz)
  {
    tlast_xyz = t_act;

    for(motor=0; motor<3; motor++)
    {
      // En el modo Manual xyz nos interesa conmutar el "sentido positivo" del eje Y:
      if(motor==1)
      {
```

```
          joy[motor] = 1023 - joy[motor];
        }
        // Si los joyzticks están activos a su nivel mímimo o máximo, variamos los valores de las xyz de referencia
        if (joy[motor] < (calibre_joy[motor]-delta_joy))
        {
          xyz[motor] = xyz_des[motor] + inc_xyz * (calibre_joy[motor]-joy[motor])/calibre_joy[motor];
        }
        else if(joy[motor] > (calibre_joy[motor]+delta_joy))
        {
          xyz[motor] = xyz_des[motor] - inc_xyz * (joy[motor]-calibre_joy[motor])/calibre_joy[motor];
        }
      }

      // Comprobamos si la posixión xyz a la que se pretendería llegar está dentro del espacio de trabajo, y solo entonces la hacemos efectiva
      if(dentro_ET())
      {
        xyz_des[0] = xyz[0];
        xyz_des[1] = xyz[1];
        xyz_des[2] = xyz[2];
      }

    }


    // Paralelamente a la modificación de las referencias xyz, controlamos los motores:
    // Obtenemos la CI de la posición actualmente deseada
    xyz[0] = xyz_des[0];
    xyz[1] = xyz_des[1];
    xyz[2] = xyz_des[2];
    CI();

    // Para cada coordenada articular:
    for(motor=0; motor<3; motor++)
    {
      // Si ha transcurrido el dT
      if(t_act-tlast[motor] >= dT_p[motor])
      {
        // Calculamos el error de posición en pasos. La posición q123 deseada está almacenada en q123[] tras hacer la CI
        e_p = PPRad*q123[motor] - q_p[motor];
        // Aplicamos la expresión del control
        vact_pud[motor] =P*e_p;


        // Si la velocidad es suficientemente alta, aplicamos pulso y recalculamos dT
        if(vact_pud[motor]!=0)
        {
          // Si la velocidad es demasiado alta, la saturamos (vmax_sat en [radianes/s])
          if(abs(vact_pud[motor]) > (PPRad*vmax_sat/1e6))
          {
            if(vact_pud[motor] > 0)
            {
              vact_pud[motor] = PPRad*vmax_sat/1e6;
            }
            else
            {
              vact_pud[motor] = -PPRad*vmax_sat/1e6;
            }
          }

          // Detectamos si hay cambio de signo de la velocidad antes de dar el pulso
          if(vact_pud[motor]>0 && sentido_p[motor]<0)
          {
            sentido_p[motor] = 1;
            digitalWrite(DIR[motor], sentido_positivo[motor]);
          }
          else if(vact_pud[motor]<0 && sentido_p[motor]>0)
          {
            sentido_p[motor] = -1;
            digitalWrite(DIR[motor], !sentido_positivo[motor]);
          }

          // Damos el pulso
          digitalWrite(STEP[motor], HIGH);
          digitalWrite(STEP[motor], LOW);
          q_p[motor] += sentido_p[motor];
          tlast[motor] = t_act;

          // Calculamos el nuevo dT
          dT_p[motor] = 1/abs(vact_pud[motor]);

        }
        // Si vact_pud es demasiado pequeña (es nula), asignamos el dT asociado a la vmin, y no damos pulso
        else
        {
          tlast[motor] = t_act;
          dT_p[motor] = 1/vmin_pud;  // Valor alto de dT pero inferior al correspondiente a la vact
        }


      }
    }




  }



}




////////////////////// CLOSE ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·CLOSE: cierre total y brusco de la pinza

void CLOSE(void)
```

```
{
  // Si existe algún movimiento acumulativo pendiente, debe ser ejecutado primero
  if(moveq_pte || movej_pte)
  {
    ejecuta_moveq();
  }


  apertura_act_pinza = 0;
  pinza.write(pinza_sup – apertura_act_pinza);
  Serial.println("–CLOSE: cierre total de la pinza");
}







////////////////////// OPEN ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·OPEN: apertura y cierre progresivos de la pinza dejando un determinado ángulo entre sus partes (el ángulo total será el doble del indicado como argumento de la función)

void OPEN(int apertura_deseada)
{
  // Si existe algún movimiento acumulativo pendiente, debe ser ejecutado primero
  if(moveq_pte || movej_pte)
  {
    ejecuta_moveq();
  }




  if (apertura_deseada > apertura_max_pinza)
  {
    apertura_deseada = apertura_max_pinza;
  }
  else if(apertura_deseada < 0)
  {
    apertura_deseada = 0;
  }
  Serial.print("–OPEN: apertura de la pinza un ángulo "); Serial.print(apertura_deseada); Serial.println("grados");

  // Movimiento progresivo de la pinza
  while((apertura_act_pinza != apertura_deseada) && flag_ejecuta_funcion)
  {
    t_act = micros();

    if((t_act – tlast_pinza) > dT_pinza_auto)
    {
      tlast_pinza = t_act;

      if(apertura_act_pinza > apertura_deseada)
      {
        apertura_act_pinza ––;
      }
      else if(apertura_act_pinza < apertura_deseada)
      {
        apertura_act_pinza ++;
      }

      // Imponemos la apertura actual
      pinza.write(pinza_sup – apertura_act_pinza);
    }

    // Llamamos a general:
    general();
  }



}







////////////////////// configura_mvto_parabolico y prepara_parabolico ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Â·configura_mvto_parabolico: Función que establece los parámetros para calcular posiciones instantaneas deseadas de cada coordenada articular en un movimiento parabólico,
// en el que cada velocidad v(t) realiza tramos:
//    –Constantes ("Tramo reto")
//    –Lineales ("Tramo curvo"), variando según una a_max prefijada
// En la curva de q(t), cada punto tiene asociado asociado un tramo que consta de:
//    –Un primer subtramo recto
//    –Un segundo subtramo curvo
// Solo al final del todo existirá un subtramo recto (asociado al fin del movimiento) que NO va seguido de tramo curvo

// NOTA: Si npuntos<3, ESTA FUNCION NO VALE. Además, siempre será llamada con al menos 4 puntos, que sería el caso de un movimiento trapezoidal
// Es decir, si el usuario solo proporciona un punto, la función prepara_parabolico ya se encarga de añadir siempre 3 más

// Â·prepara_parabolico: función que prepara el array global qdestinos[][] para que contenga los puntos inicial y final repetidos y pueda llamarse a "configura_mvto_parabolico"



void prepara_parabolico(void)
{
  int k;
  int j;
  int motor;

  // Saturamos la velocidad máxima de movimiento del robot en cada tramo
  for(k=0; k<npuntos_acumulados; k++)
  {
    if(vmax_destinos[k] > vmax_sat)
    {
      vmax_destinos[k] = vmax_sat;
    }
  }
```

```
// Excluimos de los puntos acumulados a aquellos que excedan los límites alcanzables por el robot
for(k=0; k<npuntos_acumulados; k++)
{
 q123[0] = qdestinos[0][k];
 q123[1] = qdestinos[1][k];
 q123[2] = qdestinos[2][k];

 if(!dentro_q123())
 {
  Serial.println("-prepara_parabolico: descartando punto q123 que excede limites");

  for(j=k; j<npuntos_acumulados-1; j++)
  {
   qdestinos[0][j] = qdestinos[0][j+1];
   qdestinos[1][j] = qdestinos[1][j+1];
   qdestinos[2][j] = qdestinos[2][j+1];
  }

  // Decrementamos la cantidad de puntos acumulados
  npuntos_acumulados --;

  // Decrementamos la k para que no se salte ningún punto en la comprobación
  k--;
 }

}



if(npuntos_acumulados>0)
{
 // PREPARACION DE LOS PUNTOS
 // Se añaden tres puntos adicionales a los dados por el usuario
 npuntos = npuntos_acumulados + 3;

 // El último punto debe estar repetido
 for(motor=0; motor<3; motor++)
 {
  qdestinos[motor][npuntos-1] = qdestinos[motor][npuntos_acumulados-1];
 }

 // Desplazamos el resto de puntos dos posiciones
 for(k=npuntos_acumulados-1; k>=0; k--)
 {
  for(motor=0; motor<3; motor++)
  {
   qdestinos[motor][k+2] = qdestinos[motor][k];
  }
 }

 // El primer y segundo punto deben ser el mismo e iguales a la posición actual del robot
 for(k=0; k<2; k++)
 {
  for(motor=0; motor<3; motor++)
  {
   // La q_p está en [pasos], y las qdestinos están en [grados]
   qdestinos[motor][k] = RadPP*q_p[motor];
  }
 }


 // PREPARACION DE LAS VELOCIDADES MAXIMAS
 // Velocidades máximas de cada tramo: la primera y ultima vmax deben estar repetidas.
 // NO van a intervenir ya que en el primer y ultimo tramo se aplican T1 y T2 iguales a T12min,
 // pero es necesario inroducirlas para que el algoritmo sea programable de forma cómoda
 // Nota: la cantidad de vmax es (npuntos_acumulados + 2), ya que el primer punto NO debe ser alcanzado (ya se está en el)

 // Repetimos la ultima vmax
 vmax_destinos[npuntos_acumulados+1] = vmax_destinos[npuntos_acumulados-1];

 // Desplazamos todas las vmax una posición a la derecha
 for(k=npuntos_acumulados-1; k>=0; k--)
 {
  vmax_destinos[k+1] = vmax_destinos[k];
 }
 // Repetir la vmax inicial no es necesario, ya que ya está guardada en vmax_destinos[0]


 // Llamamos a la función encargada de calcular los parámetros del movimiento
 configura_mvto_parabolico();
}
}




void configura_mvto_parabolico(void)
{
 int motor = 0;
 // El último punto para el que tenemos que hacer cálculos es el antepenúltimo punto
 for(pto_i=0; pto_i<(npuntos-2); pto_i++)
 {
  // Búsqueda del motor que realiza el mayor movimiento en cada tramo para calcular T1 y T2
  motor_Dq10_max=0;
  Dq10_max = 0;
  motor_Dq21_max=0;
  Dq21_max = 0;
  for(motor=0; motor<3; motor++)
  {
   Dq10[motor] = qdestinos[motor][pto_i+1] - qdestinos[motor][pto_i];
   Dq21[motor] = qdestinos[motor][pto_i+2] - qdestinos[motor][pto_i+1];

   if(abs(Dq10[motor])>Dq10_max)
   {
    motor_Dq10_max = motor;
    Dq10_max = abs(Dq10[motor]);
   }

   if(abs(Dq21[motor])>Dq21_max)
   {
    motor_Dq21_max = motor;
```

```
      Dq21_max = abs(Dq21[motor]);
    }
  }

  // Cáculo de T1 y T2
  // Si estamos en el tramo inicial, que consiste siempre en ir de la posición inicial a sí misma, hay que imponer un T1 no nulo
  if(Dq10[motor_Dq10_max]==0)
    T1 = T12min;
  else
  // En el tramo de q0 a q1 interviene la vmax asociada a salir de q0
    T1 = abs(Dq10[motor_Dq10_max])/vmax_destinos[pto_i];

  // Lo mismo ocurre con T1 y el tramo final
  if(Dq21[motor_Dq21_max]==0)
    T2 = T12min;
  else
  // En el tramo de q1 a q2, interviene la vmax del siguiente punto
    T2 = abs(Dq21[motor_Dq21_max])/vmax_destinos[pto_i+1];


  // Cálculo de los parámetros de los tramos rectilineo y curvo tras el punto actual
  for(motor=0; motor<3; motor++)
  {
    // v1 (v cte) del tramo rectilineo del motor
    v1_pud[motor][pto_i] = (PPRad*Dq10[motor]/T1)/1e6;    // [pasos/us]


    // Aceleración a aplicar en el tramo curvo (calculamos directamente la aceleración con signo partida por dos)
    if(Dq21[motor]/T2==Dq10[motor]/T1)
    {
      a_2d[motor][pto_i] = 0.0;
      amax = 0.0;
    }
    else if(Dq21[motor] > Dq10[motor])
    {
      a_2d[motor][pto_i] = (PPRad*0.5*a_abs[motor])/1e12;   // [pulsos/us^2]
      amax = a_abs[motor];
    }
    else if(Dq21[motor] < Dq10[motor])
    {
      a_2d[motor][pto_i] = -(PPRad*0.5*a_abs[motor])/1e12;
      amax = -a_abs[motor];
    }


    // Calculo de tau con su formula. Si la amax=0, tau=0
    if(amax!=0)
    {
      // En valor absoluto porque es un tiempo
      tau = abs((T1*Dq21[motor] - T2*Dq10[motor])/(2*T1*T2*amax));
    }
    else
    {
      tau = 0.0;
    }


    // Guardamos los taus y T1s en [us]:
    taus[motor][pto_i] = tau*1e6;
    T1s[pto_i] = T1*1e6;


  }


  // Comprobación adicional: si los taus previo y actual son mayores que T1, es porque la vmax es excesiva (explicado en Prog 12)
  flag_repetir_pto = false;
  T1min = 0.0;
  T1min_mot = 0.0;
  if(pto_i >=1)
  {
    for(motor=0; motor<3; motor++)
    {
      // Comparamos los tiempos entre sí en [us], que es como los tenemos guardados
      if(T1s[pto_i] < (taus[motor][pto_i] + taus[motor][pto_i-1]))
      {
        flag_repetir_pto = true;
        T1min_mot = taus[motor][pto_i] + taus[motor][pto_i-1];
        if(T1min_mot > T1min)
        {
            T1min = T1min_mot;
        }

      }
    }
  }

  if(flag_repetir_pto)
  {
      // Convertimos el T1min a [s]
      T1min = T1min*1e-6;

      // Modificamos la vmax de este tramo para que quepan los dos taus dentro del T1
      vmax_destinos[pto_i] = abs(Dq10[motor_Dq10_max])/T1min;

      // Retrasamos en 1 el contador de puntos para que se repitan los cálculos para este punto y para el anterior
      pto_i = pto_i - 2;

  }
  // Desactivamos el flag de que hay que repetir los cálculos para el punto siguiente
  flag_repetir_pto = false;

 }


// Dado que estamos configurando un nuevo movimiento, establecemos además que el movimiento anterior de cada motor estaba finalizado
 for(motor=0; motor<3; motor++)
 {
  finalizado[motor] = true;
 }

}
```

```
////////////////////// run_motor ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Â·run_motor: función a ejecutar con la mayor frecuencia posible para cada motor (similar a la función run() de AccelStepper)
// que comprueba si corresponde dar un pulso a los motores, y recalcula el tiempo que debe transcurrir hasta el próximo pulso

bool run_motor(int motor)
{
 t_act = micros();
 t_dif = t_act − tini_tramo[motor];

 // Si el movimiento estaba finalizado, lo estamos comenzando, y reseteamos las variables
 // Cuando finalizado se vuelva a poner a true, esta función no será llamada para este motor hasta que se inicie un nuevo movimiento
 if(finalizado[motor])
 {
  finalizado[motor] = false;
  tini_tramo[motor] = t_act;
  t_dif=0;
  tlast[motor] = t_act;

  vact_pud[motor] = 0;    // La velocidad inicial es nula. El error se encargará de incrementarla
                          // Al usar una velocidad inicial nula, e imponerle un umbral para calcular dT, nos evitamos los problemas asociados a velocidades muy bajas
  dT_p[motor] = 0;        // Establecemos el dT a 0 para que se comiencen a dar pulsos inmediatamente

  pto_motor[motor] = 0;

 }




 // dT
 if(t_act−tlast[motor] >= dT_p[motor])
 {
  // Fase 1: tramo lineal: se realiza igual para todos los puntos, incluido el antepenúltimo
  // Si tdif<(T1−tau):
  if(t_dif < (T1s[pto_motor[motor]] − taus[motor][pto_motor[motor]]))
  {
   //Cálculo del e_p en el tramo de velocidad constante:
   e_p = PPRad*qdestinos[motor][pto_motor[motor]] + v1_pud[motor][pto_motor[motor]]*t_dif − q_p[motor];
   vact_pud[motor] =P*e_p;


   // Si la velocidad es suficientemente alta, aplicamos pulso y recalculamos dT
   if(vact_pud[motor]!=0)
   {
    // Detectamos si hay cambio de signo de la velocidad antes de dar el pulso
    if(vact_pud[motor]>0 && sentido_p[motor]<0)
    {
     sentido_p[motor] = 1;
     digitalWrite(DIR[motor], sentido_positivo[motor]);
    }
    else if(vact_pud[motor]<0 && sentido_p[motor]>0)
    {
     sentido_p[motor] = −1;
     digitalWrite(DIR[motor], !sentido_positivo[motor]);
    }

    // Damos el pulso
    digitalWrite(STEP[motor], HIGH);
    digitalWrite(STEP[motor], LOW);
    q_p[motor] += sentido_p[motor];
    tlast[motor] = t_act;

    // Calculamos el nuevo dT
    dT_p[motor] = 1/abs(vact_pud[motor]);

   }
   // Si vact_pud es demasiado pequeña (es nula), asignamos el dT asociado a la vmin, y no damos pulso
   else
   {
    tlast[motor] = t_act;
    dT_p[motor] = 1/vmin_pud;  // Valor alto de dT pero inferior al correspondiente a la vact
   }

  }


  // Fase 2: tramo curvo: tdif>=(T1−tau) y tdif < (T1+tau)
  else if(t_dif < (T1s[pto_motor[motor]] + taus[motor][pto_motor[motor]]))
  {
   // Cálculo del e_p en el tramo de aceleración:
   // qdes = q1 + v1*(tdif−T1) + a/2*(tdif−T1+tau)^2
   e_p = PPRad*qdestinos[motor][pto_motor[motor]+1] + v1_pud[motor][pto_motor[motor]]*(t_dif − T1s[pto_motor[motor]]) + a_2d[motor][pto_motor[motor]]*(t_dif − T1s[pto_motor[motor]] + taus[
         motor][pto_motor[motor]])*(t_dif − T1s[pto_motor[motor]] + taus[motor][pto_motor[motor]]) − q_p[motor];
   vact_pud[motor] =P*e_p;


   // Si la velocidad es suficientemente alta, aplicamos pulso y recalculamos dT
   if(vact_pud[motor]!=0)
   {
    // Detectamos si hay cambio de signo de la velocidad antes de dar el pulso
    if(vact_pud[motor]>0 && sentido_p[motor]<0)
    {
     sentido_p[motor] = 1;
     digitalWrite(DIR[motor], sentido_positivo[motor]);
    }
    else if(vact_pud[motor]<0 && sentido_p[motor]>0)
    {
     sentido_p[motor] = −1;
     digitalWrite(DIR[motor], !sentido_positivo[motor]);
    }

    // Damos el pulso
    digitalWrite(STEP[motor], HIGH);
    digitalWrite(STEP[motor], LOW);
    q_p[motor] += sentido_p[motor];
    tlast[motor] = t_act;
```

```
        // Calculamos el nuevo dT
        dT_p[motor] = 1/abs(vact_pud[motor]);

      }
      // Si vact_pud es demasiado pequeña (es nula), asignamos el dT asociado a la vmin, y no damos pulso
      else
      {
        tlast[motor] = t_act;
        dT_p[motor] = 1/vmin_pud;  // Valor alto de dT pero inferior al correspondiente a la vact
      }

    }



    // Fase 3: se diferencia según estemos en el antepenultimo punto o no
    // tdif >= (T1+tau)
    else
    {
      // Puntos previos al antepenultimo punto: cambiamos de tramo
      if(pto_motor[motor] < (npuntos-3))
      {
        // Hacemos dT = 0 para que el proximo tramo se evalue en cuanto sea posible
        dT_p[motor] = 0;

        // Actualizamos el valor de tini para el proximo tramo
        // Lo establecemos como el sintante en que se debió pasar por el punto q1 del punto actual q0, para que sea común a todos los motores
        tini_tramo[motor] = tini_tramo[motor] + T1s[pto_motor[motor]];

        // Incrementamos el contador de pto_motor
        pto_motor[motor]++;

      }

      // Tramo final: pto_motor[motor] == (npuntos-3)
      else  // Si estamos en el antepenúltimo punto, tras la aceleración -> comienza el Tramo final
      {
        // Imponemos que la posición deseada es el punto de destino final de este motor
        e_p = PPRad*qdestinos[motor][npuntos-1] - q_p[motor];
        vact_pud[motor] =P*e_p;


        // Si la velocidad es suficientemente alta, aplicamos pulso y recalculamos dT
        if(vact_pud[motor]!=0)
        {
          // Detectamos si hay cambio de signo de la velocidad antes de dar el pulso
          if(vact_pud[motor]>0 && sentido_p[motor]<0)
          {
            sentido_p[motor] = 1;
            digitalWrite(DIR[motor], sentido_positivo[motor]);
          }
          else if(vact_pud[motor]<0 && sentido_p[motor]>0)
          {
            sentido_p[motor] = -1;
            digitalWrite(DIR[motor], !sentido_positivo[motor]);
          }

          // Damos el pulso
          digitalWrite(STEP[motor], HIGH);
          digitalWrite(STEP[motor], LOW);
          q_p[motor] += sentido_p[motor];
          tlast[motor] = t_act;

          // Calculamos el nuevo dT
          dT_p[motor] = 1/abs(vact_pud[motor]);

        }
        // Si vact_pud es demasiado pequeña (es nula), es porque e_p=0, y eso es porque hemos alcanzado la posición de destino
        else
        {
          finalizado[motor] = true;
        }

      }
    }

  }  // Fin del dT

  // Devolvemos finalizado, que solo será true cuando se complete la última desaceleración
  return finalizado[motor];
}




////////////////////// almacena_q ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Ã·almacena_q():función que almacena en los vectores globales la información de coordenadas y velocidad, en [radianes] de los puntos para movimientos acumulativos

void almacena_q(float q_d1, float q_d2, float q_d3, float v_max)
{
  // Construimos progresivamente el vector de puntos de destino
  qdestinos[0][npuntos_acumulados] = q_d1;
  qdestinos[1][npuntos_acumulados] = q_d2;
  qdestinos[2][npuntos_acumulados] = q_d3;

  // Construimos progresivamente el vector de velocidades en la llegada a los puntos de destino
  vmax_destinos[npuntos_acumulados] = v_max;

  // Incrementamos el contador de puntos acumulados para el movimiento trapezoidal
  npuntos_acumulados++;
}




////////////////////// MOVEQp ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Ã·MOVEQp: función que recibe un punto de destino en coordenadas articulares
// y ejecuta el movimiento (punto a punto desde el punto actual) hasta llegar a él
// La función no se abandona hasta que se completa el movimiento, pero permite la ejecución
```

```
// simulánea de otras tareas recogidas en general()

void MOVEQp(float q_d1, float q_d2, float q_d3, float v_max)
{
 // Si existe algún movimiento acumulativo pendiente, debe ser ejecutado el primero
 if(moveq_pte || movej_pte)
 {
  ejecuta_moveq();
 }


 int motor;


 // Almacenamos el punto de destino y la velocidad, en [radianes]
 almacena_q(q_d1 *pi/180.0, q_d2 *pi/180.0, q_d3 *pi/180.0, v_max *pi/180.0); // Tras esto, npuntos_acumulados debe valer 1


 // Llamamos a la función para configurar los parámetros del movimiento
 prepara_parabolico();


 // Realizamos el movimiento solo si el punto en cuestión estaba dentro de los límites de q123
 if(npuntos_acumulados>0)
 {
  // Reseteamos las variables de movimiento completado:
  for(motor=0; motor<3; motor++)
  {
   completado_mot[motor] = false;
  }


  // Bucle hasta completar la ejecución del movimiento, permitiéndo la ejecución simultánea de "general()"
  while((!(completado_mot[0]*completado_mot[1]*completado_mot[2])) && flag_ejecuta_funcion)
  {
   // Llamamos a la función run() de los motores que no hayan llegado a su destino
   for(motor=0; motor<3; motor++)
   {
    if(!completado_mot[motor])
    {
     completado_mot[motor] = run_motor(motor);
     if(completado_mot[motor])
     {
      Serial.print("-MOVEQp: terminado motor ");
      Serial.println(motor);
     }
    }
   }

   // Llamamos a la función general
   general();

  }
  Serial.println("-MOVEQp: Movimiento finalizado");

  // Al salir, establecemos npuntos_acumulados a 0 para saber que no existen puntos pendientes
  npuntos_acumulados = 0;
 }
 else
 {
  Serial.println("-MOVEQp: NO hay movimiento que realizar");
 }
}




////////////////////// MOVEJp ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·MOVEJp: función que permite la realización de movimientos punto a punto, proporcionando dichos puntos en coordenadas cartesianas (xyz), y ejecutando el movimiento
// siguiendo la trayectoria más cómoda y directa para el robot

void MOVEJp(float x, float y, float z, float v_max)
{
 // Si existe algún movimiento acumulativo pendiente, debe ser ejecutado el primero
 if(moveq_pte || movej_pte)
 {
  ejecuta_moveq();
 }


 // Introducimos en el vector global la posición dada por el usuario
 xyz[0] = x;
 xyz[1] = y;
 xyz[2] = z;

 // Comprobamos si el punto queda dentro del espacio de trabajo con la funcion "dentro_ET()"
 if(dentro_ET()) // Si el punto esta dentro del ET
 {
  // Aplicamos CI al punto en cuestión
  CI();

  // Utilizamos los valores de q123 para realizar un movimiento MOVEQp
  Serial.println("-MOVEJp: iniciando movimiento mediante MOVEQp");

  // MOVEQp recibe los valores de q123 en [grados], y la vmax también en [grados], que es como nos ha llegado
  MOVEQp(q123[0]*180.0/pi, q123[1]*180.0/pi, q123[2]*180.0/pi, v_max);
 }
 else
 {
  Serial.println("-MOVEJp: el punto queda fuera del ET. NO hay movimiento que ejecutar");
 }

}




////////////////////// MOVEQ ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·MOVEQ: función que recibe un punto de destino en coordenadas articulares, y la vmax con la que el robot se debe mover a él
//  y lo almacena para realizar posteriormente un movimiento acumulativo con evolución trapezoidal por estos puntos, alcanzando de forma exacta solo el primero y el último
```

```
//  y pasando todo lo cerca posible de los demás de acuerdo con la aceleración máxima permitida en el robot
//  Permite, por tanto, generar un "movimiento acumulativo" donde el usuario añade con cada sentencia MOVEQ un nuevo por el que pasar

void MOVEQ(float q_d1, float q_d2, float q_d3, float v_max)
{
  // Si existe algún movimiento acumulativo pendiente, debe ser ejecutado el primero
  if(movej_pte)
  {
    ejecuta_moveq();
  }

  // Activamos el flag de que hay puntos pendientes para un movimiento MOVEQ
  moveq_pte = true;


  Serial.println("–MOVEQ: almacenando punto");

  // Almacenamos el punto, con coordenadas y velocidad en [radianes]
  almacena_q(q_d1 *pi/180.0, q_d2 *pi/180.0, q_d3 *pi/180.0, v_max *pi/180.0);
}




////////////////////// MOVEJ ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·MOVEJ: función que permite la acumulación de puntos proporcionando en coordenadas cartesianas (xyz), para posteriormente ejecutar un movimiento parabólico en coordenadas articulares
// pasando sin pausa por la proximidad de esos puntos, acercándose a ellos todo lo que permiten las aceleraciones de los ejes.

void MOVEJ(float x, float y, float z, float v_max)
{
  // Si existe algún movimiento acumulativo pendiente, debe ser ejecutado el primero
  if(moveq_pte)
  {
    ejecuta_moveq();
  }


  // Introducimos en el vector global la posición dada por el usuario
  xyz[0] = x;
  xyz[1] = y;
  xyz[2] = z;

  // Comprobamos si el punto queda dentro del espacio de trabajo con la funcion "dentro_ET()"
  if(dentro_ET())
  {
    // Activamos el flag de movimiento MOVEJ pendiente
    movej_pte = true;

    // Aplicamos CI al punto en cuestión
    CI();

    // Utilizamos los valores de q123 para realizar un movimiento MOVEQp
    Serial.println("–MOVEJ: almacenando punto");

  // Almacenamos el punto, con coordenadas y velocidad en [radianes]
    almacena_q(q123[0], q123[1], q123[2], v_max *pi/180.0);
  }
  else
  {
    Serial.println("–MOVEJ: el punto queda fuera del ET y NO será almacenado");
  }

}



////////////////////// fin_programa ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·fin_programa: función que es llamada tras finalizar la ejecución del código proporcionado por el usuario para, en caso de que las ultimas sentencias se correspondiesen
//  con funciones asociadas a un movimiento acumulativo (MOVEQ), éste sea ejecutado

void fin_programa(void)
{
  // Si existe algún movimiento acumulativo pendiente, debe ser ejecutado
  if(moveq_pte || movej_pte)
  {
    ejecuta_moveq();
  }
}




////////////////////// ejecuta_moveq ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·ejecuta_moveq: función que es llamada por otras funciones del programa cuando existe un conjunto de puntos acumulado por los que se debe pasar realizando un movimiento MOVEQ.
//  –Utiliza las variables globales en las que se han ido acumulando estos puntos
//  –La función no se abandona hasta que se completa el movimiento
//  –Al finalizar, resetea las variables para indicar que no existe movimiento acumulado

void ejecuta_moveq(void)
{
  int motor;

  // Llamamos a la función para configurar los parámetros del movimiento
  prepara_parabolico();


  // Realizamos el movimiento solo si hay puntos de los límites de q123
  if(npuntos_acumulados>0)
  {
    // Reseteamos las variables de movimiento completado:
    for(motor=0; motor<3; motor++)
    {
      completado_mot[motor] = false;
    }


    // Bucle hasta completar la ejecución del movimiento, permitiéndo la ejecución simultánea de "general()"
    while((!(completado_mot[0]*completado_mot[1]*completado_mot[2])) && flag_ejecuta_funcion)
    {
```

```
    // Llamamos a la función run() de los motores que no hayan llegado a su destino
    for(motor=0; motor<3; motor++)
    {
     if(!completado_mot[motor])
     {
      completado_mot[motor] = run_motor(motor);
      if(completado_mot[motor])
      {
       Serial.print("-ejecuta_moveq: terminado motor ");
       Serial.println(motor);
      }
     }
    }

    // Llamamos a la función general
    general();

   }
   Serial.println("-ejecuta_moveq: Movimiento finalizado");


   // Reseteamos las variables asociadas a los puntos acumulados
   npuntos_acumulados = 0;
   moveq_pte = false;
   movej_pte = false;

  }
  else
  {
   moveq_pte = false;
   movej_pte = false;
   Serial.println("-ejecuta_moveq: NO hay movimiento que realizar");
  }

 }




////////////////////// PAUSA ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·PAUSA: función para realizar una espera durante un determinado tiempo dado en segundos, permitiendo la ejecución simultánea de general()

void PAUSA(float duracion)
{
 // Si existe algún movimiento acumulativo pendiente, debe ser ejecutado primero
 if(moveq_pte || movej_pte)
 {
  ejecuta_moveq();
 }



 long tini_pausa = micros();
 t_act = tini_pausa;

 long t_duracion = duracion*1e6;


Serial.print("-PAUSA de duracion: ");
Serial.print(t_duracion);
Serial.println(" us");

 while(((t_act-tini_pausa)<t_duracion) && flag_ejecuta_funcion)
 {
  t_act = micros();

  general();
 }
}




////////////////////// HOME ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·HOME: función que llevar el Robot a su posición inicial conocida q123 = (0, 0, 0) haciendo uso de los finales de carrera
//  La función HOME está pensada para ser ejecutada al arrancar el robot, por lo que no atiende que pudieran existir puntos acumulados para un movimiento
//  La función sigue la estructura de MOVEQp, pero utilizando las señales de los FDC para finalizar los movimientos de cada motor

void HOME(void)
{
 // Activamos la variable de que estamos haciendo un HOME para poder exceder los limites de q123
 en_home = true;

 int motor, motor2;

 Serial.println("-HOME: comenzando HOME");
 PAUSA(0.5);

 // Utilizamos que las vairables de posición absoluta q_p[] tienen el valor 0 al arrancar el programa, y nos movemos con velocidades trapezoidales

 // Hacemos el HOME de los motores en el orden: 0, 1, 2
 for(motor=0; motor<3; motor++)
 {
 // Configuramos el primer punto de destino ficticio de cada motor: dar una vuelta completa en un sentido
  // Configuracion para el motor 0
  if(motor == 0)
  {
   qdestinos[0][0] = 360.0*pi/180.0;
   qdestinos[1][0] = RadPP*q_p[1]; // Para que no se mueva
   qdestinos[2][0] = RadPP*q_p[2]; // Para que no se mueva
  }

  // Configuracion para el motor 1
  if(motor == 1)
  {
   qdestinos[0][0] = RadPP*q_p[0]; // Para que no se mueva
   qdestinos[1][0] = -360.0*pi/180.0;
   qdestinos[2][0] = RadPP*q_p[2]; // Para que no se mueva
  }
```

```
  // Configuracion para el motor 2
  if(motor == 2)
  {
   qdestinos[0][0] = RadPP*q_p[0]; // Para que no se mueva
   qdestinos[1][0] = RadPP*q_p[1]; // Para que no se mueva
   qdestinos[2][0] = 360.0*pi/180.0;
  }



  // Movimiento rápido hasta tocar el FDC:
  vmax_destinos[0] = v_home_r;
  npuntos_acumulados = 1;
  prepara_parabolico();

  Serial.print("-HOME: movimiento rápido de aproximación al FDC del Motor ");
  Serial.println(motor);
  // Reseteamos las variables de movimiento completado:
  for(motor2=0; motor2<3; motor2++)
  {
   completado_mot[motor2] = false;
  }

  // Bucle hasta completar la ejecución del movimiento, permitiéndo la ejecución simultánea de "general()"
  while((digitalRead(FDC[motor]) != FDCA) && flag_ejecuta_funcion)
  {
   // Llamamos a la función run() de los motores que no hayan llegado a su destino
   for(motor2=0; motor2<3; motor2++)
   {
    if(!completado_mot[motor2])
    {
     completado_mot[motor2] = run_motor(motor2);
    }
   }
   // Llamamos a la función general
   general();
  }
  Serial.println("-HOME: movimiento rápido de aproximación completado");
  PAUSA(0.5);



 // Configuramos el alejamiento para realizar el posterior ajuste fino:
  // Configuracion para el motor 0
  if(motor == 0)
  {
   // Establecemos la posición actual del motor como 0 provisionalmente, para movernos relativametne a esta
   q_p[motor] = 0;
   qdestinos[0][0] = -alejamiento;
   qdestinos[1][0] = RadPP*q_p[1]; // Para que no se mueva
   qdestinos[2][0] = RadPP*q_p[2]; // Para que no se mueva
  }

  // Configuracion para el motor 1
  if(motor == 1)
  {
   q_p[motor] = 0;
   qdestinos[0][0] = RadPP*q_p[0]; // Para que no se mueva
   qdestinos[1][0] = alejamiento;
   qdestinos[2][0] = RadPP*q_p[2]; // Para que no se mueva
  }

  // Configuracion para el motor 2
  if(motor == 2)
  {
   q_p[motor] = 0;
   qdestinos[0][0] = RadPP*q_p[0]; // Para que no se mueva
   qdestinos[1][0] = RadPP*q_p[1]; // Para que no se mueva
   qdestinos[2][0] = -alejamiento;
  }


  // Movimiento de separación del FDC:
  vmax_destinos[0] = v_home_l;
  npuntos_acumulados = 1;
  prepara_parabolico();

  Serial.print("-HOME: movimiento de separación del FDC del Motor ");
  Serial.println(motor);
  // Reseteamos las variables de movimiento completado:
  for(motor2=0; motor2<3; motor2++)
  {
   completado_mot[motor2] = false;
  }

  // Bucle hasta completar la ejecución del movimiento, permitiéndo la ejecución simultánea de "general()"
  while((!(completado_mot[0]*completado_mot[1]*completado_mot[2])) && flag_ejecuta_funcion)
  {
   // Llamamos a la función run() de los motores que no hayan llegado a su destino
   for(motor2=0; motor2<3; motor2++)
   {
    if(!completado_mot[motor2])
    {
     completado_mot[motor2] = run_motor(motor2);
    }
   }
   // Llamamos a la función general
   general();
  }
  Serial.println("-HOME: movimiento de separación del FDC completado");
  PAUSA(0.5);



 // Configuramos el movimiento de aproximación para ajuste fino
  // Configuracion para el motor 0
  if(motor == 0)
  {
   qdestinos[0][0] = 360.0*pi/180.0;
   qdestinos[1][0] = RadPP*q_p[1]; // Para que no se mueva
   qdestinos[2][0] = RadPP*q_p[2]; // Para que no se mueva
  }
```

```
  // Configuracion para el motor 1
  if(motor == 1)
  {
   qdestinos[0][0] = RadPP*q_p[0]; // Para que no se mueva
   qdestinos[1][0] = -360.0*pi/180.0;
   qdestinos[2][0] = RadPP*q_p[2]; // Para que no se mueva
  }

  // Configuracion para el motor 2
  if(motor == 2)
  {
   qdestinos[0][0] = RadPP*q_p[0]; // Para que no se mueva
   qdestinos[1][0] = RadPP*q_p[1]; // Para que no se mueva
   qdestinos[2][0] = 360.0*pi/180.0;
  }

  vmax_destinos[0] = v_home_l;
  npuntos_acumulados = 1;
  prepara_parabolico();

  Serial.print("-HOME: movimiento de ajuste fino al FDC del Motor ");
  Serial.println(motor);
  // Reseteamos las variables de movimiento completado:
  for(motor2=0; motor2<3; motor2++)
  {
   completado_mot[motor2] = false;
  }

  // Bucle hasta completar la ejecución del movimiento, permitiéndo la ejecución simultánea de "general()"
  while((digitalRead(FDC[motor]) != FDCA) &&flag_ejecuta_funcion)
  {
   // Llamamos a la función run() de los motores que no hayan llegado a su destino
   for(motor2=0; motor2<3; motor2++)
   {
    if(!completado_mot[motor2])
    {
     completado_mot[motor2] = run_motor(motor2);
    }
   }
   // Llamamos a la función general
   general();
  }
  Serial.println("-HOME: movimiento de ajuste fino completado");
  PAUSA(0.5);
 }


 // Estándo tocando todos los FdC, guardamos la posición en la que se encuentra cada Motor:
 q_p[0] = (qreposo[0]+pi)*PPRad;
 q_p[1] = qreposo[1]*PPRad;
 q_p[2] = qreposo[2]*PPRad;


 // Configuramos el movimiento final a la posición (0,0,0)
 qdestinos[0][0] = qreposo[0];
 qdestinos[1][0] = qreposo[1];
 qdestinos[2][0] = qreposo[2];

 // Movimiento:
 vmax_destinos[0] = v_home_r;
 npuntos_acumulados = 1;
 prepara_parabolico();

 Serial.println("-HOME: movimiento final a la posición (0,0,0)");
 // Reseteamos las variables de movimiento completado:
 for(motor2=0; motor2<3; motor2++)
 {
  completado_mot[motor2] = false;
 }

 // Bucle hasta completar la ejecución del movimiento, permitiéndo la ejecución simultánea de "general()"
 while((!(completado_mot[0]*completado_mot[1]*completado_mot[2])) && flag_ejecuta_funcion)
 {
  // Llamamos a la función run() de los motores que no hayan llegado a su destino
  for(motor2=0; motor2<3; motor2++)
  {
   if(!completado_mot[motor2])
   {
    completado_mot[motor2] = run_motor(motor2);
   }
  }
  // Llamamos a la función general
  general();
 }
 OPEN(0);
 PAUSA(0.2);

 // Desactivamos la vairable de HOME
 en_home = false;
 npuntos_acumulados = 0;

 Serial.println("-HOME: Fin del HOME");
}




////////////////////// NO_HOME ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·NO_HOME: función para resetear los valores de q123 a su posición de reposo rapidamente cuando no se hace HOME al arrancar el Robot

void NO_HOME(void)
{
 // Si no se hace HOME, hay que hacer
 q_p[0] = qreposo[0]*PPRad;
 q_p[1] = qreposo[1]*PPRad;
 q_p[2] = qreposo[2]*PPRad;

 Serial.println("\n-NO_HOME.\n");
}
```

```
////////////////////// CD ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·CD: función que aplica la Cinemática Directa del Robot. Recoge los valores de q1, q2, q3 de un vector global q123, y almacena los de x,y,z en otro vector global xyz.

void CD(void)
{
  float q1, q2, q3;
  q1 = q123[0];
  q2 = q123[1];
  q3 = q123[2];

  // Ecuaciones de la Cinemática Directa
  xyz[0] = cos(q1)*( L2*sin(q2) − L36*cos(q2 + q3) + L45*sin(q2 + q3) );
  xyz[1] = sin(q1)*( L2*sin(q2) − L36*cos(q2 + q3) + L45*sin(q2 + q3) );
  xyz[2] = L1 +      L2*cos(q2) + L36*sin(q2 + q3) + L45*cos(q2 + q3);

}




////////////////////// CI ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·CI: función que aplica la Cinemática Inversa del Robot. Recoge los valores de x, y, z de un vector global xyz, y almacena los de q1,q2,q3 en otro vector global q123.

void CI(void)
{
  float r = sqrt(pow(xyz[0],2) + pow(xyz[1],2));
  float z = xyz[2];
  float q1, q2, q3;

  float A3;
  float A2, B2, gamma2, K2;

  // Cálculo de q1
  q1 = atan2(xyz[1], xyz[0]);


  // Cálculo de q3
  // gamma3 = atan2(L36, L45); (cte)
  A3 = (pow(r,2) + pow((z−L1),2) − pow(L2,2) − pow(L36,2) − pow(L45,2)) / (2*L2*sqrt(pow(L36,2) + pow(L45,2)));

  // Arreglo para evitar complejos
  if(A3>1)
  {
   Serial.print("A3>1 en CI"); Serial.println(A3);
   A3 = 1.0;
  }
  else if(A3<−1)
  {
   Serial.print("A3<−1 en CI"); Serial.println(A3);
   A3 = −1.0;
  }
  // q3 = gamma3 + acos(A3);
  q3 = atan2(L36, L45) + acos(A3);



  // Cálculo de q2
  A2 = L2 + L36*sin(q3) + L45*cos(q3);
  B2 = L36*cos(q3) − L45*sin(q3);
  gamma2 = atan2(B2, A2);
  K2 = (z−L1)/(sqrt(pow(A2,2) + pow(B2,2)));

  // Arreglo para evitar complejos
  if(K2>1)
  {
   Serial.print("K2>1 en CI"); Serial.println(K2);
   K2 = 1.0;
  }
  else if(K2<−1)
  {
   Serial.print("K2<−1 en CI"); Serial.println(K2);
   K2 = −1.0;
  }
  q2 = gamma2 + acos(K2);

  q123[0] = q1;
  q123[1] = q2;
  q123[2] = q3;

}




////////////////////// MOVELp ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Â·MOVELp: función que permite realizar movimientos punto a punto proporcionando el destino en coordenadas cartesianas xyz

void MOVELp(float xdes, float ydes, float zdes, float vmax)
{
  // Si existe algún movimiento acumulativo pendiente, debe ser ejecutado el primero
  if(moveq_pte || movej_pte)
  {
   ejecuta_moveq();
  }


  // Leemos la posición actual y la guardamos en el vector global q123
  q123[0] = RadPP*q_p[0];
  q123[1] = RadPP*q_p[1];
  q123[2] = RadPP*q_p[2];

  // Obtenemos la CD de la posición actual, que será guardada en el vector global xyz
```

```
  CD();

  // Calculamos la distancia con signo a recorrer a lo largo de cada eje cartesiano
  dist[0] = xdes − xyz[0];
  dist[1] = ydes − xyz[1];
  dist[2] = zdes − xyz[2];
  // Distancia total a recorrer
  dist_tot = sqrt( pow(dist[0],2) + pow(dist[1],2) + pow(dist[2],2) );


  // Dividimos la distancia de cada eje cartesiano en tramos de tamaño "paso_inter"
  // Si el número de puntos que resulta es mayor que el límite, imponemos el límite
  // Hay que tener en cuenta que deberán entrar posteriormente en el cómputo el punto final dos veces y el punto actual
  // La introducción del punto actual no es necesaria, ya que ya la realizarán las funciones posteriores
  // A la cantidad de puntos que aquí se generen, el algoritmo añadirá 3 más
  npuntos_inter = dist_tot/paso_inter;

  if(npuntos_inter > (NPTOS_DIV−4))
  {
   npuntos_inter = NPTOS_DIV−4;
  }

  // Obtenemos el "paso de interpolación" con signo entre puntos para cada eje
  paso_xyz[0] = dist[0]/npuntos_inter;
  paso_xyz[1] = dist[1]/npuntos_inter;
  paso_xyz[2] = dist[2]/npuntos_inter;


  // Buscamos el eje cartesiano de mayor recorrido
  eje_mayor = 0;
  xyz_mayor = xdes;
  if(abs(dist[1]) > abs(dist[eje_mayor]))
  {
   eje_mayor = 1;
   xyz_mayor = ydes;
  }
  if(abs(dist[2]) > abs(dist[eje_mayor]))
  {
   eje_mayor = 2;
   xyz_mayor = zdes;
  }




  // Se recorren los puntos, calculando su CI y guardando los valores de los q123 asociados en adestinos
  // La condición del while se aplica solo al eje cartesiano de mayor recorrido para asegurar que se entre en el while cuando haya ejes que no se desplzan
  while( abs(xyz_mayor−xyz[eje_mayor]) > abs(paso_xyz[eje_mayor]) )
  {
   xyz[0] = xyz[0] + paso_xyz[0];
   xyz[1] = xyz[1] + paso_xyz[1];
   xyz[2] = xyz[2] + paso_xyz[2];

    // Pasamos solo por los puntos de la linea recta que estén dentro del espacio de trabajo
    if(dentro_ET())
    {
     // Aplicamos CI al nuevo punto
     CI();

     // Guardamos el punto obtenido como punto de destino a alcanzar
     qdestinos[0][npuntos_acumulados] = q123[0];
     qdestinos[1][npuntos_acumulados] = q123[1];
     qdestinos[2][npuntos_acumulados] = q123[2];
     vmax_destinos[npuntos_acumulados] = vmax∗pi/180.0;

     npuntos_acumulados ++;
    }

  }


  // Introducimos el punto final de destino para alcanzarlo de forma exacta
  xyz[0] = xdes;
  xyz[1] = ydes;
  xyz[2] = zdes;
  CI();

  qdestinos[0][npuntos_acumulados] = q123[0];
  qdestinos[1][npuntos_acumulados] = q123[1];
  qdestinos[2][npuntos_acumulados] = q123[2];
  vmax_destinos[npuntos_acumulados] = vmax∗pi/180.0;

  npuntos_acumulados ++;



  // Llamamos a "ejecuta_moveq" para ejecutar el movimiento almacenado
  Serial.print("−MOVELp: movimiento de "); Serial.print(npuntos_inter); Serial.print(" puntos de interpolación. ");
  Serial.println("Llamando a ejecuta_moveq()");
  ejecuta_moveq();

 }







 ////////////////////// dentro_ET ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
 // Â·dentro_ET: función que comprueba si un punto dado en coordenadas (x,y,z) está dentro del Espacio de Trabajo del Robot

 bool dentro_ET(void)
 {
  bool respuesta = false;

  float r = sqrt(pow(xyz[0],2) + pow(xyz[1],2));
  float z = xyz[2];

  float Rfue2, Cr_fue, Cz_fue;
```

```
  float Rden2, Cr_den, Cz_den;

  if(z>=0 && z < zmax)
  {

    if (z <= ( L1 − sqrt(pow((L45 − L2),2) + pow(L36,2) − pow(colision,2)) ))
    {
     Rden2 = pow(L45,2) + pow(L36,2);                  // cian: C = [L2 L1] dentro
     Cr_den = L2;
     Cz_den = L1;
     // r >= colision
     if((pow((r−Cr_den),2) + pow((z−Cz_den),2) < Rden2) && (r>=colision))
     {
       respuesta = true;
     }
    }
    else if (z <= L1)
    {
     Rfue2 = pow((L45 − L2),2) + pow(L36,2);           // negra: C = [0 L1] fuera
     Cr_fue = 0;
     Cz_fue = L1;

     Rden2 = pow(L45,2) + pow(L36,2);                  // cian: C = [L2 L1] dentro
     Cr_den = L2;
     Cz_den = L1;

     if( (pow((r−Cr_fue),2) + pow((z−Cz_fue),2) >= Rfue2) && (pow((r−Cr_den),2) + pow((z−Cz_den),2) < Rden2) )
     {
       respuesta = true;
     }
    }
    else if (z <= L1 + L36)
    {
     Rfue2 = pow((L45 − L2),2) + pow(L36,2);           // negra: C = [0 L1] fuera
     Cr_fue = 0;
     Cz_fue = L1;

     Rden2 = pow((L2 + sqrt(pow(L45,2) + pow(L36,2))),2);    // roja: C = [0 L1] dentro
     Cr_den = 0;
     Cz_den = L1;

     if( (pow((r−Cr_fue),2) + pow((z−Cz_fue),2) >= Rfue2) && (pow((r−Cr_den),2) + pow((z−Cz_den),2) < Rden2) )
     {
       respuesta = true;
     }
    }
    else // z < zmax
    {
     Rfue2 = pow(L45,2) + pow(L36,2);                  // azul: C = [−L2 L1] fuera
     Cr_fue = −L2;
     Cz_fue = L1;

     Rden2 = pow((L2 + sqrt(pow(L45,2) + pow(L36,2))),2);    // roja: C = [0 L1] dentro
     Cr_den = 0;
     Cz_den = L1;

     if( (pow((r−Cr_fue),2) + pow((z−Cz_fue),2) >= Rfue2) && (pow((r−Cr_den),2) + pow((z−Cz_den),2) < Rden2) )
     {
       respuesta = true;
     }

    }

  } // Fin del (z>=0 && z < zmax)

  // Excepción para que los valores que se obtienen de calcular la CI a la posición de reposo (que pierden decimales) sí pertenezcan a la CI:
  float xyz_aux[3];
  if(!respuesta)
  {
   if(q_p[0]==qreposo[0]*PPRad && q_p[1]==qreposo[1]*PPRad && q_p[2]==qreposo[2]*PPRad)
   {
     respuesta = true;
     //Serial.println("−dentro_ET: Se ha aplicado la excepción para la posición de reposo");
   }
  }

  if(xyz[0]==0.06189999103546142578125 && z==0.2043499708175659179687 5)
  {
   respuesta = true;
   //Serial.println("−dentro_ET: Se ha aplicado la excepción");
  }

  if(!respuesta)
  {
   Serial.println("−dentro_ET: El punto queda fuera del espacio de trabajo");
  }

  return respuesta;

 }



 ////////////////////// dentro_q123 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
 // Ã-dentro_q123: función que comprueba si un punto dado en coordenadas (q1,q2,q3) es alcanzable por el Robot

 bool dentro_q123(void)
 {
  bool respuesta=false;

  // Comprobamos si q123 esta dentro de los limites. Durante el HOME se hace excepcion
  if( en_home || (q123[0]<=q123max[0] && q123[0]>=q123min[0] && q123[1]<=q123max[1] && q123[1]>=q123min[1] && q123[2]<=q123max[2] && q123[2]>=q123min[2]) )
  {
   respuesta = true;
  }
  if(!respuesta)
  {
   Serial.println("−dentro_q123: el punto queda fuera del rango de q123");
  }
  return respuesta;
 }
```

Apéndice B
Presupuesto

En este Apéndice se presenta el presupuesto mínimo de construcción de una unidad de este robot con las piezas concretas que monta en su versión definitiva.

Objeto	**€/ud**	**uds**		**€ tot**
Plástico de impresora 3D	0,0139 €	471		6,54 €
Fuente 12V	20,98 €	1		20,98 €
Arduino Mega 2560 Pro	15,99 €	1		15,99 €
Motores Nema 17	6,99 €	3		20,96 €
CNC Shield V3 y 4x Drv 8825	6,16 €	1		6,16 €
Correas	3,00 €	3		8,99 €
Poleas GT2	0,50 €	3		1,50 €
Finales de Carrera	1,23 €	3		3,70 €
Rodamientos 623ZZ	1,20 €	8		9,58 €
Rodamientos 608ZZ	1,00 €	2		2,00 €
Varillas acero 15mm 3mm diam	0,63 €	2		1,27 €
Resistencias 220ohm	0,04 €	3		0,12 €
Joysticks PS2	2,00 €	2		4,00 €
Leds	0,02 €	3		0,05 €
Interruptores 2.54mm	0,11 €	2		0,21 €
Botones 6mmx6mm	0,06 €	2		0,13 €
Servo SG90 9G	1,60 €	1		1,60 €
Escudo JLCPCB envio estandar	1,70 €	1		1,70 €
Pines Male y Female 2.54mm	0,00 €	130		0,54 €
Pines Female Angle 2.54mm	0,04 €	10		0,45 €
Pines Female Estra largos 2.54mm	0,07 €	28		2,07 €
Tornillo M3 10mm	0,02 €	31		0,62 €
Tornillo M3 20mm	0,03 €	6		0,18 €
Tornillo M3 30mm	0,04 €	2		0,08 €
Tuerca M3	0,01 €	27		0,27 €
Arandelas M3	0,01 €	4		0,05 €
Interruptor 220V	0,26 €	1		0,26 €
Tablero de madera	3,95 €	1		3,95 €
Total				**113,94 €**

Figura B.1 Presupuesto mínimo de construcción de un robot.

El cálculo de importe total presentado en este presupuesto no se corresponde, sin embargo, con la inversión real que ha sido necesaria para obtener el sistema presentado en este proyecto desde la fase de concepción de la idea. En la tabla aquí presentada se tienen en cuenta los gastos asociados a la adquisición de las cantidades concretas de cada objeto que se utilizan en la construcción de un robot, pero en el caso del proceso completo desde las fases iniciales de prototipado se deben tener en cuenta los siguientes comentarios:

- El PLA utilizado en la impresión 3D es un material muy barato, pero que para grandes cantidades de plástico no debe ser despreciado. Aquí se refleja la cantidad exacta de material necesaria para la impresión de las piezas definitivas del robot, pero en la fase de prototipado se ha requerido de imprimir varias versiones parciales o totales de cada pieza hasta obtener la definitiva.
- Los cables, protoboard y otros elementos utilizados durante la fase de prototipado no han sido incluidos en el presupuesto de construcción del robot por no ser necesarios una vez se tiene completado el proyecto, pero son evidentemente necesarios en las fases previas al diseño definitivo.
- Para elementos pequeños como botones e interruptores, el precio unitario es bajo, pero siempre es necesario comprarlos en grandes cantidades.
- En elementos como las correas, se puede ahorrar mucho comprándolas en páginas como *Aliexpress* en lugar de *Amazon*, siempre y cuando no se trabaje con horizontes temporales exigentes y se puedan asumir tiempos de entrega largos.
- El coste de adquisición de los innumerables elementos de bricolaje, fundamentalmente herramientas, que se han utilizado para el montaje del robot, y de las que el alumno ya disponía antes del inicio de este proyecto, no han sido incluidas en la tabla.

Teniendo en cuenta lo anterior, se pueden extraerse las siguientes conclusiones:

- El precio de un robot completo para el caso en que se obtuvieran todos los elementos al menor precio unitario disponible, de acuerdo con los datos de tiendas online habituales, podría llegar a reducirse por debajo de los 100€; pero para que esos precios unitarios sean aplicables deberían producirse en torno a un centenar de unidades del robot.
- El precio real de la construcción de un solo robot, teniendo en cuenta la obligatoriedad de la compra de muchos de los elementos de coste bajo en cantidades superiores a las estrictamente necesarias, superaría probablemente los 130€.
- El coste total de la realización de este proyecto, si se tienen en cuenta la fase de prototipado completa, con la impresión de numerosas piezas de prueba, la compra o construcción de elementos que terminan no resultando útiles en el robot final, y la aplicación en ocasiones de gastos de envío superiores a la opción más básica con el fin de no paralizar el proyecto, se encuentra en torno a los 250€.

Bibliografía

[1] J. Domínguez, M. Acosta, *Teoría de Máquinas y Mecanismos*, Ed. Universidad de Sevilla, 2015.

[2] A. Barrientos, *Fundamentos de Robótica*, 2ª ed., Ed. McGraw-Hill Interamericana de España S.L., 2007.

[3] A. Barrientos, *Trasnparencias de la asignatura Fundamentos de Robótica*

[4] PCB TRACE WIDTH CALCULATOR, mclpcb.com, "*https://www.mclpcb.com/blog/pcb-trace-width-vs-current-table/*", (accessed Aug 2021).

[5] Axel's PCB Via Diameter Calculator, google.com/spreadsheets, "*https://docs.google.com/spreadsheets/d/1n8DW1zGfy29GRw2g0gTAG2-SsfYFg5FQo5QQkT9V43I/edit#gid=1988600577*", (accessed Aug 2021).

[6] ABB, *Manual de referencia técnica. Instrucciones, funciones y tipos de datos de RAPID*

[7] Libraries reference, arduino.cc, "*https://www.arduino.cc/reference/en/libraries/*", (accessed Aug 2021).

[8] J. Drake, *Fundamentos Físicos de la Ingeniería Mecánica*, Ed. Universidad de Sevilla, 2016.

[9] Calibrar driver DRV8825 - Voltaje de referencia - CNC shield, youtube.com, "*https://www.youtube.com/watch?v=v74PWb6E6No*", (accessed Aug 2021).

[10] DRV8825 Controlador de motor a pasos para Arduino, hetpro-store.com, "*https://hetpro-store.com/TUTORIALES/controlador-de-motor-pasos-drv8825-con-arduino/*", (accessed Aug 2021).

[11] Tutorial EasyEDA Completo + Crear Componente, youtube.com, "*https://www.youtube.com/watch?v=BvvHJ-H79l8&t=69s*", (accessed Aug 2021).

[12] Arduino Mega 2560 Rev3, arduino.cc, "*https://store.arduino.cc/products/arduino-mega-2560-rev3*", (accessed Aug 2021).