Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Brazo robot de bajo coste con motores paso a paso que escribe

Autor: Juan Ignacio Medrano Trujillo
Tutor: Ignacio Alvarado Aldea

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

Sevilla, 2020

Ingeniería de Sistemas y Automática

Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Brazo robot de bajo coste con motores paso a paso que escribe

Autor:
Juan Ignacio Medrano Trujillo

Tutor:
Ignacio Alvarado Aldea
Profesor titular

Dpto. Ingeniería de Sistemas y Automáticas
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo Fin de Grado: Brazo robot de bajo coste con motores paso a paso que escribe

Autor: Juan Ignacio Medrano Trujillo

Tutor: Ignacio Alvarado Aldea

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

-

Somos nuestro propio motor de mejora. Y para una nueva actualizción antes hay que buscar y encontrar los errores

- Pilar Alonso -

A mi familia

A mis amigos

A mis profesores

Agradecimientos

Me gustaría agradecer la ayuda aportada por el que ha sido mi tutor durante la elaboración del Trabajo Fin de Grado. Ignacio, muchas gracias por todo el empeño que has tenido en que aprendiese e hiciera las cosas bien, por la confianza y la paciencia.

Mostrar mi gratitud a todo aquel que ha estado conmigo este tiempo, con cada detalle aprendia una cosa nueva y me servía para crecer.

Muchas gracias a mi familia por el apoyo que me han aportado siempre que lo he necesitado. Gracias a mis padres Inés y Juan que han tenido la paciencia y perseverancia que hoy me permiten estar aquí a las puertas de terminar una etapa importante de mi vida. Reconocimiento especial a mi padre por su inestimable ayuda, en muchas ocasiones cuando yo perdía la ilusión o las ganas cuando no salían las cosas bien, siempre estaba para aportar y orientarme en buscar nuevas soluciones, muchas gracias padre.

Gracias.

Juan Ignacio Medrano Trujillo

Sevilla, enero de 2020

Resumen

El trabajo presentado como fin de grado es un brazo articulado con tres grados de libertad y un efector final fijo en el que se le ha encomendado la tarea de escribir.

El movimiento es aplicado por tres motores paso a paso -Nema17- que controlados mediante el microprocesador de Arduino Mega y sus respectivos drivers pondremos en funcionamiento. Estarán limitados en movimiento por sensores fin de carrera.

En este proyecto se ha considerado que la estructura del brazo será en madera MDF de tres milímetros de grosor cuyo previo diseño se ha realizado en un software gráfico como es CorelDraw. Para los mecanismos se ha usado metacrilato de tres milímetros para los engranajes y de cinco milímetros para los piñones ya que son piezas que estarán en movimiento y la fricción será alta.

El proyecto que se presenta, consiste en la construcción y programacón de un brazo robótico en madera para su uso didáctico dirigidos a alumnos de ingeniería o aquellos que sientan curiosidad por este proyecto, ya que uno de los objetivos es hacerlo accesible a todos, de fácil comprensión ante estos nuevos conocimientos.

Lo primero que haremos será explicar el modelo usado para el robot y su construcción, las piezas usadas, la electrónica y los materiales, así como las medidas aproximadas de cada una de ellas, ya que serán esenciales en la programación. Por último, se hablará del lenguaje que usaremos para programarlo, así como del código que se ha usado para su funcionamiento.

Abstract

This work in order of degree talks about a robotic 3 degree of freedom arm with a fix end-effector handle to write.

Motion is due to 3 stepper motors -Nema17- controlled by a microprocessor from Arduino Mega and drivers respectively. Also, we will use 3 endstop sensors to limit the movement.

In this project it has been considered that the structure of the arm will be in wood three millimeters thick, designed in a graphic software named CorelDraw. For the gears we will use methacrylate of three and five millimeters because of the high friction.

The project is based on the construction and programming of a robotic arm in wood for teaching uses leaded to engineering students or those who are curious about this project, since one of the objectives is to make it accessible to everyone, easily comprenhension.

First at all, we will explain the model used for the robot and its construction, the parts used, the electronics and the materials. Finally, we will talk about the language we will use to program it, as well as the code that has been used for its operation.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

A^*	Conjugado
Π	Número pi = 3.1416
sen	Función seno
tg	Función tangente
arctg	Función arco tangente
cos	Función coseno
$\partial y\ \partial x$	Derivada parcial de *y* respecto
x^o	Notación de grado, *x* grados.
$f_x(q_n,\ldots)$	En función de de q_n
$<$	Menor o igual
$>$	Mayor o igual
GDL	Grado de libertad
$\Leftrightarrow$	Si y sólo si

1 INTRODUCCIÓN

Los robots no nos matarán, pero para salvarnos necesitamos ética y protección de la información

- Bruno Siciliano -

El origen etimológico de la palabra róbotica, lo encontramos ni mas ni menos que en el checo con la palabra *Robota* que significa "trabajo forzado, servidumbre".

1.1 Robótica

La robótica es la ciencia y la técnica que está involucrada en el diseño, la fabricación y la utilización de robots. Un robot es, por otra parte, una máquina que puede programarse para que interactúe con objetos y lograr que realice el comportamiento humano o animal.

La informática, la electrónica, la mecánica y la ingeniería son sólo algunas de las disciplinas que se combinan en la robótica. El objetivo principal de la robótica es la construcción de dispositivos que funcionen de manera automática y que realicen trabajos dificultosos o imposibles para los seres humanos.

Se pueden clasificar:

- *Según su cronología:*

  ∴ **1.ª Generación:** robots manipuladores. Son sistemas mecánicos multifuncionales con un sencillo sistema de control, bien manual, de secuencia fija o de secuencia variable.

  ∴ **2.ª Generación:** robots de aprendizaje. Repiten una secuencia de movimientos que ha sido ejecutada previamente por un operador humano. El modo de hacerlo es a través de un dispositivo mecánico. El operador realiza los movimientos requeridos mientras el robot le sigue y los memoriza.

  ∴ **3.ª Generación:** robots con control sensorizado. El controlador es un ordenador que ejecuta las órdenes de un programa y las envía al manipulador o robot para que realice los movimientos necesarios.

  ∴ **4.ª Generación:** robots inteligentes. Son similares a los anteriores, pero además poseen sensores que envían información a la computadora de control sobre el estado proceso. Esto permite una toma inteligente de decisiones y el control del proceso en tiempo real.

∴ **5.ª Generación:** directamente relacionadas con la inteligencia artificial. El alto desarrollo tecnológico que supone crear una nueva generación de robots. Hoy día vemos ejemplos de robots autónomos y capacitados para atender las necesidades de los humanos. La investigación tecnológica trabaja para crear robots capaces de imitar las funciones cognitivas de la mente humana.

Fig. 1-1: Pierna robótica

Fig. 1-2: Mano robótica Shadow

Fig. 1-3: Sophia. Robot humanoide

- *Según su estructura:*
  ∴ **Poliarticulados:** en este grupo se encuentran los robots manipuladores, los robots industriales y los robots cartesianos, que se emplean cuando es preciso abarcar una zona de trabajo relativamente amplia o alargada.
  ∴ **Móviles:** robots de gran capacidad de desplazamiento, basados en carros o plataformas y dotados de un sistema rodante motor.
  ∴ **Androides:** tipo de robot que intentan reproducir total o parcialmente la forma y el compartimento cinemático del ser humano.
  ∴ **Zoomórficos:** constituyen una clase caracterizada principalmente por sus sistemas de locomoción que imitan a los diversos seres vivos.
  ∴ **Híbridos:** aquellos de difícil clasificación, estructura combinada con alguna de las anteriores ya expuestas.

Fig. 1-4: Robot poliarticulado

Fig. 1-5: Robot móvil

Fig. 1-7: Robot zoomórfico

Fig. 1-6: Robot androide Nao

Este proyecto se ajusta a la estructura poliarticulada. Se estima que hoy día existen millones de robots industriales trabajando en todo el mundo. En general estas máquinas se limitan a realizar tareas pesadas y de precisión, una función programada que, eso sí, llevan a cabo más eficazmente que cualquier operario humano.

Primero, y de acuerdo con la Asociación de Industrias de Robótica (RIA: Robotic Industry Association), un robot industrial es «un manipulador multifuncional reprogramable, capaz de mover materiales, piezas, herramientas, o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas».

Tiene tres distintivos esenciales: es multifuncional, puede ser programado por un operador humano o un dispositivo lógico y es reprogramable.

Fig. 1-8: Robot manipulador

Además de estas características que definen a los robots industriales, también cabe destacar la composición del robot, que se divide en elementos estructurales rígidos, llamados eslabones o enlaces. Estos son conectados por articulaciones que pueden ser lineales o rotatorias. Tiene un efector final el cual "especializa" la aplicación del manipulador.

1.2 Justificación del Proyecto

Es importante justificar la realización de este proyecto ya que solo no aborda aspectos teóricos sobre robótica, sino que además considera un fuerte punto el desarrollo práctico, llevando a cabo un diseño de un prototipo de brazo articulado.

Con motivo de hacer más accesible la posibilidad de tratar con un prototipo de brazo robótico y que sirva para la práctica asemejando con algunos que podríamos encontrar en la mayoría de empresas y fábricas técnicas (más avanzados) y convertirlo en una tarea lúdica cercana a todo aquel que sienta curiosidad.

Además, plantea ser una herramienta motivadora para ayudar al alumnado a asentar los conocimientos alcanzados durante el Grado.

1.3 Objetivos

Este Trabajo Fin de Grado se centra en enseñar a diseñar, construir y programar un brazo articulado de tres grados de libertad, facilitando esquemas de todo el proceso de manera visual y cómoda para el lector.

Objetivo principal: funcionalidad

El objetivo principal es crear dos estados de movimiento del brazo:

- *Manual:* mediante joysticks se controla el ángulo de cada articulación de manera independiente. Además, controlar las articulaciones con movimiento en los ejes.
- *Automático:* desde el puerto serie se manda orden de una coordenada deseada para ser alcanzada de manera autónoma por el brazo.

Objetivos secundarios:

- Hacer automáticamente que siga una trayectoria.

- Conseguir que escriba.

Una vez se cumpla el objetivo principal, mientras más objetivo secundarios se cumplan, más valor tendrá el diseño final que se haga.

En resumen, lo que se pretende conseguir es una herramienta factible, económica y didáctica.

2 ROBOTS MANIPULADORES

En este capítulo se comentará algunos tipos de brazos robóticos que existen en el mercado actual y se definirá en qué grupo se incluye el tipo de brazo que se va a desarrollar.

2.1 Clasificación

Existen multitud de tipos y modelos que se clasifican principalmente en seis grupos, ampliamente diferenciados por el movimiento y el espacio de trabajo de cada uno.

- **Robot cartesiano:** robot cuyo brazo tiene tres articulaciones prismáticas, cuyos ejes son coincidentes con los ejes cartesianos. Son usados en trabajos de 'pick & place', operaciones de ensamblado, manipulación de máquinas herramientas y soldaduras por arco.

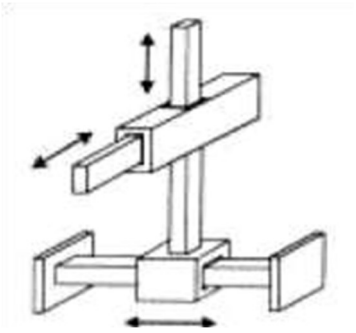


Fig. 2-1: Ejemplo cartesiano

- **Robot cilíndrico:** robot cuyos ejes forman un sistema de coordenadas cilíndricas. Empleado en operaciones de ensamblaje, manipulación de máquinas herramientas, soldadura por punto y manipulación en máquinas de fundición a presión.

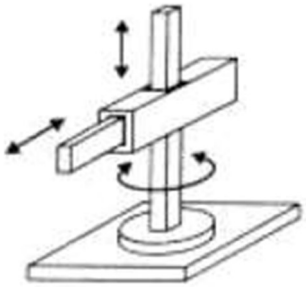


Fig. 2-2: Ejemplo cilíndrico

- **Robot esférico/polar:** robot cuyos ejes forman un sistema polar de coordenadas. Utilizado en la manipulación de máquinas herramientas, soldadura por punto, fundición a presión, máquinas de desbarbado, soldadura por gas y por arco.

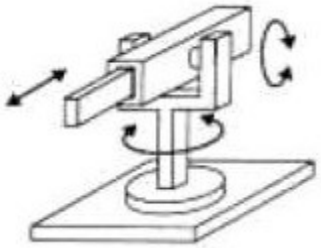


Fig. 2-3: Ejemplo esférico

- **Robot SCARA:** robot que tiene dos articulaciones rotatorias paralelas para proporcionar elasticidad en un plano. Empleados en trabajos de 'pick & place', aplicación de impermeabilizantes, operaciones de ensamblado y manipulación de máquinas herramientas.

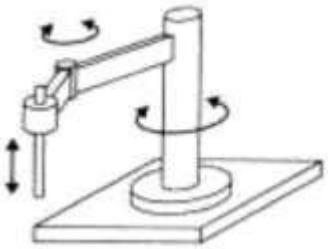


Fig. 2-4: Ejemplo SCARA

- **Robot articulado:** robot cuyo brazo tiene como mínimo tres articulaciones rotatorias. Utilizado para operaciones de ensamblaje, fundición a presión, máquinas de desbarbado, soldadura a gas, soldadura por arco y pintado por spray.

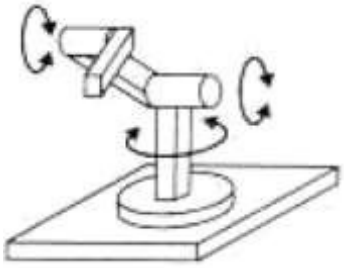


Fig. 2-5: Ejemplo articulado

- **Robot paralelo:** robot cuyos brazos tiene articulaciones prismáticas o rotatorias concurrentes. Uno de sus usos es la plataforma móvil que manipula las cabinas de los simuladores de vuelo.

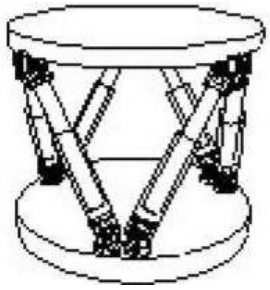


Fig. 2-6: Ejemplo paralelo

Como se observa, las funciones finales de cada tipo se asemejan, esto nos hace indicar que un punto a tener en cuenta para elegir el tipo de robot que debemos usar en la tarea que vayamos a emprender va a depender del espacio de trabajo del robot, el tamaño y la tarea a desempeñar.

En resumen, el robot de este proyecto puede estar ubicado en el mercado como un modelo más, dentro del Robot Articulado. Simple, de tamaño y costes reducido.

3 FUNDAMENTOS TEÓRICOS

3.1 Introducción

En este capítulo explicaremos los pasos que hemos seguido para entender el movimiento del brazo articulado, siendo la cinemática, la parte teórica fundamental en la que nos basaremos para el desarrollo del trabajo, incluyendo ejemplos dirigidos directamente a nuestro brazo en cuestión.

3.1.1 Cinemática

Hay que dejar claro que la base de este proyecto será definir el movimiento del brazo articulado mediante la teoría que habla sobre el modelo cinemático del robot.

La cinemática de un robot es el estudio de la posición, velocidad y aceleración de cada uno de los elementos del robot siendo calculados sin considerar las fuerzas que originan el movimiento, la dinámica del robot, que no veremos. La velocidad se determina como la derivada de la posición respecto al tiempo y la aceleración como la derivada de la velocidad respecto al tiempo.

La principal herramienta que usaremos para el análisis del movimiento será resolver el problema de cinemática del brazo articulado que, como veremos, se puede resolver mediante el modelo cinemático directo y el modelo cinemático inverso. Además, hay distintos métodos para resolver los modelos cinemáticos: geométrico, por matrices de transformación entre sistemas de referencias o transformación de coordenadas. Dependiendo del número de grados de libertad y si se usa el modelo directo o inverso, variará la dificultad y con ella la elección del método a utilizar.

La cinemática directa es el problema geométrico estático de calcular la posición y orientación del efector final del manipulador. Se usa para calcular la posición de partes de una estructura articulada a partir de sus componentes fijas y las transformaciones inducidas por las articulaciones de la estructura. El proceso inverso que calcula el conjunto de parámetros a partir de una posición específica del actuador final es la cinemática inversa.

3.2 Modelo directo e inverso aplicado al brazo

Como ya hemos dicho, la cinemática del robot estudia el movimiento del mismo con respecto a un sistema de referencia sin considerar las fuerzas que interviene. Se interesa por la relación entre la posición y orientación del extremo final del robot con los valores que toman sus coordenadas articulares. La formulación de los modelos cinemáticos adecuados es crucial para analizar el comportamiento de manipuladores industriales.

La cinemática inversa es un problema mucho más difícil que la cinemática directa. La solución del problema inversa es en general más tediosa, llevando mucho tiempo en el control de manipuladores en tiempo real. Existen singularidades o no-linealidades que hacen que el problema sea más difícil de resolver. Esto se explicará con más detalle.

Diferenciamos dos modelos:

- *Modelo cinemático directo*: consiste en determinar cuál es la posición y orientación del extremo final del robot, con respecto a un sistema de coordenadas que se tome como referencia, conocidos los valores de las articulaciones ylos parámetros geométricos de los elementos del robot.

- *Modelo cinemático inverso:* resuelve la configuración que debe adoptar el robot para una posición y orientación del extremo previamente conocidas.

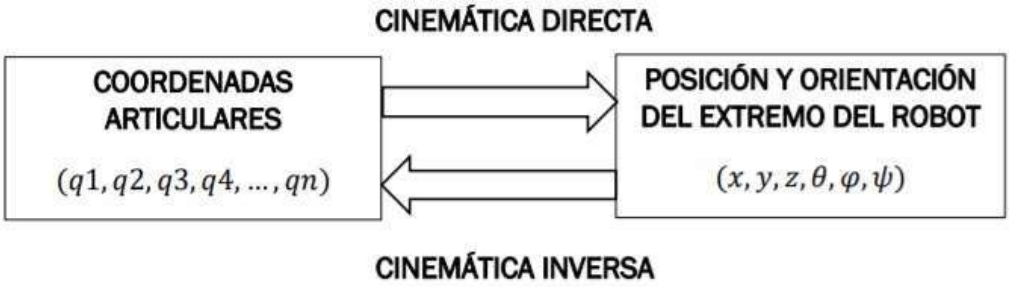


Fig. 3-1: Esquema Directa/Inversa GDL

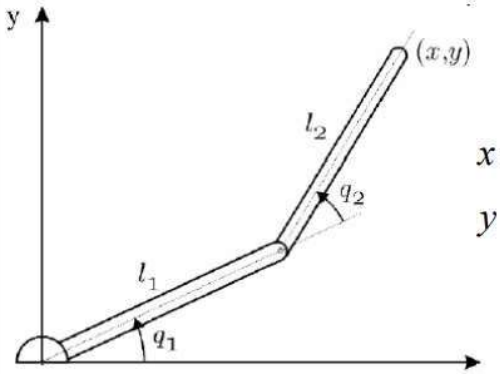
La cinemática del robot trata también de encontrar las relaciones entre las velocidades del movimiento de las articulaciones y las del extremo del robot. Es lo que se conoce como modelo diferencial, expresado mediante la matriz Jacobiana. Lo veremos más adelante, ya que será una parte clave para poder darle mayor funcionalidad al proyecto.

3.2.1 Problema Directo

a) Método geométrico:

Es una forma sencilla de resolver el problema directo de la cinemática si el número de grados de libertad no es muy alto, ya que dificultaría en gran medida la obtención del resultado. Consiste en utilizar las relaciones geométricas para hallar directamente la posición del extremo del robot en función de las variables articulares.

A continuación, se detalla el método geométrico para un brazo de dos grados de libertad.



$$x = l_1 \cos(q_1) + l_2 \cos(q_1 + q_2)$$
$$y = l_1 \sin(q_1) + l_2 \sin(q_1 + q_2)$$

Fig. 3-2: Brazo 2 GDL. Expresiones

b) Representación Denavit-Hartenberg:

Este método se basa en la transformación de matrices homogéneas en representación Denavit-Hartenberg que relaciona las traslaciones y rotaciones entre enlaces adyacentes. La letra *i* representará el elemento del robot al que nos referimos. De este modo cuando hablemos de una articulación de rotación el ángulo girado será θ_i y si es una articulación prismática indicaremos el desplazamiento mediante d_i. Los otros dos parámetros de la articulación son la distancia a_{i-1} entre el eje de la articulación i-1 y el eje de la articulación i, medida sobre la línea perpendicular común y el ángulo α_{i-1} entre estos dos ejes (ángulo entre las proyecciones de los dos ejes en un plano cuya normal es la perpendicular común) medido como rotación alrededor de la perpendicular común hasta hacer coincidir las direcciones de los ejes. Cuando el eje i-1 y el i intersectan, el valor del parámetro a_{i-1} es cero.

Las cadenas cinemáticas se describirán indicando los cuatro parámetros de *Denavit-Hartenberg* de las articulaciones. En la primera articulación de la cadena el valor de los parámetros a_0, α_0, es arbitrario y se toma como cero. Si la articulación es de rotación, el parámetro d_i correspondiente se toma también cero. Cuando la articulación es prismática, el parámetro θ_i se hace igual a cero.

Los sistemas de referencia se asignan haciendo coincidir uno de los ejes del sistema de coordenadas, típicamente el Z_i, con el eje de la articulación. El origen de {i} se escoge en el punto en el que la línea sobre la que se define a_i intersecta el eje de la articulación i. El eje X_i se elige en la dirección de la perpendicular común entre el eje de la articulación y el eje de la siguiente articulación. Para elegir el eje Y_i se sigue la regla de la mano derecha.

Por consiguiente, a_i y α_i resultan ser respectivamente la distancia desde Z_i hasta Z_{i+1} y el ángulo entre estos dos ejes medida sobre el eje X_i. El signo de α_i será positivo si al llevar Z_i sobre Z_{i+1} por el camino más corto, el sentido que resulte de aplicar la regla de la mano derecha es el mismo que el del vector X_i. Asimismo, d_i y θ_i son, respectivamente, la distancia desde X_{i-1} hasta X_i y el ángulo entre estos dos ejes medidos sobre el eje Z_i. El signo de θ_i será positivo si al llevar X_{i-1} sobre X_i por el camino más corto, el sentido que resulte de aplicar la regla de la mano derecha es el mismo que el del vector Zi.

El sistema de referencia {0} se elige de forma que Z_0 coincide con Z_1 cumpliéndose siempre que $a_0 = \alpha_0$. Cuando la última articulación es de rotación, el sistema de referencia {n} se elige con la dirección de Z_n alineada con X_{n-1}cuando $d_n = 0$

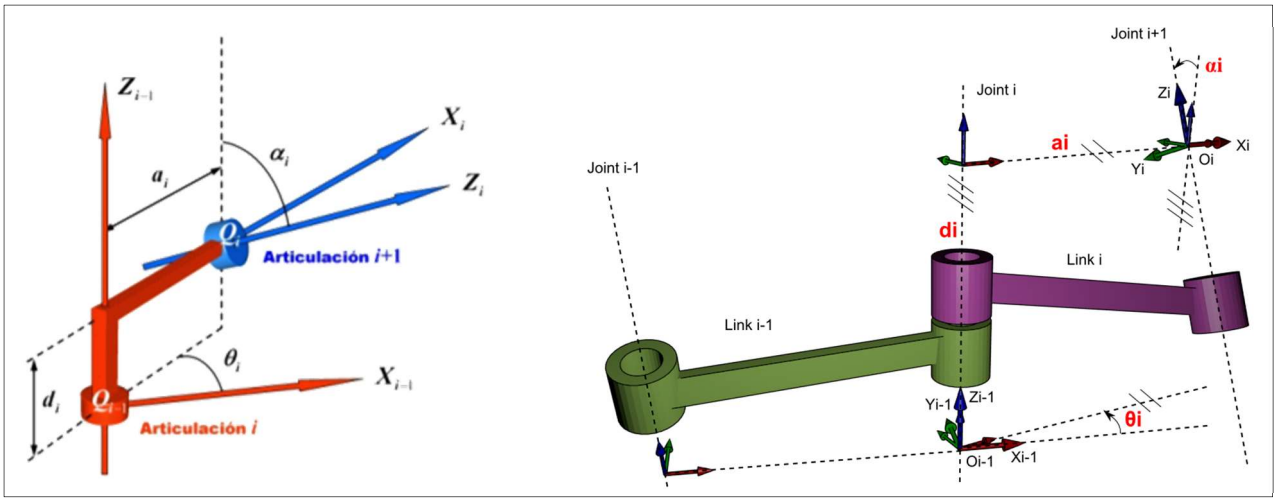


Fig. 3-3: Ejemplo Denavit-Hartenberg

Aplicando las matrices de trasformación elementales para las rotaciones y traslaciones, se obtiene la siguiente forma general asociada a la articulación

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_{i-1} \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Siendo: $c\theta_i = \cos(\theta_i), s\theta_i = \text{sen}(\theta_i)$

Cada articulación tiene una matriz asociada y a partir de las mismas, se puede obtener la trasformación compuesta de la forma:

$$p = \varphi(q) = {}^{0}_{1}T * {}^{1}_{2}T \ldots {}^{n-1}_{n}T = {}^{0}_{n}T$$

La aplicación de estas ecuaciones permite estimar la posición y orientación del efector final del manipulador conociendo los valores de las variables articulares. No obstante, desde el punto de vista de la implementación, resulta más interesante desarrollar las ecuaciones involucradas eliminando las expresiones redundantes.

Como ejemplo veremos el método en el brazo del proyecto:

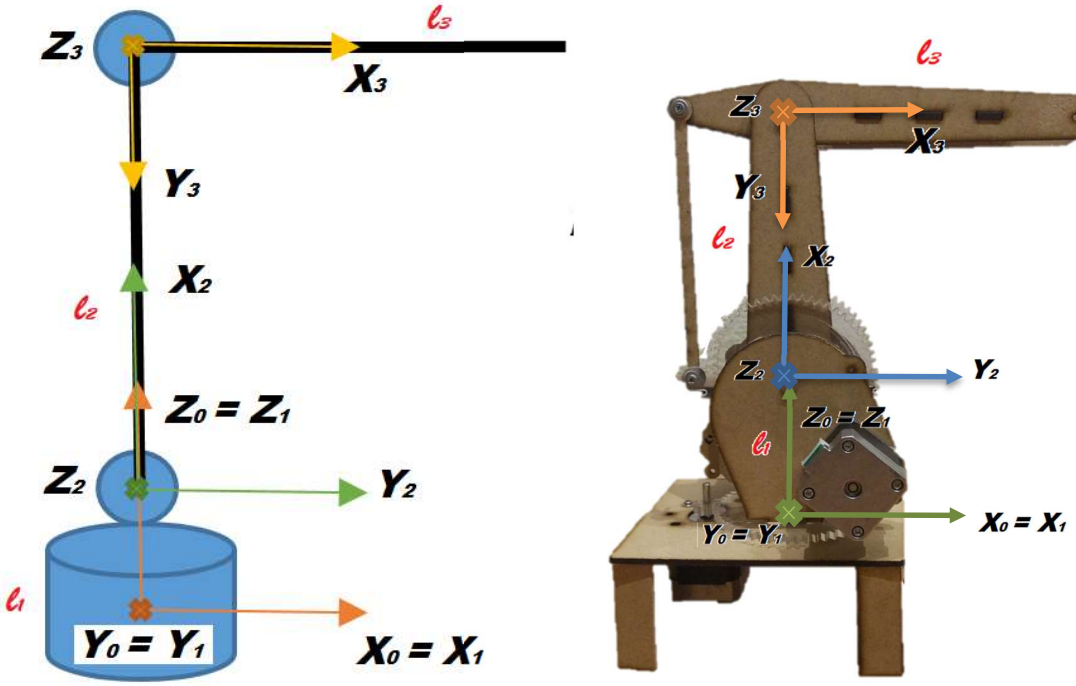


Fig. 3-4: Esquema brazo articulado

El primer paso es enumerar los eslabones, siendo {0} el punto fijado a tierra, {1} el primer eslabón móvil L1, etc.

Enumerar las articulaciones, siendo {1} el primer grado de libertad, {3} el último.

Siguiendo el resto de pasos indicados arriba, llegamos a los valores en la siguiente tabla:

i	Θ_i	α_{i-1}	a_{i-1}	d_i
1	Θ_1	90	0	L_1
2	Θ_2	0	L_2	0
3	Θ_3	0	L_3	0

Tabla 3-1: Parámetros Denavit-Hartenberg

Parámetro	*Medida(mm)*
L_1	75
L_2	142
L_3	152

Tabla 3-2: Longitud eslabones

De este modo, hallamos las matrices de trasformación de las tres articulaciones son:

$$ {}^{0}_{1}T = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{1}_{2}T = \begin{bmatrix} c_2 & -s_2 & 0 & l_2c_2 \\ s_2 & c_2 & 0 & l_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{2}_{3}T = \begin{bmatrix} c_3 & -s_3 & 0 & l_3c_3 \\ s_3 & c_3 & 0 & l_3s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

En nuestro caso, la matriz homogénea correspondiente que representa la posición y orientación del extremo del brazo robótico se obtiene como:

$$ {}^{0}_{3}T = {}^{0}_{1}T * {}^{1}_{2}T * {}^{2}_{3}T = \begin{bmatrix} c_1c_2c_3 - c_1s_2s_3 & c_1c_2s_3 - c_1c_3s_2 & s_1 & l_2c_1c_2 + l_3(c_1c_2c_3 - c_1s_2s_3) \\ s_1c_2c_3 - s_1s_2s_3 & -s_1s_3c_2 - s_1s_2c_3 & -c_1 & l_2s_1c_2 + l_3(s_1c_2c_3 - s_1s_2s_3) \\ c_2c_3 - c_2s_3 & c_2c_3 - c_2s_3 & 0 & l_1 + l_2s_2 + l_3(s_2c_3 + c_2s_3) \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Siendo:

$c_1 = \cos(\Theta_1), c_2 = \cos(\Theta_2), c_3 = \cos(\Theta_3)$

$s_1 = \text{sen}(\Theta_1), s_2 = \text{sen}(\Theta_2), s_3 = \text{sen}(\Theta_3)$

Definida la matriz homogénea genérica como:

$$ {}^{0}_{n}T = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

La posición final del efector final será:

$$ p_x = l_2c_1c_2 + l_3(c_1c_2c_3 - c_1s_2s_3) $$
$$ p_y = l_2s_1c_2 + l_3(s_1c_2c_3 - s_1s_2s_3) $$
$$ p_z = l_1 + l_2s_2 + l_3(s_2c_3 + c_2s_3) $$

Esta matriz nos aporta mucha información, siendo así la posición representada por el vector *p* y la orientación del efector final por los vectores *n, o, a* (normal, deslizamiento, aproximación).

En este proyecto, la orientación se obviará ya que el efector final no tendrá ningún grado de libertad, permaneciendo fijo en el último eslabón del brazo. Aunque se detallará los valores para mejor entendimiento.

3.2.2 Problema Inverso

Aquí se hace referencia a la obtención de las variables articulares para que la posición y orientación del robot, o en particular del efector final sea la deseada.

Se trata de resolver el problema inverso al apartado anterior. El problema puede descomponerse en dos también:

1) Determinar las transformaciones que permiten obtener la posición del efector final. De forma más precisa, suponiendo conocido un objetivo {O} se efectuarían las siguientes operaciones:
    a. Cálculo de la posición en la que debe estar la herramienta {H} conociendo el objetivo {O}
    b. Cálculo de la posición de la muñeca a partir de {H}

2) Obtener los valores de las variables articulares conocida la posición y orientación en el espacio cartesiano definida por la muñeca.

Fíjese en que la relación entre el sistema de la muñeca y la base del manipulador viene dada por el modelo directo de este. Se trata ahora de resolver el modelo inverso del manipulador para obtener los valores de las variables articulares:

$$q = F(\varphi^{-1}(p))$$

Nótese que se involucra la resolución de un sistema de ecuaciones no lineales. Así en un robot manipulador con seis articulaciones rotacionales, la matriz ${}^0_6T$ que define el modelo directo tiene dieciséis elementos, de los cuales cuatro son triviales (última fila de la matriz de trasformación). Se plantea doce ecuaciones no lineales para obtener los seis ángulos de las seis articulaciones. Por tanto, la resolución de este problema hace necesario considerar, en primer lugar, la existencia de muchas soluciones. En efecto, se trata de asegurar que la posición y orientación de la muñeca pueda o no ser alcanzada, es decir, si está o no dentro de lo que se denomina espacio de trabajo alcanzable.

Además, al igual que en cinemática directa, existen varios métodos, viéndose el geométrico y el de matrices homogéneas, se verá la explicación de los métodos con un ejemplo:

a) Método geométrico:

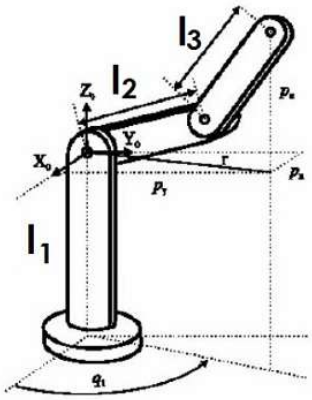


Fig. 3-5: Brazo 3 GDL

Analizando geométricamente hallamos:

$$q_1 = arctg(\frac{p_y}{p_x})$$

$$\left.\begin{aligned} r^2 &= p_x^2 + p_y^2 \\ r^2 + p_z^2 &= l_2^2 + l_3^2 + 2l_2l_3\cos(q_3) \end{aligned}\right\} \quad \cos(q_3) = \frac{p_x^2 + p_y^2 + p_z^2 - l_2^2 - l_3^2}{2l_2l_3}$$

$$\sin(q_3) = \pm\sqrt{1 - \cos(q_3)^2}$$

$$q_3 = arctg\left(\frac{\pm\sqrt{1 - \cos(q_3)^2}}{\cos(q_3)}\right)$$

Para hallar q_2 hay que tener en cuenta la posición en la que queremos el brazo de esta forma:

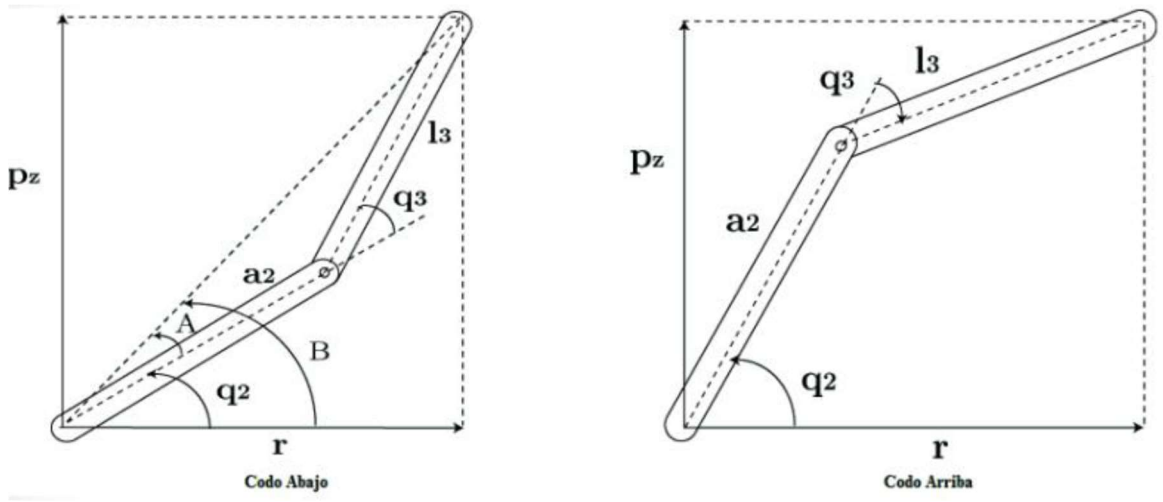


Fig. 3-6: Configuración codo Abajo, codo Arriba

La posición buscada es codo arriba, por tanto:

$q_2 = \beta + \alpha\ (codo\ arriba)$

$q_2 = \beta - \alpha\ (codo\ abajo)$

Siendo:

$$\beta = arctg\left(\frac{p_z}{r}\right) = arctg\left(\frac{p_z}{\pm\sqrt{p_x^2 + p_y^2}}\right)$$

$$\alpha = arctg\left(\frac{l_3 \sin(q_3)}{l_2 + l_3 \cos(q_3)}\right)$$

b) **Método de matrices homogéneas:**

Se basa en manipular las ecuaciones obtenidas a partir del modelo Cinemático Directo. Esto es, despejar las n variables q_i en función de los vectores **n, o, a, p.**

Operamos de la siguiente forma:

$$^0_3T = {}^0_1T * {}^1_2T * {}^2_3T = \begin{bmatrix} c_1c_2c_3 - c_1s_2s_3 & c_1c_2s_3 - c_1c_3s_2 & s_1 & l_2c_1c_2 + l_3(c_1c_2c_3 - c_1s_2s_3) \\ s_1c_2c_3 - s_1s_2s_3 & -s_1s_3c_2 - s_1s_2c_3 & -c_1 & l_2s_1c_2 + l_3(s_1c_2c_3 - s_1s_2s_3) \\ c_2c_3 - c_2s_3 & c_2c_3 - c_2s_3 & 0 & l_1 + l_2s_2 + l_3(s_2c_3 + c_2s_3) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$(^0_1T)^{-1} * {}^0_3T = {}^1_2T * {}^2_3T \longrightarrow$ despejamos q_1

$(^1_2T)^{-1} * (^0_1T)^{-1} * {}^0_3T = {}^2_3T \longrightarrow$ despejamos q_2 y q_3

3.2.3 Cinemática diferencial

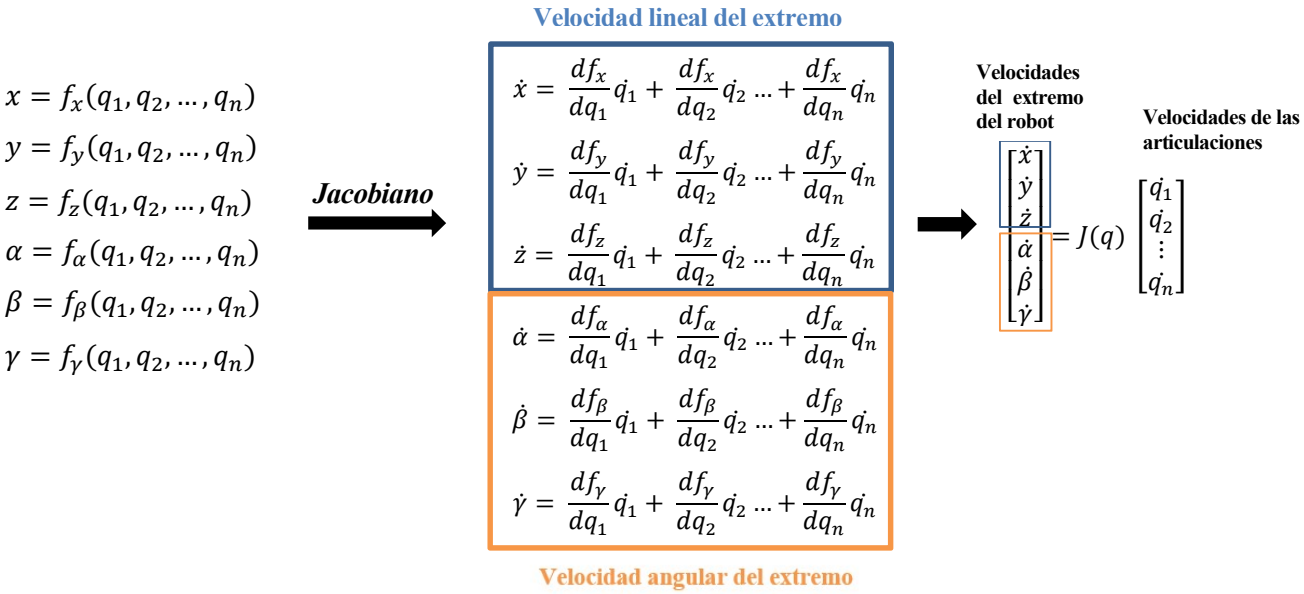
Hasta ahora se ha considerado las relaciones de las articulaciones de una manera estática en ausencia de movimiento del robot (problemas de cinemática directa e inversa).

Cuando el robot se mueve, los elementos de la cadena cinemática se propagan de una articulación a la siguiente tanto con velocidades lineales como angulares. El modelo diferencial se encarga de esta tarea. Usa como herramienta la matriz jacobiana que nos dará las velocidades de las articulaciones o del extremo del robot, dependiendo si se obtiene para el modelo directo o el modelo inverso de la cinemática.

La jacobiana es una matriz formada por las derivadas parciales de primer orden de una función. En general, para

un conjunto de *m* funciones que depende de *n* variables independientes. Si se considera que las *n* variables son dependientes del tiempo, habrá que tener en cuenta que para cada instante la matriz Jacobiana será distinta, como pasará en este proyecto.

En robótica, la jacobiana se utiliza tomando como funciones las correspondientes a las posiciones del extremo del robot siendo las variables independientes de dichas funciones las articulaciones. A partir de ahora hablaremos de las ecuaciones de posición *(x, y, z)* y orientación (α, β, γ)



3.3 Gráficas cinemáticas del brazo en MatLab

A continuación, se aprecian ciertas gráficas del movimiento relativo a cada articulación respecto al desplazamiento en ejes cartesianos, siendo estos, dados en dos direcciones paralelas al suelo (en X e Y) y otro vertical (en eje Z).

Además, incluiremos el comportamiento de la posición y la velocidad en el caso en que nuestro brazo funcione en automático ya que será gobernado por una trayectoria implícita en las funciones de la librería AccelStepper.

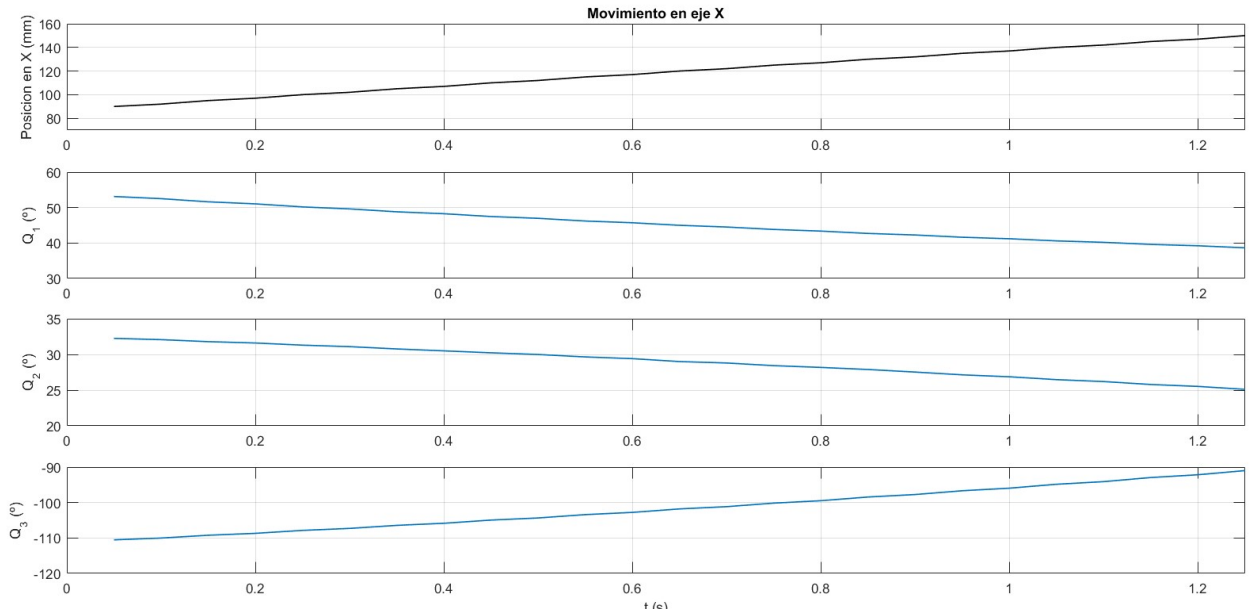


Fig. 3-7: Desplazamiento en eje X

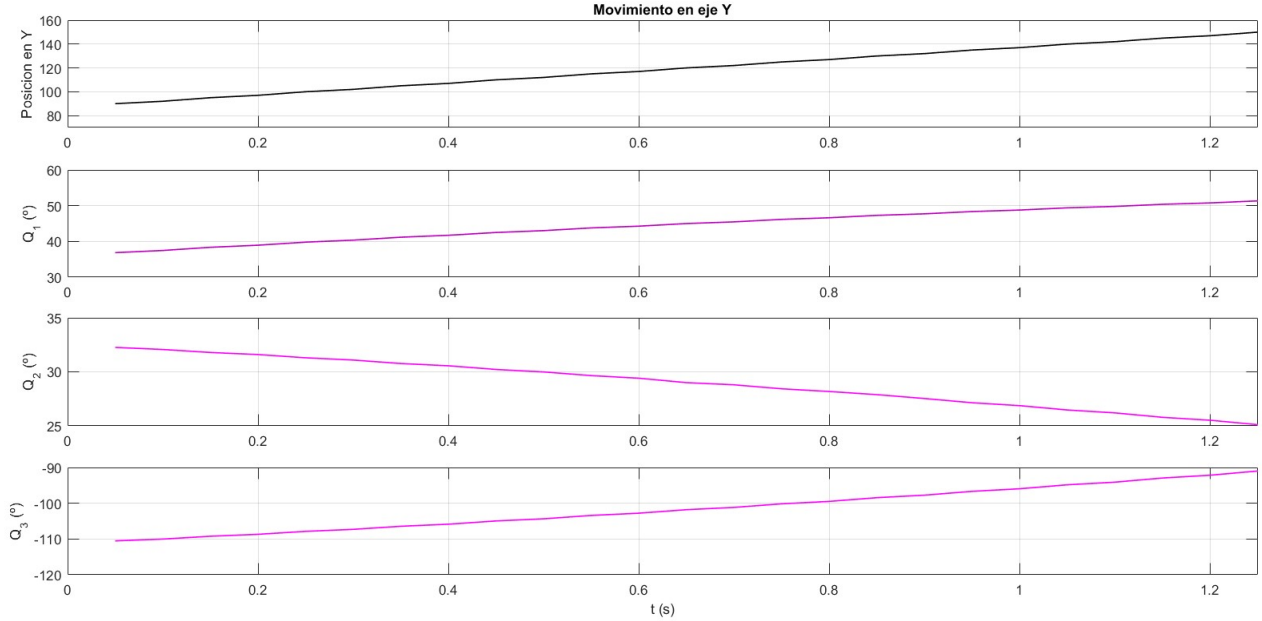


Fig. 3-8: Desplazamiento en eje Y

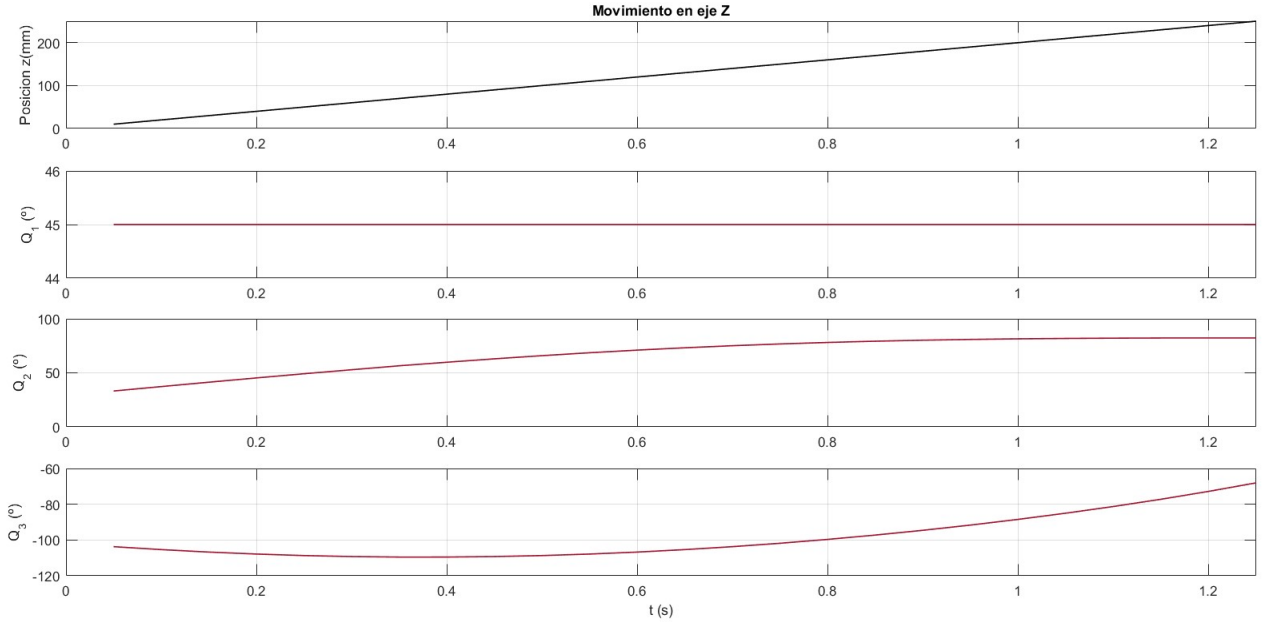


Fig. 3-9: Desplazamiento en eje Z

Como vemos en la figura en el eje Z, la articulación Q_3 crece y decrece en función del desplazamiento. Es un movimiento muy interesante que no se da en nuestro proyecto debido a las restricciones físicas del brazo. Al hablar de las velocidades, serán lineales, variando el valor de los mismo por el ángulo inclinado de cada joystick asociado en cada caso.

Al usar el jacobiano, hemos tenido en cuenta unas aproximaciones para que los costes computacionales de las operaciones no sean muy altas, aproximando velocidades y desplazamientos, el desplazamiento en automático regirá una trayectoria trapezoidal de velocidad.

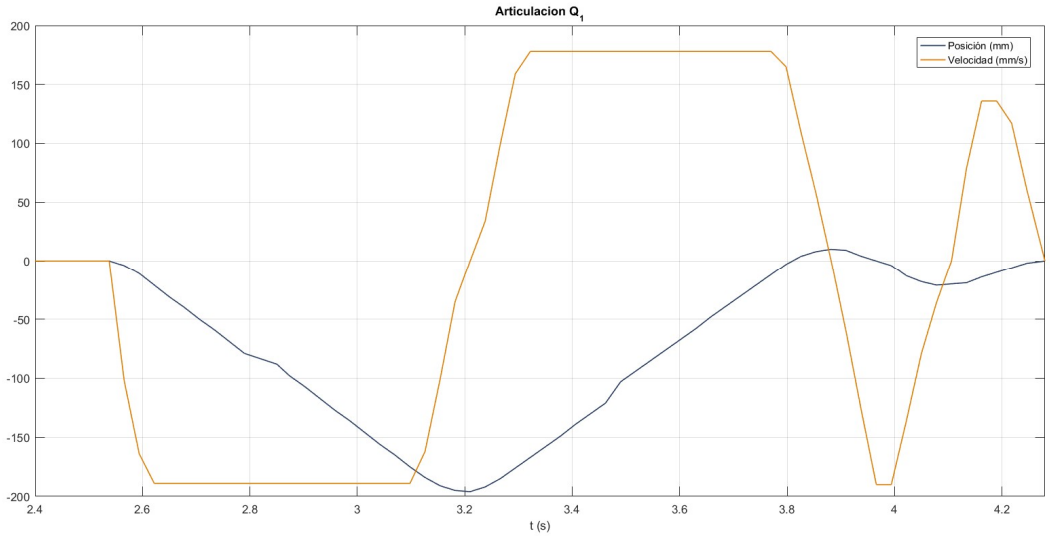


Fig. 3-10: Articulación Q_1

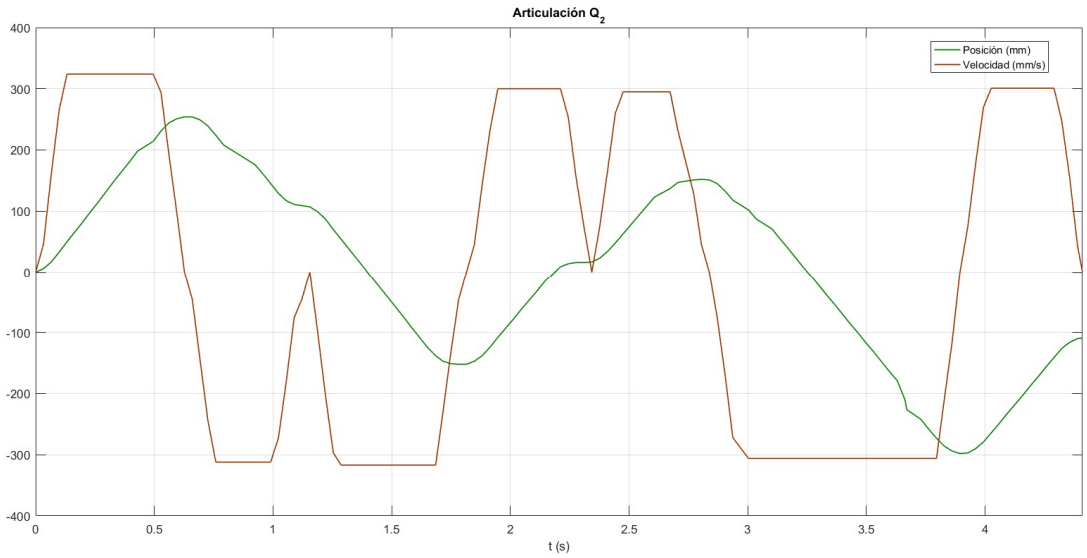


Fig. 3-11: Articulación Q_2

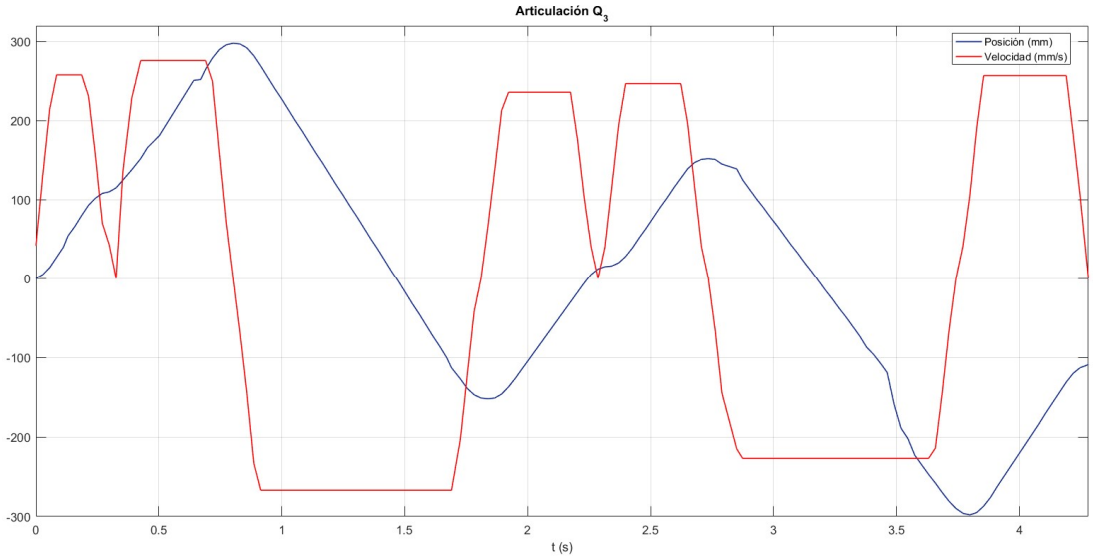


Fig. 3-12: Articulación Q_3

Las figuras 3-10, 3-11 y 3-12 representan el comportamiento de la posición y la velocidad de las articulaciones al escribir la letra ‘H’. Como vemos, el perfil de velocidad para alcanzar cada posición es trapezoidal como hemos comentado anteriormente, lo que implica un aumento de velocidad hasta alcanzar una velocidad constante para disminuir al final del movimiento, en el que se consideran velocidades y aceleraciones distintas para cada posición, calculadas mediante las matrices realizadas anteriormente.

Hay que aclarar que los ‘picos’ que se ven en las gráficas, tienen el valor cero y eso significa que el motor al finalizar la posición, se para, después comienza el siguiente movimiento, así se forma el perfil de velocidad que vemos.

4 COMPONENTES

4.1 Arduino

Fig. 4-1: Logo Arduino

4.1.1 Introducción

Arduino es una plataforma de desarrollo de software y hardware libre basada en una placa con un microcontrolador y entorno de desarrollo integrado (IDE), que vamos a usar en este proyecto, cuya principal ventaja es la facilidad de programación.

En concreto usaremos el modelo Mega 2560.

4.1.2 Micropocresador y microcontrolador:

El microprocesador es un chip integrado basado en silicio con solo una unidad de procesamiento central. Es el corazón de un sistema informático que está diseñado para realizar muchas tareas que involucran datos. Los microprocesadores no tienen RAM, ROM, pines E/S, temporizadores ni otros periféricos en el chip. Deben agregarse externamente para que sean funcionales. Consiste en la Unidad Aritmética Lógica (ALU en inglés): circuito digital que calcula y maneja todas las operaciones aritméticas y lógicas. Además, la Unidad de control que gestiona y maneja el flujo de instrucciones en todo el sistema y un Vector de Registro, que almacena los datos de la memoria para un acceso rápido. Están diseñados para aplicaciones de propósito general, como operaciones lógicas en sistemas informáticos.

En términos simples, es una CPU completamente funcional en un solo circuito integrado que es utilizado por un sistema informático para hacer su trabajo.

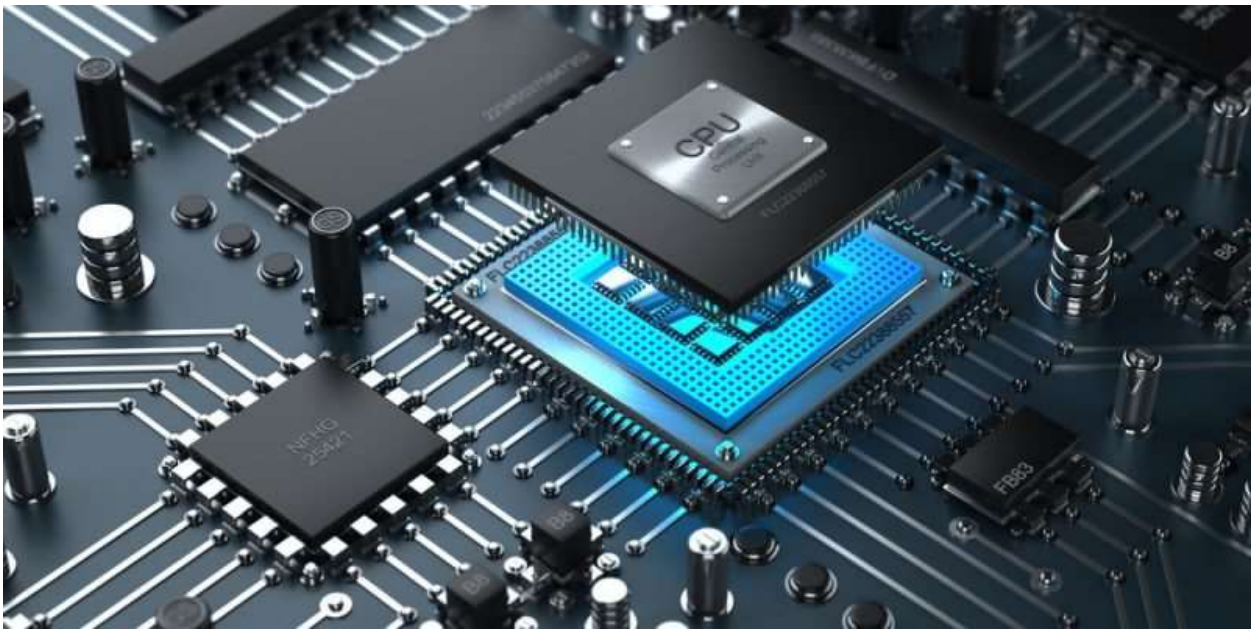


Fig. 4-2: Micropocesador

El microcontrolador es como una mini computadora con una CPU junto con RAM, ROM, puertos serie, temporizadores y periféricos E/S, todo integrado en un solo chip. Está diseñado para realizar tareas específicas de la aplicación que requieren un cierto grado de control, como un control remoto de TV, pantalla LED, relojes inteligentes, vehículos, control del semáforo, control de temperatura, etc. Es un dispositivo de alta gama con un microprocesador, memoria, y puertos de entrada/salida en un solo chip. Es el cerebro de un sistema informático que contiene suficientes circuitos para realizar funciones específicas sin memoria externa. Como carece de componentes externos, el consumo de energía es menor, lo que lo hace ideal para dispositivos que funcionan con baterías. Hablando en términos simples, un microcontrolador es un sistema completo de computadora con menos hardware externo.

Fig. 4-3: Microcontrolador

La diferencia clave entre ambos términos es la presencia de periféricos. A diferencia de los microcontroladores, los microprocesadores no tienen memoria incorporada, ROM, puertos serie, temporizadores y otros periféricos que constituyen un sistema. Se requiere un bus externo para interactuar con los periféricos. Un microcontrolador, por otro lado, tiene todos los periféricos, como procesador, RAM, ROM y pines E/S, todos integrados en un solo chip. Tiene un bus de control interno que no está disponible para el diseñador. Como todos los componentes están empacados en un solo chip, es compacto lo que lo hace ideal para aplicaciones industriales a gran escala. El microprocesador es el corazón del sistema informático y el microcontrolador es el cerebro.

4.1.3 Arduino Mega 2560. Conexión:

Es un microcontrolador basado en ATmega2560.

Características principales del microcontrolador ATmega2560 son:

- La frecuencia de trabajo máxima del chip es de 16 MHz.
- El tamaño de la memoria RAM, utilizada para cálculos y operaciones es de 8kb.
- La memoria que almacena el programa es de 256kb.
- Tiene 54 pines de entrada/salida digital, de los cuales 14 se pueden usar como salidas PWM.,
- Tiene 16 entradas analógicas y 4 líneas de puerto serie.
- Intervalo de tensión de alimentación admisibles es de 1,8 a 5,5 V.
- Conexión a ordenador mediante cable USB

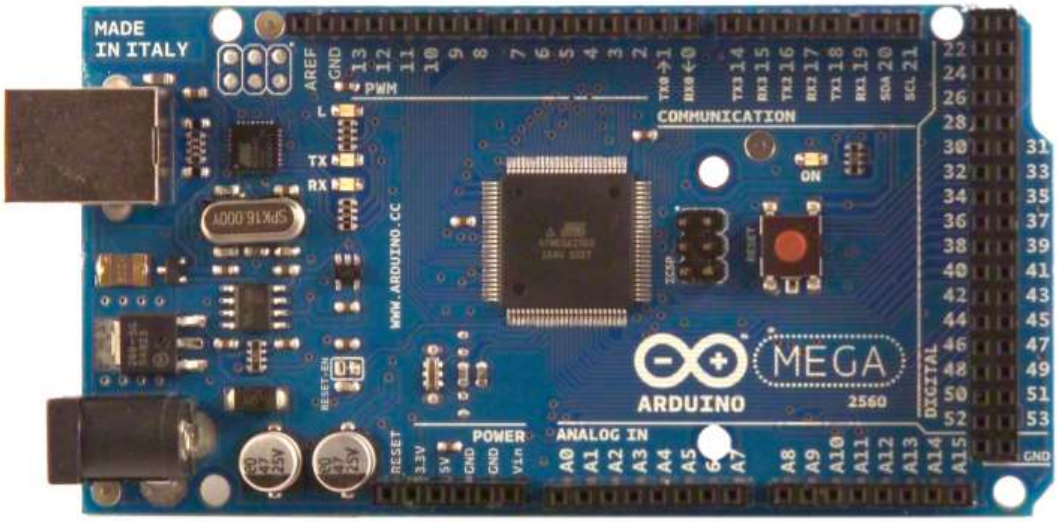


Fig. 4-4: Arduino Mega

Para conectar Arduino Mega 2560[1] a nuestro ordenador, lo primero que tenemos que hacer es ir a la página web de Arduino, en la dirección https://www.arduino.cc/en/Main/Software, y descargar el IDE de Arduino (software de código abierto), que está disponible para Windows, Mac OS X, y Linux. El software puede usarse con cualquier placa de Arduino. Descargaremos la última versión disponible compatible con nuestro sistema, ARDUINO 1.8.5 (en el momento en que lo instalamos).

Además, en la página web de Arduino también podemos encontrar una guía sobre cómo empezar con Arduino Mega 2560, en la dirección https://www.arduino.cc/en/Guide/ArduinoMega2560. Necesitamos la placa Arduino Mega y un cable micro-USB para conectarla al ordenador. Al conectar la placa al ordenador, esta debe encenderse y un LED empezará a parpadear. Windows (en nuestro caso, es el sistema en que lo hemos instalado) detectará la presencia de un nuevo hardware e instalará nuevos controladores automáticamente. Si no lo hace automáticamente, deberemos instalarlos. Para Arduino Mega 2560, deberíamos instalar el driver CH-340 para el USB. En nuestro caso no ha hecho falta. Una vez dentro del programa, en la pestaña Herramientas debemos seleccionar nuestra placa, el procesador que utilizaremos y el puerto serie al que está conectada (COM). En la parte inferior derecha del programa, nos aparecerá el nombre de nuestra placa, el procesador y el puerto COM al que está conectado (en nuestro caso COM3).

Dentro de la pestaña Archivo – Ejemplos – 01. Basics – Blink, cargamos el primer sketch para comprobar el funcionamiento de nuestra placa. Este sketch hace parpadear un LED que ya está incluido en la placa, conectado al pin número 13, por lo que no necesitamos nada más que nuestra placa para comprobar que funciona.

[1] Hoja de datos de la placa de Arduino: http://www.mantech.co.za/datasheets/products/A000047.pdf

4.1.4 Programar en Arduino:

Cuando hablamos de 'sketch' nos referimos al código del programa y se diferencia en dos secciones principales:

- setup: ejecutado al encender la placa o después de pulsar RESET, se utiliza para inicializar los modos de trabajo de los pines o el puerto serie.
- loop: contiene las instrucciones repetidas continuamente (en bucle) mientras arduino es alimentado.

EL sketch se muestra asi:

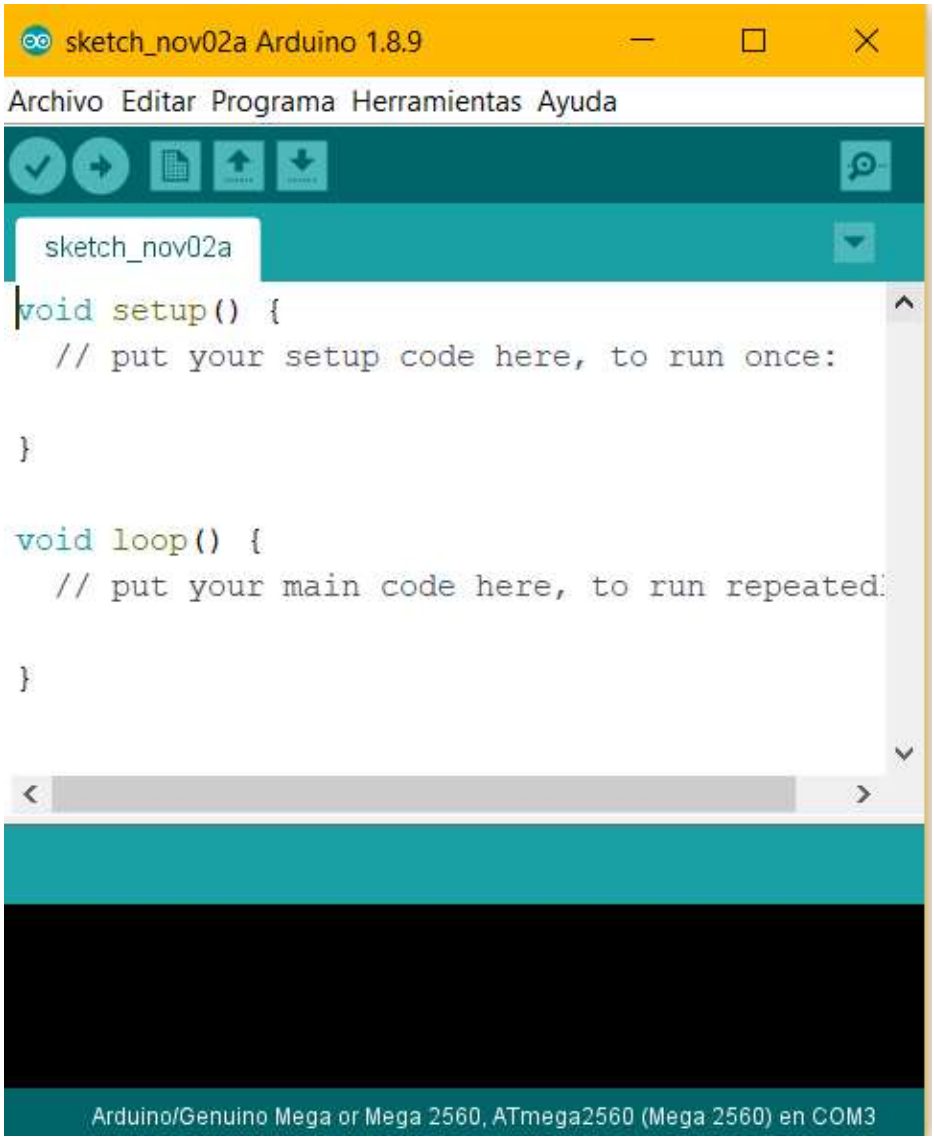


Fig. 4-5: Interfaz del IDE de Arduino

Las funciones que no devuelven ningún resultado van precedidas de la palabra clave *void*. Es obligatorio crear un sketch con *loop* y *setup*, pero no lo es insertar el código dentro de las dos secciones. Puede existir código solo en una de ellas.

Una vez escrito el código y montado el circuito, debemos cargarlo. Pulsamos el botón Verificar para comprobar el código. Esto tardará unos segundos. Después pulsamos el segundo botón Subir para transferir el código a la placa. Los LED TX y RX parpadearán hasta que todo el programa se haya subido a la placa.

Al finalizar la transferencia, en la parte inferior de la placa aparecen indicados el tamaño en bytes del sketch, el

porcentaje de espacio ocupado en memoria y otras indicaciones. Si ha surgido algún error, aparecerán mensajes en color naranja y la línea del sketch donde está el error en caso de que sea de escritura del código. Los errores más habituales se deben a la conexión del puerto serie.

Alguno de los comandos más usados en Arduino:

- **pinMode** (pin, mode). Esta instrucción es utilizada en la parte de configuración setup () y sirve para configurar el modo de trabajo de un PIN pudiendo ser INPUT (entrada) u OUTPUT (salida). Los terminales de Arduino, por defecto, están configurados como entradas.
- **digitalWrite** (pin, value). Envía al pin definido previamente como OUTPUT el valor HIGH o LOW (poniendo en 1 o 0 la salida). El pin se puede especificar ya sea como una variable o como una constante.
- **digitalRead** (pin) Lee el valor de un pin (definido como digital) dando un resultado HIGH (alto) o LOW (bajo). El pin se puede especificar ya sea como una variable o una constante.
- **analogWrite** (pin, value) Esta instrucción sirve para escribir un pseudo-valor analógico utilizando el procedimiento de modulación por ancho de pulso (PWM) a uno de los pines de Arduino marcados como "pin PWM". El valor que se puede enviar a estos pines de salida analógica puede darse en forma de variable o constante, pero siempre con un margen de 0-255.
- Si enviamos el valor 0 genera una salida de 0 voltios en el pin especificado; un valor de 255 genera una salida de 5 voltios de salida en el pin especificado. Para valores de entre 0 y 255, el pin saca tensiones entre 0 y 5 voltios - el valor HIGH de salida equivale a 5v. Debido a que esta es una función de hardware, en el pin de salida analógica (PWM) se generará una onda constante después de ejecutada la instrucción analogWrite hasta que se llegue a ejecutar otra instrucción analogWrite (o una llamada a digitalRead o digitalWrite en el mismo pin).
- **analogRead** (pin): Lee el valor de un determinado pin definido como entrada analógica con una resolución de 10 bits. El rango de valor que podemos leer oscila de 0 a 1023.

Comunicación Puerto Serie:

A través de este tipo de comunicación podremos enviar datos a y desde nuestro Arduino a otros microcontroladores o a un computador ejecutándolo.

El mismo cable con el que programamos el Arduino desde un computador es un cable de comunicación serial. La velocidad de comunicación con el puerto serie se ajusta normalmente a 9600 baudios (grupos de bits transmitidos por segundo a través de una línea digital). En comunicación serial este valor es muy importante ya que todos los dispositivos que van a comunicarse deben tener la misma velocidad para poder entenderse.

Se configura de la siguiente manera:

```
void setup() {
Serial.begin(9600);
}
void loop() {
}
```

Ademas nos ofrece varios comandos para leer en el IDE (Entorno de Desarrollo Integrado) para ver los valores que salen automáticamente:

*Serial.print(***valor***)*: Si añadimos comillas (**"** … **"**) dentro del paréntesis, podremos escribir cualquier frase para que aparezca en el IDE.

4.2 Motores paso a paso

En este proyecto usamos motores paso a paso Nema 17, que nos aporta más precisión y control de los movimientos que un servomotor. Cada motor estará ensamblado en un mecanismo engranaje-piñon con el que moverá nuestro brazo. Estarán alimentados por una fuente de alimentación de 12 voltios y 5 amperios, suficiente para todo el circuito, ya que cada motor consume 1 amperio.

4.2.1 Control del motor paso a paso

Un motor paso a paso (también llamado stepper motor) es un dispositivo electromagnético que convierte impulsos eléctricos en movimientos mecánicos de rotación. La principal característica de estos motores es que se mueven un paso por cada impulso que reciben. Normalmente los pasos pueden ser de 1.8º a 90º por paso, dependiendo del motor. Por ejemplo: un motor paso a paso que se mueve 2º cada paso, quiere decir que para completar una vuelta (360º) tendrá que dar (360º/2ºpor paso) 180 pasos.

Son motores con mucha precisión, que permiten quedarse fijos en una posición (como un servomotor) y también son capaces de girar libremente en un sentido u otro (como un motor DC)

Los Motores paso a paso están formados por dos partes:

-El **estator** es la parte fija del motor donde en sus cavidades van depositadas las bobinas.

-El **rotor** es la parte móvil del motor construido por un imán permanente.

Estas dos partes van montadas sobre un eje.

Fig. 4-6: Motores paso a paso

Cuando circula corriente por una o más bobinas del estator se crea un campo magnético apareciendo los polos Norte-Sur. Luego el rotor se equilibra magnéticamente orientando sus polos Norte-Sur hacia los polos Sur-Norte del estator. Cuando el estator vuelva a cambiar la orientación de sus polos a través de un nuevo impulso recibido hacia sus bobinas, el rotor volverá a moverse para equilibrarse magnéticamente. Si se mantiene esta situación, obtendremos un movimiento giratorio permanente del eje. El ángulo de paso depende de la relación entre el nombre de polos magnéticos del estator y el nombre polos magnéticos del rotor.

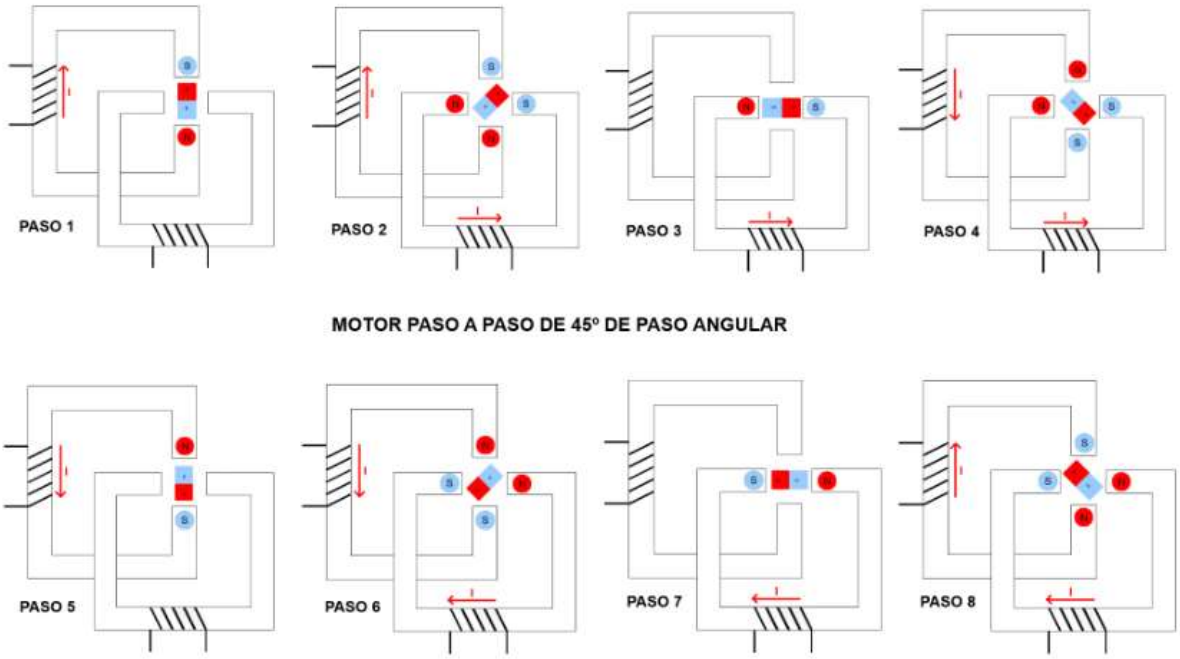


Fig. 4-7: Esquema funcionamiento motor stepper

Hay dos tipos de motores paso a paso: los unipolares y los bipolares.

Los bipolares se componen de 2 bobinas y los unipolares de 4 bobinas. Para diferenciarlos físicamente basta con observar el número de terminales de cada motor. Los bipolares siempre tienen 4 terminales, dos para cada bobina, y los unipolares normalmente tienen 6 terminales, dos para cada bobina y los otros dos son los comunes de estas. Hay motores unipolares con 5 terminales en que los dos comunes están unidos internamente.

La diferencia entre los dos es que en un motor paso a paso unipolar se activa una bobina a la vez mientras que un motor bipolar se activa más de una bobina a la vez. Esto hace que un motor bipolar tenga más par motor que un motor unipolar. Por contra, un motor bipolar es más complejo de controlar que un unipolar.

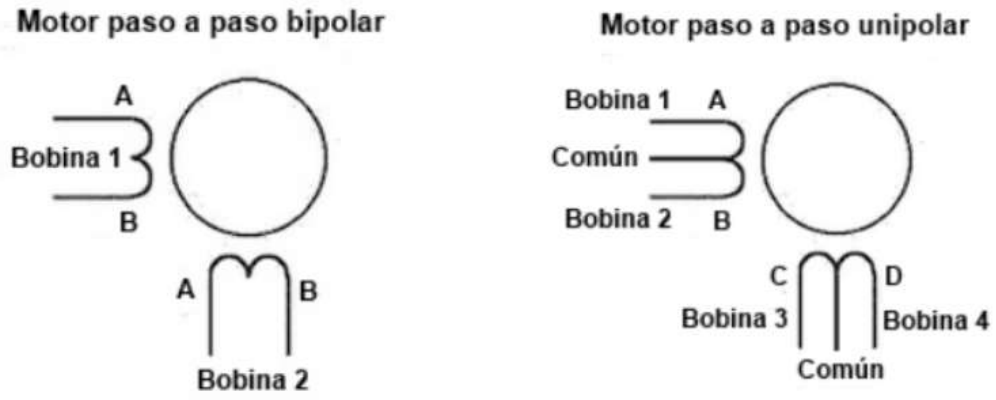


Fig. 4-8: Esquema interno stepper

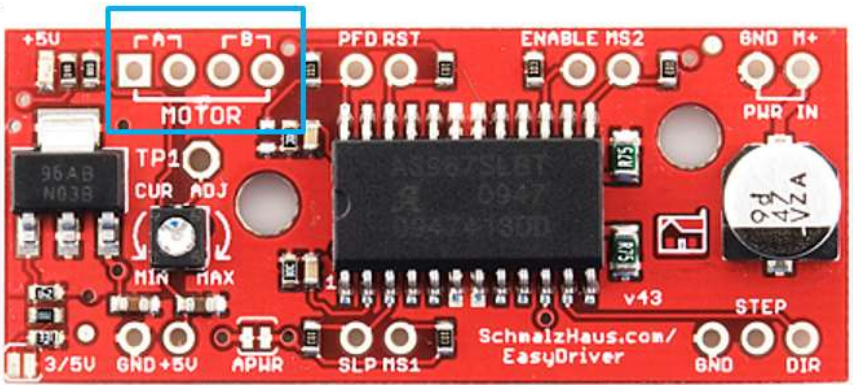
Además, para controlar el motor paso a paso mediante arduino usamos unos drivers para cada motor con el A3967LSB de EasyDriver[2] que junto con la librería *AccelStepper.h* de arduino podremos configurar a nuestro antojo el movimiento del brazo.

[2] Datasheet del driver: https://www.electronicoscaldas.com/datasheet/A3967-EDMOD_Manual.pdf

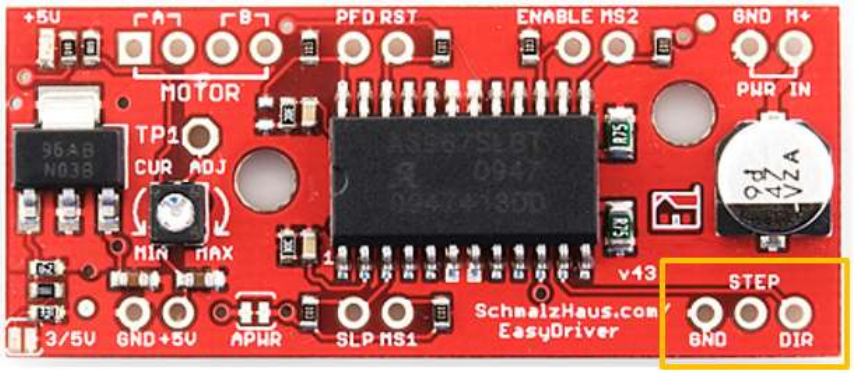
4.3 Driver A3967LSB

Estos son los drivers que se conectan a los motores paso a paso para su funcionamiento de paso/dirección que permite controlar una alimentación entre 7 y 30V y un consumo de hasta 750mA por bobina. Incluye un regulador de tensión de 5V del que podemos obtener la alimentación necesaria para nuestro controlador digital. Tiene control de microsteps de 4 niveles; el valor predeterminado de micropaso es de 8 pasos (si el motor tiene 200 pasos completos por revolución, obtendrá 1600 pasos por vuelta con el driver). Esta configuración se puede anular fácilmente conectando el pin MS1 y/o MS2 para configurar el controlador para 1/8, 1/4 o 1/2 para ajustar la resolución. Es compatible con motores bipolares de 4, 6 y 8 cables.

Al ser motor bipolar, A y B corresponderá a la conexión de cada polo del motor, conectándose cada pin en orden como se indica arriba para el correcto funcionamiento.



Para aportar el sentido de giro en todo momento del motor, el driver nos facilita tres pines que irán conectadas a tres pines digitales de nuestro arduino, configurándose en el propio programa.



Por último, falta alimentar el motor a través del driver con los pines GND y M+

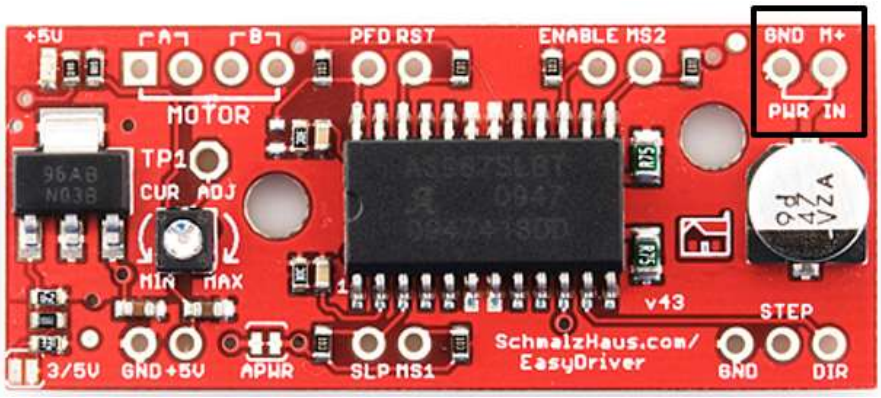


Fig. 4-9: Drivers

4.4 Joystick

Es un periférico de entrada que gira sobre una base e informa su ángulo o dirección al dispositivo que está controlando. Tiene dos ejes de giros, cada uno conectado a un potenciómetro y además tiene un pulsador que se activa cuando empujamos el mando del joystick hacia la base.

Arduino mediante la función AnalogRead() lee el recorrido del joystick en cada uno de los ejes, usando además la función map() podremos cambiar la escala del valor leído permitiendo un control progresivo. El joystick al moverse crea una diferencia de tensión que es regulada por el potenciómetro, creando una lectura digital de la posición movida por el mismo.

Como el microcontrolador tiene una resolución ADC de 10 bits, los valores en cada canal analógico pueden variar de 0 a 1023. Al conectar el VRx a A0 y VRy a A1, las entradas analógicas deben mostrar los valores como se muestra en la imagen:

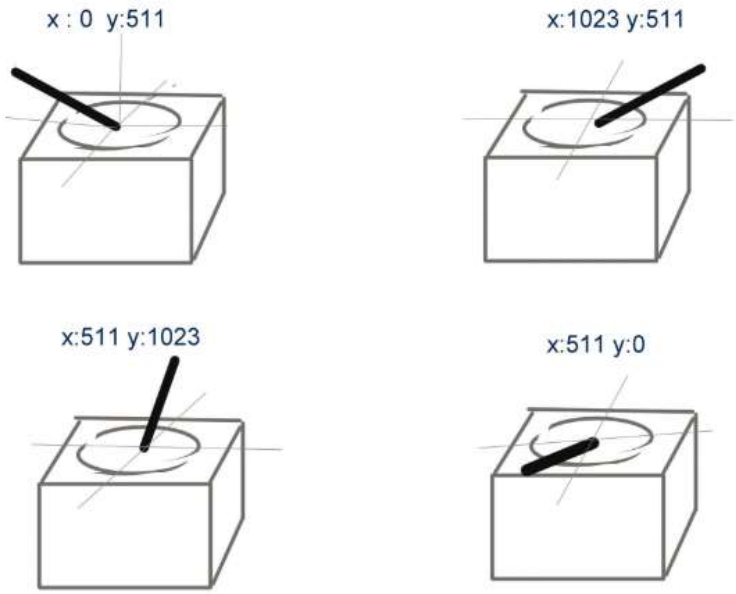


Fig. 4-10: Movimiento relativo joystick

La posición inicial para el joystick es en (x, y) = (511, 511). Si el stick se mueve en el eje X de un extremo al otro, los valores de X cambiarán de 0 a 1023 y sucederá algo similar cuando se mueva a lo largo del eje Y. De esta forma, se puede generar cualquier combinación de valores entre 0 y 1023 para cada coordenada.

Como ejemplo:

```
1. int x = analogRead(A0); //Asignamos el pin 0 a la lectura del Joystick
2. int mapx = map(x, 0, 1023, -1, 1);
```

La función *map()* de 0 a 1023 es el valor que reconoce el joystick, de -1 a 1 es el rango de valores en el que transforma ese valor.

4.5 Sensor Fin de Carrera

Fig. 4-11: Fin de carrera

Es un interruptor que se acciona, en nuestro caso, cuando una articulación llega al límite del movimiento o para establecer la posición origen. Se situa al final de un recorrido para, o bien hacer parar el movimiento o cambiar el sentido.

Su diseño se asemeja al de un botón al que se le ha incluido un accionador, el cabezal en forma de chapa rectangular, que ayudará a mantener en contacto la pieza que hará cambiar el estado del pulsador.

Internamente pueden contener interruptores normalmente abiertos (NA), normalmente cerrados (NC) o conmutadores dependiendo de la operación que cumplan al ser accionados, de ahí la gran variedad de finales de carrera que existen en mercado.

Como se aprecia en la Figura 4-11, tiene 3 terminales, la primera pata corresponderá al común y las otras dos siguientes alternarán entre circuito NA o NC. Si es NA, el circuito se encontrará abierto (no pasa corriente entre los terminales) y cuando el cabezal del sensor pulse el botón, cerrará el circuito haciendo que pase corriente entre los terminales. Si es NC, ocurrirá el efecto contrario.

Para este proyecto la configuración escogida ha sido la normalmente abierto.

4.6 Conexión de todos los componentes

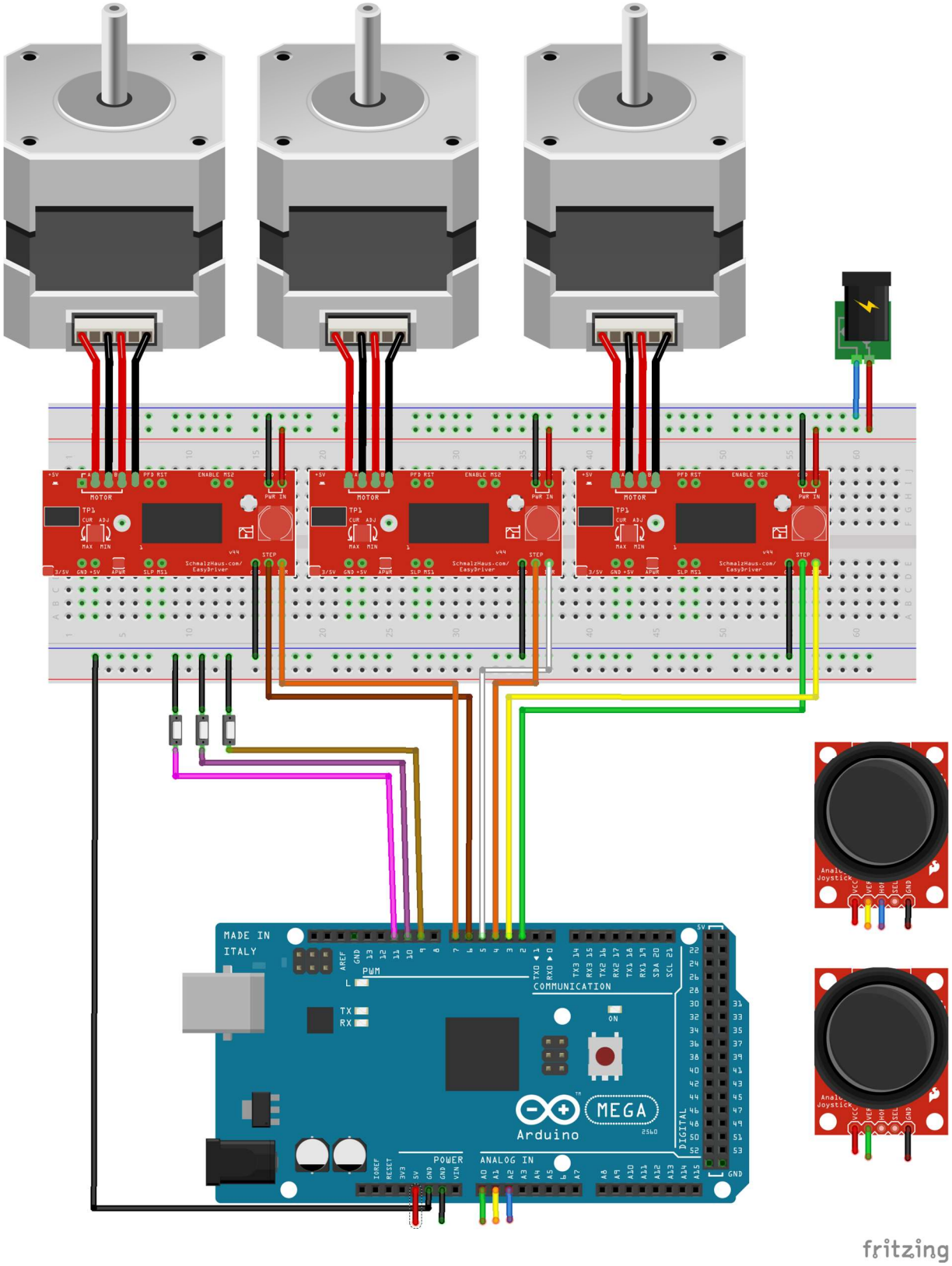


Fig. 4-12: Circuito esquemático conexión general

5 PROGRAMACIÓN

Primero se expondrá un pseudocódigo[3] en el que se podrá ver el proceso seguido para implementar el control en el microcontrolador.

5.1 Pseudocódigo

Existirán problemas de singularidad cuando el jacobiano se hace nulo, lo que implica que un pequeño incremento de las coordenadas cartesianas provocaría un incremento infinito en las coordenadas articulares. Esto provocaría que los actuadores del robot intentasen dar unas velocidades imposibles de alcanzar por el extremo del robot, así como perjudicial para el sistema robótico al completo. Esto no se ha tenido en cuenta ya que nuestra configuración permite el movimiento adecuado.

El pseudocógido es el siguiente:

- **`Cabeceras:`**
  - Bibliotecas
  - Variables usadas
- **`Void Setup:`**
  - Reserva de memoria para puerto serie
  - Activación puerto serie
  - Habilitación de motores.
  - Motores a posición home
  - Void *Loop:*
- **`Void Loop:`**
  - Pulsar joystick para modo de movimiento.

[3] En el Anexo A de este documento se podrá ver el código completo

- Si Joystick = 1
  - Modo Autónomo:
    - Reset Flags
    - Leer posiciones
    - Elegir letra
    - Hacer:
      - Trayectoria
      - Calcular cinématica inversa
      - Calcular velocidades y aceleración
      - Mover motores
      - Actualizar posición

- Si Joystick = 0
  - Modo Manual:
    - Leer entradas de dos Joysticks
    - Cada joystick mueve uno o dos motores que corresponde a las articulaciones

- `Fin.`

5.2 IDE Arduino

El entorno de desarrollo integrado (IDE) es una aplicación escrita en lenguaje de programación Java. Se utiliza para escribir y cargar programas en placas compatibles con Arduino.

5.2.1 Librería AccelStepper

Las librerías estándares que trae Arduino incluye la librería <*Stepper.h*> para los motores paso a paso, en este caso habría que descargar la librería <*AccelStepper.h*> que será más adecuada para las aplicaciones que usaremos en nuestro proyecto, pudiendo hacerse uso de la aceleración de los motores. En la siguiente dirección https://www.airspayce.com/mikem/arduino/AccelStepper/classAccelStepper.html encontrarás el lugar para descargarla y además la descripción de las funciones que ofrece esta librería, junto con algunos ejemplos.

Esta librería mejora significativamente la que Arduino trae para los paso a paso aportando aceleración positiva y negativa, control de varios motores a la vez independientemente, etc.

Un apartado importante es saber que la librería usa pasos por segundo en lugar de radianes por segundo, porque no conocemos el ángulo de paso del motor. Permite controlar la velocidad y aceleración de múltiples motores a la vez. Dependiendo del driver a usar, además ofrece la capacidad de poder usar los motores en función de los pasos, y a su vez, la precisión que queremos aplicar a nuestros motores.

Los motores[4] tienen 4 cables:

Las funciones más importantes de la librería son:

[4] Enlace ayuda para entender las configuraciones de los motores paso a paso. http://www.electroensaimada.com/motor-paso-a-paso.html

- **void** move (*long* relativa): esta instrucción es utilizada para establecer la posición objetivo desde la posición actual. *Long* será el tipo de dato.
- **void** moveTo (*long* absoluta): esta instrucción establece la posición objetivo.
- **boolean** run(): Activa el motor para que actúe. Es necesario indicar una velocidad y aceleración previa a declarar esta función para que actúe.
- **boolean** runSpeed(): moverá el motor a la velocidad que se indicará llamando a la función setSpeed().
- **void** setMaxSpeed (*float* velocidad): establece la velocidad máxima permitida. Junto con la función run() acelerará el motor hasta llegar a la velocidad máxima, si no se indica nada más. *Float* será el tipo de dato.
- **void** setAcceleration (*float* aceleración): establece la tasa de aceleración/desaceleración.
- **long** distanceToGo(): distancia que queda entre la posición actual y la objetivo.
- **long** targetPosition(): establece la posición objetivo.
- **long** currentPosition(): posición actual del motor.
- **void** setCurrentPosition(*long position*): resetea la posición actual del motor, considerándolo el nuevo cero.

5.2.2 Función loop

A continuación, y con la ayuda de *GeSHi*, una herramienta para convertir el código a un formato legible en Word, veremos el código usado diferenciado en partes con explicaciones.

La función *loop()* de nuestro código siempre estará esperando que mediante el botón habilitado en uno de los joysticks se le mande el pulso para indicar si el modo a utilizar es autónomo o manual.

EligeLetra() permitirá que mandes la orden mediante el teclado en el serial de Arduino. *Manual()* permitirá mover mediante los joysticks las articulaciones del brazo robótico. Dentro del modo manual si pulsamos el botón del joystick derecho, encontraremos dos subestados, en uno moveremos cada articulación independiente y en el otro, se hará en cartesianos.

```
1.  void loop() {
2.    estado = digitalRead(8);  // Leemos estado del boton del Joystick
3.    //Hacemos un contador con cada pulso de boton para diferenciar los estados
    autonomo o manual
4.    if (estado != estadoAnt ) {
5.      if (estado == 1) {        //Si boton pulsado -> contador suma 1
6.        cont++;
7.        delay(500);
8.        if (cont == 2) {        //Si contador = 2 se reincia el contador
9.          cont = 0;}
10.     }} estadoAnt = estado;
11.
12.   if (Serial.available() > 0) {
13.     if (cont == 1) {  // Como cont = 1 -> Modo autonomo
14.     if(digitalRead(home_switch0) && digitalRead(home_switch1) &&
    digitalRead(home_switch2)) {
15.       EligeLetra();
16.
17.       }
18.       else{
19.           }}
20.   } else if (cont == 0) {   // Si cont=0 -> Modo Manual
21.     Manual();}
22.
23.   }
```

5.2.3 Función EligeLetra();

Solo será para el modo autónomo, donde leeremos por el puerto serie la letra que se introduce para que nuestro brazo la escriba. Se define la letra 'H'.

Al elegir la letra, el mismo modo tendrá en consideración el bajar el boli y subir el mismo para que escriba.

```
1. // Elegimos letra desde el puerto serial //
2. void EligeLetra() {
3.   if (Serial.available() > 0) {   // Si el puerto serie esta 'abierto'
4.     data = Serial.read();         //Leemos valor introducido
5.     if (data == 'H', 'h') {       //Si coincide con H
6.       Letra_H();                  //Funcion que recorre las posiciones para H
7.       delay(200);
8.     }Serial.println(data);}        //Mostramos Letra por serial
9. }
```

5.2.4 Función CalculaInversa();

Aquí se ha implementado las ecuaciones que se han calculado previamente para la cinemática inversa.

Se hace una diferencia cuando el brazo alcanza un cierto valor límite en alguna de las coordenadas ya que puede afectar a la articulación 3 ya que, al hacer el cálculo, el resultado te puede dar el ángulo correcto o el complementario haciendo que cambie en 180 grados el ángulo que se busca, perjudicando al movimiento y la estructura.

```
1. // Modelo matematico cinematica inversa //
2. void CalculaInversa()
3. {
4.   r = sqrt(pow(xleida, 2) + pow(yleida, 2));
5.   ln = zleida - l1;
6.
7.   c3 = ((pow(xleida, 2) + pow(yleida, 2) + pow(ln, 2) - pow(l2, 2) - pow(l3, 2)
   ) / (2 * l2 * l3));
8.   q1des = atan(yleida / xleida);
9.   q2des = atan(ln / r) - atan(l3 * sin(q3) / (l2 + l3 * c3));
10.  q3des = atan(-(sqrt(1 - pow(c3, 2))) / c3);
11.
12.  q1mov[j] = q1des * (180 / pi);      //Giro de la base en Grados
13.  q2mov[j] = q2des * (180 / pi);      //Giro de la base en Grados
14.  q3mov[j] = q3des * (180 / pi) - 180;//Giro de la base en Grados
15.
16.  if (xleida > 185) {
17.  q3mov[j] = q3des * (180 / pi);}}
```

5.2.5 Función JacInverso();

Aquí se calcula la jacobiana que será parte esencial en el cálculo de las velocidades. Es solo la implementación en código arduino de lo calculado mediante la teoría.

La función *pow(*base, exp*)* eleva la base al exponente que le indiquemos.

```
1. // Calcular jacobiano Inverso para uso de velocidades //
2. void JacInverso() {
3.   //a y b son variables auxiliares para tener el calculo mas facil
4.   a = pow(xleida, 2) + pow(yleida, 2) + pow(zleida, 2) - pow(l2, 2) - pow(l3, 2
   );
5.   b = pow(xleida, 2) + pow(yleida, 2);
6.
```

```
7.    Ji[0][0] = -(yleida) / (pow(xleida, 2) + pow(yleida, 2));
8.    Ji[0][1] = (xleida) / (pow(xleida, 2) + pow(yleida, 2));
9.    Ji[0][2] = 0 ;
10.
11.   Ji[1][0] = -xleida * ((2 * l2 * l3 * zleida * pow(xleida, 2) + 2 * l2 * l3 *
    zleida * pow(yleida, 2)) *
    sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2)))) + (pow(l2, 2) - pow(xleid
    a, 2) - pow(zleida, 2) - pow(yleida, 2) - pow(l3, 2)) * pow(b, 1.5)) / (2 * l2
    * l3 * (pow(b, 1.5)) * (pow(xleida, 2) + pow(yleida, 2) + pow(zleida, 2)) *
    sqrt(1 - (pow(a, 2)) / (4 * pow(l2, 2) * pow(l3, 2))));
12.   Ji[1][1] = -yleida * ((2 * l2 * l3 * zleida * pow(xleida, 2) + 2 * l2 * l3 *
    zleida * pow(yleida, 2)) *
    sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2)))) + (pow(l2, 2) - pow(xleid
    a, 2) - pow(zleida, 2) - pow(yleida, 2) - pow(l3, 2)) * pow(b, 1.5)) / (2 * l2
    * l3 * (pow(b, 1.5)) * (pow(xleida, 2) + pow(yleida, 2) + pow(zleida, 2)) *
    sqrt(1 - (pow(a, 2)) / (4 * pow(l2, 2) * pow(l3, 2))));
13.   Ji[1][2] =  ((2 * l2 * l3 * pow(yleida, 2) + 2 * l2 * l3 * pow(xleida, 2)) *
    sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2)))) +
    sqrt(pow(xleida, 2) + pow(yleida, 2)) * pow(zleida, 3) +
    sqrt(pow(yleida, 2) + pow(xleida, 2)) * (pow(yleida, 2) + pow(xleida, 2) + pow(
    l3, 2) - pow(l2, 2)) * zleida) / (2 * l2 * l3 *
    sqrt(pow(yleida, 2) + pow(xleida, 2)) * (pow(zleida, 2) + pow(yleida, 2) + pow(
    xleida, 2)) * sqrt(1 - ((pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2))))));
14.
15.   Ji[2][0] = -xleida / (l2 * l3
    * (sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2))))));
16.   Ji[2][1] = -yleida / (l2 * l3
    * (sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2))))));
17.   Ji[2][2] = -zleida / (l2 * l3
    * (sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2))))));
18. }
```

Adjuntamos además la función que hemos usado para la velocidad:

Declaramos una velocidad lineal del efector final, vx, vy, vz.

```
1. vx = 100;
2. vy = 100;
3. vz = 100;
4.
5. // Velocidad a traves de Jacobiano inverso
6.     vq1= Ji[0][0]*vx+Ji[0][1]*vy+Ji[0][2]*vz;
7.     vq2= Ji[1][0]*vx+Ji[1][1]*vy+Ji[1][2]*vz;
8.     vq3= Ji[2][0]*vx+Ji[2][1]*vy+Ji[2][2]*vz;
```

5.2.6 Función Letra_H();

Se define un vector de posiciones definidas por las coordenadas del plano. Delimitará el recorrido de la letra. Combinado con las funciones de calcular la inversa, las velocidades y el movimiento activará los motores manteniéndolo en funcionamiento hasta que el vector se recorra entero.

```
1. void Letra_H() {
2.   i = 0;
3.   while (1) {
4.     // Vector de posiciones a recorrer //
5.     int ArrayX[8] = {170, 170, 200, 185, 185, 200, 170, 170};
6.     int ArrayY[8] = {0, 0, 0, 0, -40, -40, -40, -40};
7.     int ArrayZ[8] = {40,0, 0, 0, 0, 0, 0, 40};
8.
9.     if (i < 8) {
10.      xleida = ArrayX[i];
11.      yleida = ArrayY[i];
12.      zleida = ArrayZ[i];
13.      CalculaInversa();
14.      delay(100);
15.      JacInverso();
```

```
16.       Mover();
17.       Movimiento();
18.
19.       Serial.println(q1mov[j]);
20.       Serial.println(mov1);
21.       Serial.println(q2mov[j]);
22.       Serial.println(mov2);
23.       Serial.println(q3mov[j]);
24.       Serial.println(mov3);
25.       Serial.println(" ");
26.
27.       // Se comprueba si ha alcanza la posicion  para pasar a la siguiente //
28.       if (Motor0.currentPosition() == int(mov1) &&
    Motor1.currentPosition() == int(-mov2) &&
    Motor2.currentPosition() == int(mov3)) {
29.         i++;
30.         delay(1500);
31.       }}
32.     if (i == 8) {
33.       break;}}
34. }
```

5.2.7 Función Manual();

Si se ha activado el botón del joystick izquierdo, sabremos que el modo ha cambiado y entraremos en el control manual del brazo robótico.

Si en el otro joystick el estado del botón se altera, podremos encontrar dos modos diferentes: Movimiento articular independiente o en cartesiano.

En este caso, mientras que los sensores de fin de carrera no estén pulsados, podremos mover libremente el brazo robótico. En el momento en que algún sensor sea activado, el motor se bloqueará y solo saldrá de ese estado de bloqueo si se cambia de dirección mediante el joystick.

Uno de los problemas que podemos encontrar a la hora controlar los motores una vez llegue al final de carrera, es el de poder detener la actividad de los mismos, ya que habrá que tener en cuenta varios factores que veremos a continuación:

- Para ello, creamos tres variables, con tres funciones distintas: ‘Cont, Cir y Joy’
- Cuando lleguemos al límite activando el final de carrera, se usará ‘cont’.
- Sabiendo en qué dirección se ha bloqueado el motor al activar el final de carrera, tenemos que obligar a nuestro brazo a girar al lado contrario. Para eso usaremos ‘cir’ y ‘joy’. En el mismo código viene explicado con detalle el funcionamiento de cada variable.
- Un error que puede darse es que al girar el joystick en el otro sentido tan solo un momento y de nuevo volvamos a la dirección bloqueada, el motor continuará girando ya que habrían cambiado los estados y de esa forma, sobrepasaría el final de carrera pudiendo romper la estructura.

De esta forma se consigue que funcione a la perfección los sensores fin de carrera.

```
1.  void Manual() {
2.    Serial.end();        //Cerramos comunicacion serial para no interferir en los
    movimientos
3.    estado1 = digitalRead(9);  // Leemos estado del boton del Joystick
4.    //Hacemos un contador con cada pulso de boton para diferenciar los estados
    autonomo o manual
5.    if (estado1 != estadoAnt1 ) {
6.      if (estado1 == 1) {        //Si boton pulsado -> contador suma 1
7.        contador++;
8.        delay(500);
9.        if (contador == 2) {     //Si contador = 2 se reincia el contador
10.         contador = 0;}}} estadoAnt1 = estado1;
11.   if (contador == 1) {
```

```
12.     ////////////////////// FIN DE CARRERA 0 ////////////////////////////
13.     //Si final de carrera no pulsado -> Mover joystick libremente
14.     while (digitalRead(home_switch0)) {
15.       cir0 = 0; cont0 = 0;                    //Contadores para saber en que
    direccion va
16.       ValueQ0 = analogRead(Joystick0);        //Lee valor Joystick
17.       if (ValueQ0 > 510) { cont0 = 0;} //Si Joystick derecha -> cont = 0
18.       else if (ValueQ0 < 500) { cont0 = 1;} //Si Joystick izquierda -> cont = 1
19.
20.       ValueQ0 = analogRead(Joystick0);        //Lee valor Joystick
21.       MapX = map(ValueQ0, 0, 1023, MinSpeed, MaxSpeed);  //Transforma lectura
    del Joystick
22.       MotorControl0(MapX);              //Funcion que activara al motor
23.       break;}                           //Salimos del bucle
24.      //Si final de carrera activado -> Significa que ha llegado al limite en
    una u otra direccion
25.     //Se bloqueará el motor.
26.     while (!digitalRead(home_switch0)) {
27.       //Creamos 'cir' para marcar la direccion inversa de la que llegabamos al
    final de carrera
28.       ValueQ0 = analogRead(Joystick0);
29.       //Si Joystick Derecha ->  cont == 1
30.       if (ValueQ0 > 510 && cont0 == 1) { cir0 = 1;} //Si viene de la derecha,
    deberiamos inclinar joystick a izquierda y 'cir = 1'
31.       //Si Joystick Izquierda ->  cont == 0
32.       else if (ValueQ0 < 500 && cont0 == 0) { cir0 = 2;} //Si viene de la
    izquierda, deberiamos inclinar joystick a derecha y 'cir = 2'
33.       //Ahora necesitamos asegurar que no se cometerá el error de sobrepasar
    los limites del final de carrera
34.       //Creamos la variable 'joy' que es parecido a 'cir' pero para que se
    active joy, cir debe estarlo, asi aseguramos que saldremos del final de carrera
35.       //Si Joystick reincide la direccion -> bloqueamos esa direccion
36.       if ( ValueQ0 > 600 && cir0 == 1) { joy0 = 1;}
37.       else if ( ValueQ0 < 400 && cir0 == 2 ) { joy0 = 2;}
38.       //Una vez se cumpla todo, el programa solo te dejará moverte en la
    direccion contraria a la que se bloqueo//
39.       // Mover solo en la direccion no bloqueada //
40.       if ((cont0 == 1 && cir0 == 1 && joy0 == 1 ) || cont0 == 0 && cir0 == 2 &&
    joy0 == 2  ) {
41.         ValueQ0 = analogRead(Joystick0);      //Lee valor Joystick
42.         MapX = map(ValueQ0, 0, 1023, MinSpeed, MaxSpeed); //Transforma lectura
    del Joystick
43.         MotorControl0(MapX);
44.         joy0 = 0;
45.       } break;}                         //Salimos del bucle
46.     ////////////////////// FIN DE CARRERA 1 ////////////////////////////
47.     //Si final de carrera no pulsado -> Mover joystick libremente
48.     while (digitalRead(home_switch1)) {
49.       cir1 = 0; cont1 = 0;
50.       ValueQ1 = analogRead(Joystick1);        //Lee valor Joystick 1
51.       if (ValueQ1 > 510) { cont1 = 0;} //Si Joystick derecha -> cont = 0
52.       else if (ValueQ1 < 500) { cont1 = 1;} //Si Joystick izquierda -> cont=0
53.
54.       ValueQ1 = analogRead(Joystick1);     //Lee valor Joystick
55.       MapY = map(ValueQ1, 0, 1023, MinSpeed, MaxSpeed);  //Transforma lectura
    del Joystick
56.       MotorControl1(MapY);              //Funcion que activara al motor
57.       break;}                           //Salimos del bucle
58.      //Si final de carrera pulsado -> Solo se podra mover en direccion
    contraria
59.     while (!digitalRead(home_switch1)) {
60.       //Creamos 'cir' para marcar la direccion inversa de la que llegabamos al
    final de carrera
61.       ValueQ1 = analogRead(Joystick1);        //Lee valor Joystick 1
62.       //Si Joystick Derecha ->  cont == 1
63.       if (ValueQ1 > 510 && cont1 == 1) { cir1 = 1;} //Si viene de la derecha,
    deberiamos inclinar joystick a izquierda y 'cir = 1'
64.       //Si Joystick Izquierda ->  cont == 1
65.       else if (ValueQ1 < 500 && cont1 == 0) { cir1 = 2;} //Si viene de la
    izquierda, deberiamos inclinar joystick a derecha y 'cir = 2'
```

```
66.       //Ahora necesitamos asegurar que no se cometerá el error de sobrepasar
   los limites del final de carrera
67.       //Creamos la variable 'joy' que es parecido a 'cir' pero para que se
   active joy, cir debe estarlo, asi aseguramos que saldremos del final de carrera
68.       //Si Joystick reincide la direccion -> bloqueamos esa direccion
69.       if ( ValueQ1 > 600 && cir1 == 1) { joy1 = 1;}
70.       else if ( ValueQ1 < 400 && cir1 == 2 ) { joy1 = 2;}
71.        //Una vez se cumpla todo, el programa solo te dejará moverte en la
   direccion contraria a la que se bloqueo//
72.        // Mover solo en la direccion no bloqueada //
73.       if ((cont1 == 1 && cir1 == 1 && joy1 == 1) || cont1 == 0 && cir1 == 2 &&
   joy1 == 2 ) {
74.         ValueQ1 = analogRead(Joystick1);   //Lee valor Joystick
75.         MapY = map(ValueQ1, 0, 1023, MinSpeed, MaxSpeed); //Transforma lectura
   del Joystick
76.         MotorControl1(MapY);
77.         joy1 = 0;
78.       } break;}                          //Salimos del bucle
79.     ////////////////////// FIN DE CARRERA 2 ////////////////////////////
80.     //Si final de carrera no pulsado -> Mover joystick libremente
81.     while (digitalRead(home_switch2)) {
82.       cir2 = 0; cont2 = 0;
83.       ValueQ2 = analogRead(Joystick2);        //Lee valor Joystick 1
84.       if (ValueQ2 > 510) { cont2 = 0;} //Si Joystick derecha -> cont = 0
85.       else if (ValueQ2 < 500) { cont2 = 1;} //Si Joystick izquierda -> cont=0
86.
87.       ValueQ2 = analogRead(Joystick2);        //Lee valor Joystick
88.       MapZ = map(ValueQ2, 0, 1023, MinSpeed, MaxSpeed);  //Transforma lectura
   del Joystick
89.       MotorControl2(MapZ);            //Funcion que activara el motor
90.       break;}                         //Salimos del bucle
91.     //Si final de carrera pulsado -> Solo se podra mover en direccion contraria
92.     while (!digitalRead(home_switch2)) {
93.       ValueQ2 = analogRead(Joystick2);        //Lee valor Joystick 1
94.       //Si Joystick Derecha ->  cont == 1
95.       if (ValueQ2 > 510 && cont2 == 1) { cir2 = 1;} //Si viene de la derecha,
   deberiamos inclinar joystick a izquierda y 'cir = 1'
96.       //Si Joystick Izquierda ->  cont == 1
97.       else if (ValueQ2 < 500 && cont2 == 0) { cir2 = 2;} //Si viene de la
   izquierda, deberiamos inclinar joystick a derecha y 'cir = 2'
98.       //Ahora necesitamos asegurar que no se cometerá el error de sobrepasar
   los limites del final de carrera
99.       //Creamos la variable 'joy' que es parecido a 'cir' pero para que se
   active joy, cir debe estarlo, asi aseguramos que saldremos del final de carrera
100.            //Si Joystick reincide la direccion -> bloqueamos esa direccion
101.            if ( ValueQ2 > 600 && cir2 == 1) { joy2 = 1;}
102.            else if ( ValueQ2 < 400 && cir2 == 2 ) { joy2 = 2;}
103.            //Una vez se cumpla todo, el programa solo te dejará moverte en la
   direccion contraria a la que se bloqueo//
104.            // Mover solo en la direccion no bloqueada //
105.            if ((cont2 == 1 && cir2 == 1 && joy2 == 1 ) || cont2 == 0 &&
   cir2 == 2 && joy2 == 2 ) {
106.              ValueQ2 = analogRead(Joystick2);                  //Lee valor
   Joystick
107.              MapZ = map(ValueQ2, 0, 1023, MinSpeed, MaxSpeed); //Transforma
   lectura del Joystick
108.              MotorControl2(MapZ);
109.              joy2 = 0;} break;}}
110.
111.       else if (contador == 0) {
112.         ////////////////////////// HORIZONTAL ////////////////////////////////
113.          //Si final de carrera no pulsado -> Se mueve con Joystick
114.         while (digitalRead(home_switch0) && digitalRead(home_switch1) &&
   digitalRead(home_switch2)) {
115.           cir0 = 0, cont0 = 0, cir1 = 0, cont1 = 0, cir2 = 0, cont2 = 0;
116.           ValueQ2 = analogRead(Joystick2);
117.           if (ValueQ2 > 510) { cont0 = 0, cont1 = 0, cont2=0;}
118.           else if (ValueQ2 < 500) { cont0 = 1, cont1 = 1 , cont2 = 1;}
119.           ValueQ2 = analogRead(Joystick2);            //Lee valor Joystick
```

```
120.            MapZ = map(ValueQ2, 0, 1023, MinSpeed, MaxSpeed); //Transforma
    lectura del Joystick
121.            MotorControl2(MapZ);    //Mover q2 a velocidad normal
122.            Control1(MapZ);         //Mover q1 a velocidad reducida
123.            break;} estadoAnt1 = estado1;
124.
125.          while (!digitalRead(home_switch0) || !digitalRead(home_switch1) ||
    !digitalRead(home_switch2)) {
126.            ValueQ2 = analogRead(Joystick2);
127.            if (ValueQ2 > 510 && cont0 == 1 && cont1 == 1 &&
    cont2 == 1) { cir0 = 1, cir1 = 1, cir2 = 1;}
128.            else if (ValueQ2 < 500 && cont0 == 0 && cont1 == 0 &&
    cont2 == 0) { cir0 = 2, cir1 = 2, cir2 = 2;}
129.
130.            if ( ValueQ2 > 600 && cir0 == 1 && cir1 == 1 &&
    cir2 == 1) { joy0 = 1, joy1 = 1, joy2 = 1;}
131.            else if ( ValueQ2 < 400 && cir0 == 2 && cir1 == 2 &&
    cir2 == 2 ) { joy0 = 2, joy1 = 2, joy2 = 2;}
132.
133.            if ((cont0 == 1 && cir0 == 1 && joy0 == 1) || cont0 == 0 &&
    cir0 == 2  && joy0 == 2) {
134.              ValueQ2 = analogRead(Joystick2);           //Lee valor Joystick
135.              MapZ = map(ValueQ2, 0, 1023, MinSpeed, MaxSpeed);  //Transforma
    lectura del Joystick
136.              MotorControl2(MapZ);        //Mover q2 a velocidad normal
137.              Control1(MapZ);             //Mover q1 a velocidad reducida
138.              joy0 = 0, joy1 = 0,joy2 = 0;    } break;}
139.      ////////////////////////////// HORIZONTAL ////////////////////////////////
140.        while (digitalRead(home_switch0) && digitalRead(home_switch1) &&
    digitalRead(home_switch2)) {
141.            cir0 = 0, cont0 = 0, cir1 = 0, cont1 = 0, cir2 = 0, cont2 = 0;
142.            ValueQ0= analogRead(Joystick0);
143.            if (ValueQ0 > 510) { cont0 = 0, cont1 = 0, cont2=0;}
144.            else if (ValueQ0 < 500) { cont0 = 1, cont1 = 1 , cont2 = 1;}
145.            ValueQ0 = analogRead(Joystick0);  //Lee valor Joystick
146.            MapX = map(ValueQ0, 0, 1023, MinSpeed, MaxSpeed); //Transforma
    lectura del Joystick
147.
148.            MotorControl0(MapX);              //Mover q0 a velocidad normal
149.            Control1(MapX);                   //Mover q1 a velocidad reducida
150.            MotorControl2(MapX);              //Mover q2 a velocidad reducida
151.            break;} estadoAnt1 = estado1;
152.
153.          while (!digitalRead(home_switch0) || !digitalRead(home_switch1) ||
    !digitalRead(home_switch2)) {
154.            ValueQ0 = analogRead(Joystick0);
155.            if (ValueQ0 > 510 && cont0 == 1 && cont1 == 1 &&
    cont2 == 1) { cir0 = 1, cir1 = 1, cir2 = 1;}
156.            else if (ValueQ0 < 500 && cont0 == 0 && cont1 == 0 &&
    cont2 == 0) { cir0 = 2, cir1 = 2, cir2 = 2;}
157.
158.            if ( ValueQ0 > 600 && cir0 == 1 && cir1 == 1 &&
    cir2 == 1) { joy0 = 1, joy1 = 1, joy2 = 1;}
159.            else if ( ValueQ0 < 400 && cir0 == 2 && cir1 == 2 &&
    cir2 == 2 ) { joy0 = 2, joy1 = 2, joy2 = 2;}
160.
161.            if ((cont0 == 1 && cir0 == 1 && joy0 == 1) || cont0 == 0 &&
    cir0 == 2  && joy0 == 2) {
162.              ValueQ0 = analogRead(Joystick0);   //Lee valor Joystick
163.              MapX = map(ValueQ0, 0, 1023, MinSpeed, MaxSpeed);  //Transforma
    lectura del Joystick
164.
165.              MotorControl0(MapX);          //Mover q0 a velocidad normal
166.              Control1(MapX);               //Mover q1 a velocidad reducida
167.              MotorControl2(MapX);          //Mover q2 a velocidad normal
168.              joy0 = 0, joy1 = 0,joy2 = 0;
169.            } break;}}}
```

6 CONCLUSIONES Y MEJORAS FUTURAS

Al principio de este proyecto se propusieron unos objetivos a alcanzar en este proyecto en cuanto a la funcionalidad y la especialización de nuestro brazo articulado. Hablábamos de conseguir un movimiento de las articulaciones de manera independendiente y tambien dependiendo unas de otras. Hay que indicar que se han alcanzado los objetivos que se propusieron para este proyecto que encontrábamos en el primer capítulo.

6.1 Objetivos

Han sido abundantes las conclusiones obtenidas durante la realización del Trabajo Fin de Grado tratando temas como el diseño y fabricación del brazo, hasta la tarea de programar el mismo.

- Objetivo principal, funcionalidad: crear dos estados de movimientos, modo manual, mediante el uso de joysticks, y el modo autónomo, mandando órdenes por puerto serie para el movimiento en coordenadas. El objetivo se alcanza, pero no será el resultado final considerado, ya que nuestra intención es sacar el máximo rendimiento a nuestro proyecto.
- Objetivos secundarios: mover en coordenadas cartesianas mediante joystick y completar una trayectoria de varios puntos consiguiendo formar una letra en mayúscula y dibujarla.
- Diseño del brazo en madera, usando otro material más resistente como el metacritlato para los engranajes para alargar la vida a los mismos, ya que estarán siempre en continuo rozamiento.

6.2 Mejoras futuras

El proyecto está pensado para reforzar los conocimientos de los alumnos en el control y la programación de brazos manipuladores de una forma económica, ya que se hace en madera principalmente y siendo el coste mayor el de los motores.

Aunque los objetivos se alcancen, a nivel mecánico nuestro brazo se puede mejorar mucho más, ya que a veces, el movimiento no es todo lo limpio que debería y pueden bloquear los motores debido al par engranaje-piñon que va perdiendo material mediante el roce y perderá juego, aumentando la distancia entre dientes. Este juego en el mecanismo podría perjudicar que, al cambiar la dirección del movimiento, los dientes choquen bloqueando el movimiento del motor.

A continuación, se enumerará algunas tareas que podrán mejorar nuestro tfg.

- Uso de la dinámica para perfeccionar el movimiento de nuestro robot. Esta mejora es una tarea amplia y compleja en la que se podría profundizar en una futura mejora, aportando mejores competencias al proyecto.
- Añadir sensores para leer la posición del efector final en cada momento, encoders para las velocidades, sensores de distancia ultrasónicos.
- Mejorar el mecanismo con el que cada motor mueve cada elemento para que el movimiento sea más limpio.

REFERENCIAS

[1] OLLERO, A. (2001) *RÓBOTICA. Manipuladores y robots móviles*. Barcelona: S.A. Marcombo.

[2] *https://www.arduino.cc*

[3] *https://es.wikipedia.org/*

[4] Diseño estructura del brazo: *http://fablab.ruc.dk/robot-arm-v-0-1/*

[5] Librería AccelStepper: *https://www.airspayce.com/mikem/arduino/AccelStepper/*

[6] *https://www.arduino.cc*

[7] *https://www.arduino.cc*

ANEXO A: CÓDIGO

```
170.      #include "AccelStepper.h"
171.
172.      #define n 10        //Tamaño maximo vector posiciones
173.      #define d 5         //Tiempo del movimiento
174.      // Pines direccion y paso de los drivers
175.      #define Step0 3
176.      #define Dir0 2
177.      #define Step1 5
178.      #define Dir1 4
179.      #define Step2 7
180.      #define Dir2 6
181.      // Pines reservados para los finales de carrera
182.      #define home_switch0 10
183.      #define home_switch1 11
184.      #define home_switch2 12
185.      //Configurar Accelstepper. Step y Dir seran los pines
186.      AccelStepper Motor0(1, Step0, Dir0);
187.      AccelStepper Motor1(1, Step1, Dir1);
188.      AccelStepper Motor2(1, Step2, Dir2);
189.      unsigned long auxtimer;
190.      int estado = 0, estadoAnt = 0, salida = 0, cont = 0, contador = 0,
   estado1 = 0, estadoAnt1 = 0;
191.      boolean MoveX, MoveY, MoveZ, EnableX;
192.      char data;
193.      float mov1, mov2, mov3;
194.      long lastPositionX, lastPositionY, lastPositionZ, thisPositionX,
   thisPositionY,  thisPositionZ;
195.      float vx, vy, vz, a, b;
196.      float Ji[3][3] = {
197.        { 0, 0, 0 },
198.        { 0, 0, 0 },
199.        { 0, 0, 0 },};
200.      float vq1, vq2, vq3, aq1, aq2, aq3, dmv, xdes, ydes, zdes;
201.      const float pi = 3.1415;
202.      long cir0 = 0, cont0 = 0, cir1 = 0, cont1 = 0, cir2 = 0, cont2 = 0,
   joy0 = 0, joy1 = 0, joy2 = 0;
203.      int move_finished = 1;      // comprobar si el movimiento se ha completado
204.      long initial_homing = -1;  // Home Stepper inicio
205.      float l1 = 75, l2 = 142, l3 = 152, ln, q1des, q2des, q3des;
206.      int i, j, FLAG, f, iteracion;
207.      float q1mov[n], q2mov[n], q3mov[n];
208.      float xleida, yleida, zleida, xini = 120, yini = 120, zini = 20;
209.      float q1ini = 45, q2ini = 37.28, q3ini = -105.37;
210.      float q1 = 45 * (pi / 180), q2 = 37.28 * (pi / 180), q3 = -105.37 * (pi
   / 180);
211.      float r, alfa, beta;
212.      float qx, qy, qz, v1, v2, v3, c3; //cos(q3)
213.      float rt = 7.75; // relacion de trasmision 8/62
214.      float rg = 1600 / 360; // relacion vueltas/grados
215.      const int Joystick0 = A0, Joystick1 = A1, Joystick2 = A2;
216.      long SpeedQ1, SpeedQ2, SpeedQ3, MapX, MapY, MapZ, ValueQ0, ValueQ1,
   ValueQ2, UmbralUp, UmbralDown;
217.      const int MaxSpeed = 500, MinSpeed = 0, Treshold = 50, MaxSpeed1 = 200,
   MinSpeed1 = 0;
218.      const float Aceleration = 400.0;
219.      ////////////////////////// SETUP ////////////////////////////////
220.      void setup() {
221.        // Activamos el puerto serie
222.        Serial.begin(9600);
```

```
223.        // Declaramos: pin 8 boton del joystick izquierdo, pin 9 boton joystick
    derecho
224.        pinMode(8, INPUT_PULLUP);
225.        pinMode(9, INPUT_PULLUP);
226.        // Asignamos los pines para cada final de carrera
227.        pinMode(home_switch0, INPUT_PULLUP);
228.        pinMode(home_switch1, INPUT_PULLUP);
229.        pinMode(home_switch2, INPUT_PULLUP);
230.        delay(15);
231.        // Asignar la velocidad y aceleracion de cada motor en 'home'
232.        Motor0.setAcceleration(2000);
233.        Motor1.setAcceleration(1000);
234.        Motor2.setAcceleration(1000);
235.        Motor0.setMaxSpeed(1000);
236.        Motor1.setMaxSpeed(400);
237.        Motor2.setMaxSpeed(400);
238.        ////////////////////////////////////////////////////////////////
239.        Serial.print("Stepper is Homing . . . . . ");
240.        while (digitalRead(home_switch0)) {
241.        Motor0.moveTo(initial_homing);    //Mover Stepper hasta activar el
    final de carrera
242.          initial_homing++;     //Incrementar en 1 hasta la siguiente posicion
243.          Motor0.run();         //Activa el movimiento del Stepper
244.          delay(5);
245.        }
246.        while (digitalRead(home_switch1)) {
247.          Motor1.moveTo(-initial_homing);    //Mover Stepper hasta activar el
    final de carrera
248.          initial_homing++;      //Incrementar en 1 hasta la siguiente posicion
249.          Motor1.run();          //Activa el movimiento del Stepper
250.          delay(5);
251.        }
252.        while (digitalRead(home_switch2)) {
253.          Motor2.moveTo(-initial_homing);    //Mover Stepper hasta activar el
    final de carrera
254.          initial_homing++;    //Incrementar en 1 hasta la siguiente posicion
255.          Motor2.run();        //Activa el movimiento del Stepper
256.          delay(5);}
257.
258.        initial_homing = 1;
259.        // Si se alcanzan los sensores los motores giran hasta separarse en
    sentido contrario
260.        while (!digitalRead(home_switch0)) {
261.          Motor0.moveTo(initial_homing);
262.          Motor0.run();
263.          initial_homing--;
264.          delay(5);
265.        }
266.        while (!digitalRead(home_switch1)) {
267.          Motor1.moveTo(-initial_homing);
268.          Motor1.run();
269.          initial_homing--;
270.          delay(5);
271.        }
272.        while (!digitalRead(home_switch2)) {
273.          Motor2.moveTo(-initial_homing);
274.          Motor2.run();
275.          initial_homing--;
276.          delay(5);
277.        }
278.        // Imprime por pantalla //
279.        Serial.println("Homing Completed");
280.        Serial.println("");
281.        Serial.print("Elija posicion: 1, 2 ,3: ");
282.        Serial.println(" ");
283.        // LLegado a 'home' los motores se asignarán a la posicion 0
284.        Motor0.setCurrentPosition (0);
285.        Motor1.setCurrentPosition (0);
286.        Motor2.setCurrentPosition (0);
```

```
287.        // Declaramos valores para el modo manual //
288.        SpeedQ1 = 0, SpeedQ2 = 0, SpeedQ3 = 0;
289.        // Umbral para Joystick
290.        UmbralDown = (MaxSpeed / 2) - Treshold;
291.        UmbralUp = (MaxSpeed / 2) + Treshold;
292.
293.        Motor0.setMaxSpeed(MaxSpeed);
294.        Motor0.setSpeed(MinSpeed);
295.        Motor0.setAcceleration(Aceleration);
296.        Motor1.setMaxSpeed(MaxSpeed);
297.        Motor1.setSpeed(MinSpeed);
298.        Motor1.setAcceleration(Aceleration);
299.        Motor2.setMaxSpeed(MaxSpeed);
300.        Motor2.setSpeed(MinSpeed);
301.        Motor2.setAcceleration(Aceleration);
302.      }
303.      ////////////////////////////// LOOP ////////////////////////////////
304.      void loop() {
305.
306.        estado = digitalRead(8);  // Leemos estado del boton del Joystick
307.        //Hacemos un contador con cada pulso de boton para diferenciar los
    estados autonomo o manual
308.        if (estado != estadoAnt ) {
309.          if (estado == 1) {        //Si boton pulsado -> contador suma 1
310.            cont++;
311.            delay(500);
312.            if (cont == 2) {        //Si contador = 2 se reincia el contador
313.              cont = 0;}
314.          }} estadoAnt = estado;
315.
316.        if (Serial.available() > 0) {
317.          if (cont == 1) {  // Como cont = 1 -> Modo autonomo
318.          if(digitalRead(home_switch0) && digitalRead(home_switch1) &&
    digitalRead(home_switch2)) {
319.            EligeLetra();}
320.           }
321.        } else if (cont == 0) {   // Si cont=0 -> Modo Manual
322.          Manual();}
323.        }
324.      ////////////////////////////////////////////////////////////////
325.      // Elegimos letra desde el puerto serial //
326.      void EligeLetra() {
327.        if (Serial.available() > 0) {   // Si el puerto serie esta 'abierto'
328.          data = Serial.read();         //Leemos valor introducido
329.          if (data == 'H', 'h') {       //Si coincide con H
330.            Letra_H();                  //Funcion que recorre las posiciones para H
331.            delay(200);
332.          }Serial.println(data);}        //Mostramos Letra por serial
333.      }
334.      ////////////////////////////////////////////////////////////////////////
335.      // Modelo matematico cinematica inversa //
336.      void CalculaInversa()
337.      {
338.        r = sqrt(pow(xleida, 2) + pow(yleida, 2));
339.        ln = zleida - l1;
340.        c3 = ((pow(xleida, 2) + pow(yleida, 2) + pow(ln, 2) - pow(l2, 2) - pow(
    l3, 2)) / (2 * l2 * l3));
341.        q1des = atan(yleida / xleida);
342.        q2des = atan(ln / r) - atan(l3 * sin(q3) / (l2 + l3 * c3));
343.        q3des = atan(-(sqrt(1 - pow(c3, 2))) / c3);
344.
345.        q1mov[j] = q1des * (180 / pi);        //Giro de la base en Grados
346.        q2mov[j] = q2des * (180 / pi);        //Giro de la base en Grados
347.        q3mov[j] = q3des * (180 / pi) - 180;  //Giro de la base en Grados
348.
349.        if (xleida > 185) {
350.        q3mov[j] = q3des * (180 / pi);}}
351.      ////////////////////////////////////////////////////////////////////////
352.      // Calcular jacobiano Inverso para uso de velocidades //
353.      void JacInverso() {
```

```
354.        //a y b son variables auxiliares para tener el calculo mas facil
355.        a = pow(xleida, 2) + pow(yleida, 2) + pow(zleida, 2) - pow(l2, 2) - pow
    (l3, 2);
356.        b = pow(xleida, 2) + pow(yleida, 2);
357.
358.        Ji[0][0] = -(yleida) / (pow(xleida, 2) + pow(yleida, 2));
359.        Ji[0][1] = (xleida) / (pow(xleida, 2) + pow(yleida, 2));
360.        Ji[0][2] = 0 ;
361.
362.        Ji[1][0] = -xleida * ((2 * l2 * l3 * zleida * pow(xleida, 2) + 2 * l2 *
    l3 * zleida * pow(yleida, 2)) *
    sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2)))) + (pow(l2, 2) - pow(xleid
    a, 2) - pow(zleida, 2) - pow(yleida, 2) - pow(l3, 2)) * pow(b, 1.5)) / (2 * l2
    * l3 * (pow(b, 1.5)) * (pow(xleida, 2) + pow(yleida, 2) + pow(zleida, 2)) *
    sqrt(1 - (pow(a, 2)) / (4 * pow(l2, 2) * pow(l3, 2))));
363.        Ji[1][1] = -yleida * ((2 * l2 * l3 * zleida * pow(xleida, 2) + 2 * l2 *
    l3 * zleida * pow(yleida, 2)) *
    sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2)))) + (pow(l2, 2) - pow(xleid
    a, 2) - pow(zleida, 2) - pow(yleida, 2) - pow(l3, 2)) * pow(b, 1.5)) / (2 * l2
    * l3 * (pow(b, 1.5)) * (pow(xleida, 2) + pow(yleida, 2) + pow(zleida, 2)) *
    sqrt(1 - (pow(a, 2)) / (4 * pow(l2, 2) * pow(l3, 2))));
364.        Ji[1][2] =  ((2 * l2 * l3 * pow(yleida, 2) + 2 * l2 * l3
    * pow(xleida, 2)) * sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2)))) +
    sqrt(pow(xleida, 2) + pow(yleida, 2)) * pow(zleida, 3) +
    sqrt(pow(yleida, 2) + pow(xleida, 2)) * (pow(yleida, 2) + pow(xleida, 2) + pow(
    l3, 2) - pow(l2, 2)) * zleida) / (2 * l2 * l3 *
    sqrt(pow(yleida, 2) + pow(xleida, 2)) * (pow(zleida, 2) + pow(yleida, 2) + pow(
    xleida, 2)) * sqrt(1 - ((pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2))))));
365.
366.        Ji[2][0] = -xleida / (l2 * l3
    * (sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2))))));
367.        Ji[2][1] = -yleida / (l2 * l3
    * (sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2))))));
368.        Ji[2][2] = -zleida / (l2 * l3
    * (sqrt(1 - (pow(a, 2) / (4 * pow(l2, 2) * pow(l3, 2))))));
369.      }
370.      ////////////////////////////////////////////////////////////////////////
371.      void Trayectoria()
372.      { //Encontrar Trayectoria
373.        qx = (q1mov[j] - q1ini);
374.        qy = (q2mov[j] - q2ini);
375.        qz = (q3mov[j] - q3ini);
376.        ActualizarPos();
377.      }
378.      ////////////////////////////////////////////////////////
379.      void ActualizarPos() {  //Necesario ya que nuestro movimiento será
    absoluto
380.        //Actualizar Trayectoria
381.        q1ini = q1mov[j];
382.        q2ini = q2mov[j];
383.        q3ini = q3mov[j];
384.        xini = xleida;
385.        yini = yleida;
386.        zini = zleida;
387.
388.        if (Motor0.distanceToGo() == 0 && Motor1.distanceToGo() == 0 &&
    Motor2.distanceToGo() == 0) {
389.          Motor0.setCurrentPosition(0);
390.          Motor1.setCurrentPosition(0);
391.          Motor2.setCurrentPosition(0);}
392.      }
393.      ////////////////////////////////////////////////////////////////////////
394.      void Letra_H() {
395.        i = 0;
396.        while (1) {
397.          // Vector de posiciones a recorrer //
398.          int ArrayX[8] = {170, 170, 200, 185, 185, 200, 170, 170};
399.          int ArrayY[8] = {0, 0, 0, 0, -40, -40, -40, -40};
400.          int ArrayZ[8] = {40,0, 0, 0, 0, 0, 0, 40};
```

```
401.
402.          if (i < 8) {
403.            xleida = ArrayX[i];
404.            yleida = ArrayY[i];
405.            zleida = ArrayZ[i];
406.            CalculaInversa();
407.            delay(100);
408.            JacInverso();
409.            Mover();
410.            Movimiento();
411.
412.            Serial.println(q1mov[j]);
413.            Serial.println(mov1);
414.            Serial.println(q2mov[j]);
415.            Serial.println(mov2);
416.            Serial.println(q3mov[j]);
417.            Serial.println(mov3);
418.            Serial.println(" ");
419.
420.       // Se comprueba si ha alcanza la posicion  para pasar a la siguiente //
421.            if (Motor0.currentPosition() == int(mov1) &&
   Motor1.currentPosition() == int(-mov2) &&
   Motor2.currentPosition() == int(mov3)) {
422.              i++;
423.              delay(1500);}}
424.          if (i == 8) {
425.            break;}}
426.      }
427.      ////////////////////////////////////////////////////////////////////////
428.      void MoverMotores() {
429.        // Previamente calculadas: Asignamos velocidad, aceleracion y pasos a
   desplazar //
430.        Motor0.setMaxSpeed(v1);          //Mover Motor
431.        Motor0.setAcceleration(aq1 * 10);//Velocidad de movimiento del motor
432.        Motor0.moveTo(mov1);            //Aceleracion de movimiento del motor
433.
434.        Motor1.setMaxSpeed(v2);           //Mover Motor
435.        Motor1.setAcceleration(aq2 * 10); //Velocidad de movimiento del motor
436.        Motor1.moveTo(-mov2);            //Aceleracion de movimiento del motor
437.
438.        Motor2.setMaxSpeed(v3);            //Mover Motor
439.        Motor2.setAcceleration(aq3 * 10);  //Velocidad de movimiento del motor
440.        Motor2.moveTo(mov3);             //Aceleracion de movimiento del motor
441.
442.        ActivarMotores();}
443.      ////////////////////////////////////////////////////////////////////////
444.      void Movimiento() {
445.        // Se comprueba si ha alcanza la posicion para pasar a la siguiente se
   activan los motores
446.        while (Motor0.distanceToGo() || Motor1.distanceToGo() ||
   Motor2.distanceToGo()) {
447.          ActivarMotores();
448.          // CurrentPosition() asigna la posicion del stepper en cada momento
449.          long thisPositionX = Motor0.currentPosition();
450.          long thisPositionY = Motor1.currentPosition();
451.          long thisPositionZ = Motor2.currentPosition();
452.          if (thisPositionX != lastPositionX || thisPositionY != lastPositionY
   || thisPositionZ != lastPositionZ)
453.          { //Si se alcanzan las posiciones, se actualizan las mismas
454.            lastPositionX = thisPositionX;
455.            lastPositionY = thisPositionY;
456.            lastPositionZ = thisPositionZ;}}
457.      }
458.      ////////////////////////////////////////////////////////////////////////
459.      void ActivarMotores() {
460.        //Activa motores//
461.        Motor0.run();
462.        Motor1.run();
463.        Motor2.run();
464.      }
```

```
465.      ////////////////////////////////////////////////////////////////////
466.      void Mover() {
467.        Trayectoria();
468.        //Declarar una velocidad lineal
469.        vx = 100;
470.        vy = 100;
471.        vz = 100;
472.        //Velocidades calculadas por Jacobiano Inverso
473.        vq1 = Ji[0][0] * vx + Ji[0][1] * vy + Ji[0][2] * vz;
474.        vq2 = Ji[1][0] * vx + Ji[1][1] * vy + Ji[1][2] * vz;
475.        vq3 = Ji[2][0] * vx + Ji[2][1] * vy + Ji[2][2] * vz;
476.        //Aplicar relaciones de transmision y de pasos a las velocidades
477.        v1 = vq1 * rt * rg * 10;
478.        v2 = vq2 * rt * rg * 10;
479.        v3 = vq3 * rt * rg * 10;
480.        //Aceleracion sacada desde la velocidad
481.        aq1 = (v1 / d) * rt * rg;
482.        aq2 = (v2 / d) * rt * rg;
483.        aq3 = (v3 / d) * rt * rg;
484.        //Movimiento de cada motor
485.        mov1 = qx * rt * rg;
486.        mov2 = qy * rt * rg;
487.        mov3 = qz * rt * rg;
488.        MoverMotores();}
489.      ////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////////
490.      void Manual() {
491.        Serial.end();        //Cerramos comunicacion serial para no interferir
   en los movimientos
492.        estado1 = digitalRead(9);  // Leemos estado del boton del Joystick
493.        //Hacemos un contador con cada pulso de boton para diferenciar los
   estados autonomo o manual
494.        if (estado1 != estadoAnt1 ) {
495.          if (estado1 == 1) {        //Si boton pulsado -> contador suma 1
496.            contador++;
497.            delay(500);
498.            if (contador == 2) {     //Si contador = 2 se reincia el contador
499.              contador = 0;}}} estadoAnt1 = estado1;
500.
501.        if (contador == 1) {
502.          ////////////////////// FIN DE CARRERA 0 ////////////////////////////
503.          //Si final de carrera no pulsado -> Mover joystick libremente
504.          while (digitalRead(home_switch0)) {
505.            cir0 = 0; cont0 = 0;    //Contadores para saber en que direccion va
506.            ValueQ0 = analogRead(Joystick0);      //Lee valor Joystick
507.            if (ValueQ0 > 510) { cont0 = 0;} //Si Joystick derecha -> cont = 0
508.            else if (ValueQ0 < 500) { cont0 = 1;} //Si Joystick izquierda ->
   cont = 1
509.            ValueQ0 = analogRead(Joystick0);        //Lee valor Joystick
510.            MapX = map(ValueQ0, 0, 1023, MinSpeed, MaxSpeed);  //Transforma
   lectura del Joystick
511.            MotorControl0(MapX);              //Funcion que activara al motor
512.            break;}                           //Salimos del bucle
513.      //Si final de carrera activado -> Significa que ha llegado al limite en
   una u otra direccion
514.          //Se bloqueará el motor.
515.          while (!digitalRead(home_switch0)) {
516.      //Creamos 'cir' para marcar la direccion inversa de la que llegabamos al
   final de carrera
517.            ValueQ0 = analogRead(Joystick0);
518.            //Si Joystick Derecha ->  cont == 1
519.            if (ValueQ0 > 510 && cont0 == 1) { cir0 = 1;} //Si viene de la
   derecha, deberiamos inclinar joystick a izquierda y 'cir = 1'
520.            //Si Joystick Izquierda ->  cont == 0
521.            else if (ValueQ0 < 500 && cont0 == 0) { cir0 = 2;} //Si viene de la
   izquierda, deberiamos inclinar joystick a derecha y 'cir = 2'
522.            //Ahora necesitamos asegurar que no se cometerá el error de
   sobrepasar los limites del final de carrera
```

```
523.            //Creamos la variable 'joy' que es parecido a 'cir' pero para que
    se active joy, cir debe estarlo, asi aseguramos que saldremos del final de
    carrera
524.            //Si Joystick reincide la direccion -> bloqueamos esa direccion
525.            if ( ValueQ0 > 600 && cir0 == 1) { joy0 = 1;}
526.            else if ( ValueQ0 < 400 && cir0 == 2 ) { joy0 = 2;}
527.            //Una vez se cumpla todo, el programa solo te dejará moverte en la
    direccion contraria a la que se bloqueo//
528.            // Mover solo en la direccion no bloqueada //
529.            if ((cont0 == 1 && cir0 == 1 && joy0 == 1 ) || cont0 == 0 &&
    cir0 == 2 && joy0 == 2  ) {
530.              ValueQ0 = analogRead(Joystick0);      //Lee valor Joystick
531.              MapX = map(ValueQ0, 0, 1023, MinSpeed, MaxSpeed); //Transforma
    lectura del Joystick
532.              MotorControl0(MapX);
533.              joy0 = 0;
534.            } break;}                   //Salimos del bucle
535.          ////////////////////////// FIN DE CARRERA 1 ////////////////////////////
536.          //Si final de carrera no pulsado -> Mover joystick libremente
537.          while (digitalRead(home_switch1)) {
538.            cir1 = 0; cont1 = 0;
539.            ValueQ1 = analogRead(Joystick1);        //Lee valor Joystick 1
540.            if (ValueQ1 > 510) { cont1 = 0;} //Si Joystick derecha -> cont = 0
541.            else if (ValueQ1 < 500) { cont1 = 1;} //Si Joystick izquierda ->
    cont=0
542.            ValueQ1 = analogRead(Joystick1);     //Lee valor Joystick
543.            MapY = map(ValueQ1, 0, 1023, MinSpeed, MaxSpeed);  //Transforma
    lectura del Joystick
544.            MotorControl1(MapY);             //Funcion que activara al motor
545.            break;}                          //Salimos del bucle
546.      //Si final de carrera pulsado -> Solo se podra mover en direccion
    contraria
547.          while (!digitalRead(home_switch1)) {
548.      //Creamos 'cir' para marcar la direccion inversa de la que llegabamos al
    final de carrera
549.            ValueQ1 = analogRead(Joystick1);        //Lee valor Joystick 1
550.            //Si Joystick Derecha ->  cont == 1
551.            if (ValueQ1 > 510 && cont1 == 1) { cir1 = 1;} //Si viene de la
    derecha, deberiamos inclinar joystick a izquierda y 'cir = 1'
552.            //Si Joystick Izquierda ->  cont == 1
553.            else if (ValueQ1 < 500 && cont1 == 0) { cir1 = 2;} //Si viene de la
    izquierda, deberiamos inclinar joystick a derecha y 'cir = 2'
554.            //Ahora necesitamos asegurar que no se cometerá el error de
    sobrepasar los limites del final de carrera
555.            //Creamos la variable 'joy' que es parecido a 'cir' pero para que
    se active joy, cir debe estarlo, asi aseguramos que saldremos del final de
    carrera
556.            //Si Joystick reincide la direccion -> bloqueamos esa direccion
557.            if ( ValueQ1 > 600 && cir1 == 1) { joy1 = 1;}
558.            else if ( ValueQ1 < 400 && cir1 == 2 ) { joy1 = 2;}
559.             //Una vez se cumpla todo, el programa solo te dejará moverte en la
    direccion contraria a la que se bloqueo//
560.             // Mover solo en la direccion no bloqueada //
561.            if ((cont1 == 1 && cir1 == 1 && joy1 == 1) || cont1 == 0 &&
    cir1 == 2 && joy1 == 2 ) {
562.              ValueQ1 = analogRead(Joystick1);   //Lee valor Joystick
563.              MapY = map(ValueQ1, 0, 1023, MinSpeed, MaxSpeed); //Transforma
    lectura del Joystick
564.              MotorControl1(MapY);
565.              joy1 = 0;
566.            } break;}                   //Salimos del bucle
567.          ////////////////////////// FIN DE CARRERA 2 ////////////////////////////
568.          //Si final de carrera no pulsado -> Mover joystick libremente
569.          while (digitalRead(home_switch2)) {
570.            cir2 = 0; cont2 = 0;
571.            ValueQ2 = analogRead(Joystick2);        //Lee valor Joystick 1
572.            if (ValueQ2 > 510) { cont2 = 0;} //Si Joystick derecha -> cont = 0
573.            else if (ValueQ2 < 500) { cont2 = 1;} //Si Joystick izquierda ->
    cont=0
574.            ValueQ2 = analogRead(Joystick2);        //Lee valor Joystick
```

```
575.            MapZ = map(ValueQ2, 0, 1023, MinSpeed, MaxSpeed);  //Transforma
    lectura del Joystick
576.            MotorControl2(MapZ);            //Funcion que activara el motor
577.            break;}                         //Salimos del bucle
578.      //Si final de carrera pulsado -> Solo se podra mover en direccion
    contraria
579.          while (!digitalRead(home_switch2)) {
580.            ValueQ2 = analogRead(Joystick2);        //Lee valor Joystick 1
581.            //Si Joystick Derecha ->  cont == 1
582.            if (ValueQ2 > 510 && cont2 == 1) { cir2 = 1;} //Si viene de la
    derecha, deberiamos inclinar joystick a izquierda y 'cir = 1'
583.            //Si Joystick Izquierda ->  cont == 1
584.            else if (ValueQ2 < 500 && cont2 == 0) { cir2 = 2;} //Si viene de la
    izquierda, deberiamos inclinar joystick a derecha y 'cir = 2'
585.            //Ahora necesitamos asegurar que no se cometerá el error de
    sobrepasar los limites del final de carrera
586.            //Creamos la variable 'joy' que es parecido a 'cir' pero para que
    se active joy, cir debe estarlo, asi aseguramos que saldremos del final de
    carrera
587.            //Si Joystick reincide la direccion -> bloqueamos esa direccion
588.            if ( ValueQ2 > 600 && cir2 == 1) { joy2 = 1;}
589.            else if ( ValueQ2 < 400 && cir2 == 2 ) { joy2 = 2;}
590.            //Una vez se cumpla todo, el programa solo te dejará moverte en la
    direccion contraria a la que se bloqueo//
591.            // Mover solo en la direccion no bloqueada //
592.            if ((cont2 == 1 && cir2 == 1 && joy2 == 1 ) || cont2 == 0 &&
    cir2 == 2 && joy2 == 2 ) {
593.              ValueQ2 = analogRead(Joystick2);      //Lee valor Joystick
594.              MapZ = map(ValueQ2, 0, 1023, MinSpeed, MaxSpeed); //Transforma
    lectura del Joystick
595.              MotorControl2(MapZ);
596.              joy2 = 0;} break;}
597.              }
598.
599.        else if (contador == 0) {
600.          ////////////////////////// HORIZONTAL ////////////////////////////////
601.           //Si final de carrera no pulsado -> Se mueve con Joystick
602.          while (digitalRead(home_switch0) && digitalRead(home_switch1) &&
    digitalRead(home_switch2)) {
603.            cir0 = 0, cont0 = 0, cir1 = 0, cont1 = 0, cir2 = 0, cont2 = 0;
604.            ValueQ2 = analogRead(Joystick2);
605.            if (ValueQ2 > 510) { cont0 = 0, cont1 = 0, cont2=0;}
606.            else if (ValueQ2 < 500) { cont0 = 1, cont1 = 1 , cont2 = 1;}
607.            ValueQ2 = analogRead(Joystick2);            //Lee valor Joystick
608.            MapZ = map(ValueQ2, 0, 1023, MinSpeed, MaxSpeed); //Transforma
    lectura del Joystick
609.
610.            MotorControl2(MapZ);    //Mover q2 a velocidad normal
611.            Control1(MapZ);         //Mover q1 a velocidad reducida
612.            break;} estadoAnt1 = estado1;
613.
614.          while (!digitalRead(home_switch0) || !digitalRead(home_switch1) ||
    !digitalRead(home_switch2)) {
615.            ValueQ2 = analogRead(Joystick2);
616.            if (ValueQ2 > 510 && cont0 == 1 && cont1 == 1 &&
    cont2 == 1) { cir0 = 1, cir1 = 1, cir2 = 1;}
617.            else if (ValueQ2 < 500 && cont0 == 0 && cont1 == 0 &&
    cont2 == 0) { cir0 = 2, cir1 = 2, cir2 = 2;}
618.
619.            if ( ValueQ2 > 600 && cir0 == 1 && cir1 == 1 &&
    cir2 == 1) { joy0 = 1, joy1 = 1, joy2 = 1;}
620.            else if ( ValueQ2 < 400 && cir0 == 2 && cir1 == 2 &&
    cir2 == 2 ) { joy0 = 2, joy1 = 2, joy2 = 2;}
621.
622.            if ((cont0 == 1 && cir0 == 1 && joy0 == 1) || cont0 == 0 &&
    cir0 == 2  && joy0 == 2) {
623.              ValueQ2 = analogRead(Joystick2);          //Lee valor Joystick
624.              MapZ = map(ValueQ2, 0, 1023, MinSpeed, MaxSpeed);  //Transforma
    lectura del Joystick
```

```
625.
626.              MotorControl2(MapZ);          //Mover q2 a velocidad normal
627.              Control1(MapZ);               //Mover q1 a velocidad reducida
628.              joy0 = 0, joy1 = 0,joy2 = 0;    } break;}
629.       ////////////////////////////// HORIZONTAL ////////////////////////////////
630.         while (digitalRead(home_switch0) && digitalRead(home_switch1) &&
   digitalRead(home_switch2)) {
631.            cir0 = 0, cont0 = 0, cir1 = 0, cont1 = 0, cir2 = 0, cont2 = 0;
632.            ValueQ0= analogRead(Joystick0);
633.            if (ValueQ0 > 510) { cont0 = 0, cont1 = 0, cont2=0;}
634.            else if (ValueQ0 < 500) { cont0 = 1, cont1 = 1 , cont2 = 1;}
635.            ValueQ0 = analogRead(Joystick0);  //Lee valor Joystick
636.            MapX = map(ValueQ0, 0, 1023, MinSpeed, MaxSpeed); //Transforma
   lectura del Joystick
637.
638.            MotorControl0(MapX);              //Mover q0 a velocidad normal
639.            Control1(MapX);                   //Mover q1 a velocidad reducida
640.            MotorControl2(MapX);              //Mover q2 a velocidad reducida
641.            break;} estadoAnt1 = estado1;
642.
643.          while (!digitalRead(home_switch0) || !digitalRead(home_switch1) ||
   !digitalRead(home_switch2)) {
644.            ValueQ0 = analogRead(Joystick0);
645.            if (ValueQ0 > 510 && cont0 == 1 && cont1 == 1 &&
   cont2 == 1) { cir0 = 1, cir1 = 1, cir2 = 1;}
646.            else if (ValueQ0 < 500 && cont0 == 0 && cont1 == 0 &&
   cont2 == 0) { cir0 = 2, cir1 = 2, cir2 = 2;}
647.
648.            if ( ValueQ0 > 600 && cir0 == 1 && cir1 == 1 &&
   cir2 == 1) { joy0 = 1, joy1 = 1, joy2 = 1;}
649.            else if ( ValueQ0 < 400 && cir0 == 2 && cir1 == 2 &&
   cir2 == 2 ) { joy0 = 2, joy1 = 2, joy2 = 2;}
650.
651.            if ((cont0 == 1 && cir0 == 1 && joy0 == 1) || cont0 == 0 &&
   cir0 == 2  && joy0 == 2) {
652.              ValueQ0 = analogRead(Joystick0);   //Lee valor Joystick
653.              MapX = map(ValueQ0, 0, 1023, MinSpeed, MaxSpeed);  //Transforma
   lectura del Joystick
654.
655.              MotorControl0(MapX);            //Mover q0 a velocidad normal
656.              Control1(MapX);                 //Mover q1 a velocidad reducida
657.              MotorControl2(MapX);            //Mover q2 a velocidad normal
658.              joy0 = 0, joy1 = 0,joy2 = 0;
659.            } break;}}}
660.       ////////////////////////////////////////////////////////////////////////
661.       // Controlar mediante el Joystick el movimiento del motor //
662.       void MotorControl0(long mapX) {
663.         if (mapX <= UmbralDown) { //Si la lectura del Joystick es menor que el
   umbral
664.           SpeedQ1 = -map(mapX, UmbralDown, MinSpeed, MinSpeed,
   MaxSpeed);   //Mover en direccion 'negativa' con velocidad progresiva
665.           MoveX = true;                   //Activar movimiento
666.         } else if (mapX >= UmbralUp) {    //Si la lectura del Joystick es mayor
   que el umbral
667.           SpeedQ1 = map(mapX, MaxSpeed, UmbralUp, MaxSpeed,
   MinSpeed);    //Mover en direccion 'positiva' con velocidad progresiva
668.           MoveX = true;                   //Activar movimiento
669.         } else {
670.           SpeedQ1 = 0;
671.           MoveX = false;                  //Desactivar movimiento
672.         }
673.
674.         if (MoveX) {                      //Si esta activado el movimiento
675.           Motor0.setSpeed(SpeedQ1);       //Mover a velocidad progresiva
676.           Motor0.run();                   //Activar motor
677.         } else {
678.           Motor0.stop();}                 //Desactivar motor
679.         }
680.
681.       void MotorControl1(long mapY) {
```

```
682.        if (mapY <= UmbralDown) {
683.          SpeedQ2 = -map(mapY, UmbralDown, MinSpeed, MinSpeed, MaxSpeed);
684.          MoveY = true;
685.        } else if (mapY >= UmbralUp) {
686.          SpeedQ2 = map(mapY, MaxSpeed, UmbralUp, MaxSpeed, MinSpeed);
687.          MoveY = true;                    //Activar movimiento
688.        } else {
689.          SpeedQ2 = 0;
690.          MoveY = false;                   //Desactivar movimiento
691.        }
692.
693.        if (MoveY) {                       //Si esta activado el movimiento
694.          Motor1.setSpeed(SpeedQ2);        //Mover a velocidad progresiva
695.          Motor1.run();                    //Activar motor
696.        } else {
697.          Motor1.stop();}                  //Desactivar motor
698.        }
699.
700.      void MotorControl2(long mapZ) {
701.
702.        if (mapZ <= UmbralDown) {
703.          SpeedQ3 = -map(mapZ, UmbralDown, MinSpeed, MinSpeed, MaxSpeed);
704.          MoveZ = true;
705.        } else if (mapZ >= UmbralUp) {
706.          SpeedQ3 = map(mapZ, MaxSpeed, UmbralUp, MaxSpeed, MinSpeed);
707.          MoveZ = true;                    //Activar movimiento
708.        } else {
709.          SpeedQ3 = 0;
710.          MoveZ = false;                   //Desactivar movimiento
711.        }
712.
713.        if (MoveZ) {                       //Si esta activado el movimiento
714.          Motor2.setSpeed(SpeedQ3);        //Mover a velocidad progresiva
715.          Motor2.run();                    //Activar motor
716.        } else {
717.          Motor2.stop();}                  //Desactivar motor
718.        }
719.      ////////////////////////////////////////////////////////////////////////
720.       // MotorControl y Control son codigos similares, solo cambia la
   velocidad //
721.      void Control0(long mapX) {
722.        if (mapX <= UmbralDown) {
723.          SpeedQ1 = -map(mapX, UmbralDown, MinSpeed,   MinSpeed1, MaxSpeed1);
724.          MoveX = true;
725.        } else if (mapX >= UmbralUp) {
726.          SpeedQ1 = map(mapX,  MaxSpeed, UmbralUp,  MaxSpeed1, MinSpeed1);
727.          MoveX = true;
728.        } else {
729.          SpeedQ1 = 0;
730.          MoveX = false;}
731.
732.        if (MoveX) {
733.          Motor0.setSpeed(SpeedQ1);
734.          Motor0.run();
735.        } else {
736.          Motor0.stop();}
737.        }
738.
739.      void Control1(long mapY) {
740.
741.        if (mapY <= UmbralDown) {
742.          SpeedQ2 = -map(mapY, UmbralDown, MinSpeed,   MinSpeed1, MaxSpeed1);
743.          MoveY = true;
744.        } else if (mapY >= UmbralUp) {
745.          SpeedQ2 = map(mapY,  MaxSpeed, UmbralUp,  MaxSpeed1, MinSpeed1);
746.          MoveY = true;
747.        } else {
748.          SpeedQ2 = 0;
749.          MoveY = false;
```

```
750.         }
751.
752.         if (MoveY) {
753.           Motor1.setSpeed(SpeedQ2);
754.           Motor1.run();
755.           // MotorZ.setSpeed(SpeedY);
756.           // MotorZ.run();
757.         } else {
758.           Motor1.stop();
759.           // MotorZ.stop();  }
760.       }
761.
762.       void Control2(long mapZ) {
763.
764.         if (mapZ <= UmbralDown) {
765.           SpeedQ3 = -map(mapZ, UmbralDown, MinSpeed,   MinSpeed1, MaxSpeed1);
766.           MoveZ = true;
767.         } else if (mapZ >= UmbralUp) {
768.           SpeedQ3 = map(mapZ,  MaxSpeed, UmbralUp,  MaxSpeed1, MinSpeed1);
769.           MoveZ = true;
770.         } else {
771.           SpeedQ3 = 0;
772.           MoveZ = false;}
773.
774.         if (MoveZ) {
775.           Motor2.setSpeed(SpeedQ3);
776.           Motor2.run();
777.         } else {
778.           Motor2.stop();}
779.         }
```

ANEXO B: USO DE LA CORTADORA LASER

En este anexo se presenta un manual con la serie de pasos a seguir para el uso de la cortadora laser del departamento en el laboratorio a partir de un diseño en 2D en formato .cdr en versión 12 de corelLaser.

-Encender el ordenador si no lo estuviera ya, usando la contraseña de usuario: impresora3d.

- Revisar la zona de corte de la máquina en busca de restos o sedimentos que puedan desequilibrar la plancha del material que vayamos a usar.

- Encender extractor.

- Configurar la potencia del laser a 90% y velocidad de 15 mm/s para madera y 90% pero aumentando la velocidad de corte al 25 mm/s para metacritalto.

- Abrir programa corelLaser del escritorio y abrir el archivo .cdr en version v12 con CorelDraw

- Configurar dimensiones del tablón 600x400mm en la esquina superior izquierda, y distribuir las diferentes piezas del archivo dentro de él.

- Pinchar en 'cutting' dentro del programa, esquina derecha.

a. Confgurar velocidad, dependiendo del material a cortar.
b. Realizar un preview de forma rectangular para ver si nuestro diseño cabe en el tablero.
c. Configurar el número de repeticiones a 2 para madera y un número superior para metacrilato.
d. Hacer click en starting para comenzar.
e. Cuando termine la primera repetición, hacer click en inciar siguiente repetición.

- Recoger las piezas cortadas y tirar los desechos a la basura.

Nota: si nos pidieran introducir KEY, sacar y volver a meter el pendrive de la figura y reiniciar el programa.

Fig. Anexo B-1: Cortadora Láser

ANEXO C: DISEÑO SOBRE PLANO

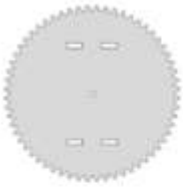
Lista de piezas:

P1 x1

P2 x1

P3 x1

P4 x2



P5 x1

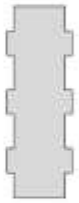
P6 x1

P7 x1

P8 x3

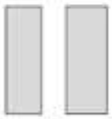
P9 x1

P10 x1



P11 x1

P12 x1



P13 x4

P14 x3

P15 x1

P16 x3

P16 x1

P17 x2

P18 x1

P19

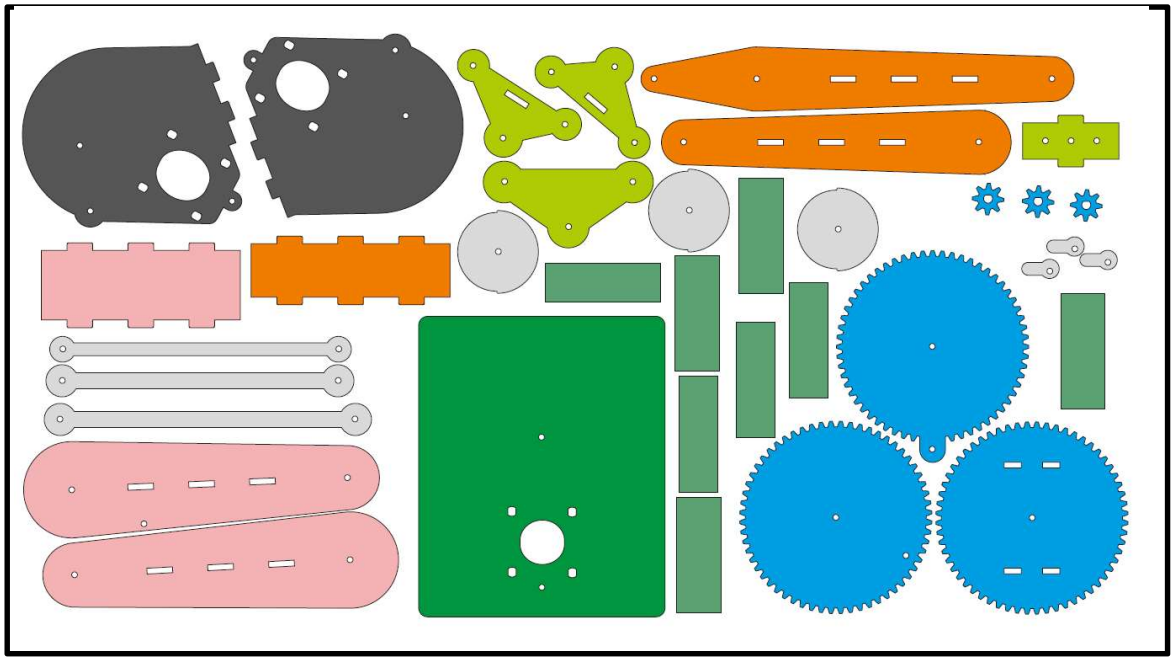


Fig. Anexo C-1: Piezas en plano

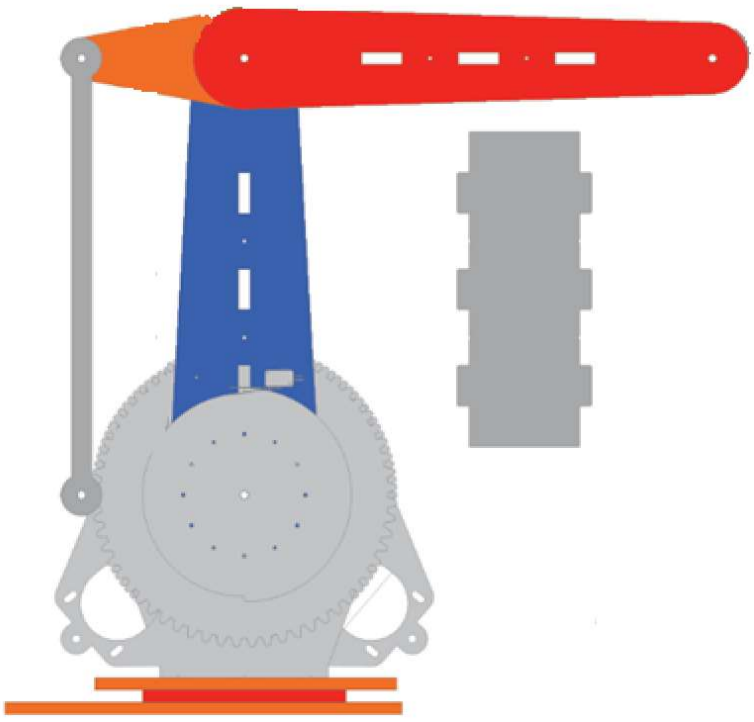


Fig. Anexo C-2: Perfil estructura

ANEXO D: MONTAJE

Se detallarán las fases para el montaje final del brazo robótico.

Base: esta parte cuenta con unos soportes en cada esquina.

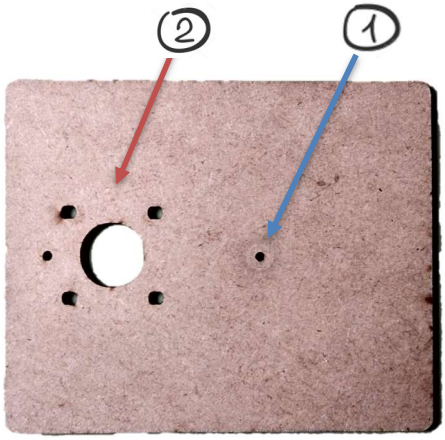
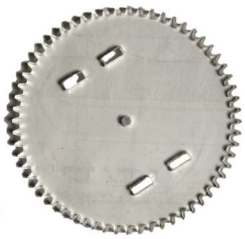


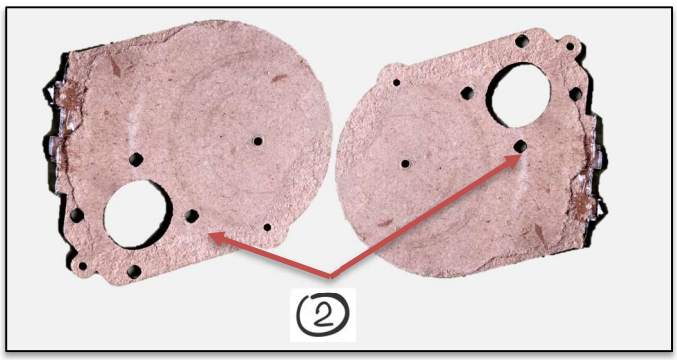
Fig. Anexo D-1: Base (a), tornillos (b), tuercas frenada (c), arandelas (d)

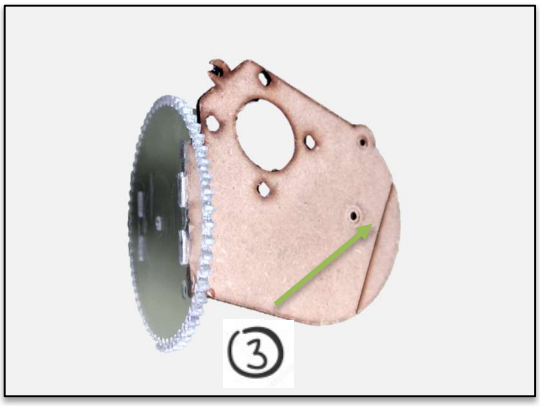


1 → Es el agujero correspondiente al engranaje de la base. Pondremos una arandela por ambas caras de la base, cerrando el tornillo con una tuerca frenada.

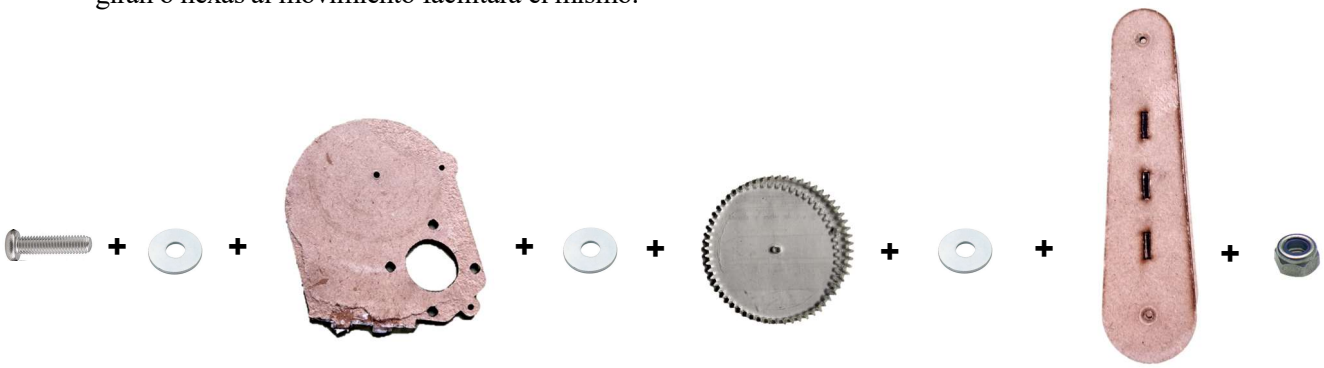
2 → Estos agujeros sirven para anclar el motor de la base, para ello se usará 4 tornillos con 4 arandelas porque el diámetro de nuestros agujeros es superior al de los tornillos. Se diseñó así para poder ajustar mejor el engranaje al piñon cuando se ponga el motor.

Sujeción de los motores: este conjunto de piezas se ha diseñado a medida para los Nema 17 que usaremos y serán dos piezas que irán fijadas en el engranaje de la base como veremos en la siguiente figura y se colocorán enfrentadas para equilibrar el peso de cada motor en la estructura. Se necesitará al menos dos tornillos con arandela para cada motor y, además, la combinación tornillo-arandela-pieza-arandela-tuerca para fijar el brazo.

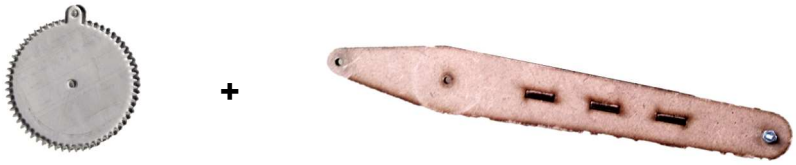




③ → Atornillaremos en cada sujeción los elementos referidos al brazo junto con sus engranajes. Entre cada una de las piezas que se ensamble deberemos introducir una arandela ya que siendo piezas que giran o nexas al movimiento facilitará el mismo.

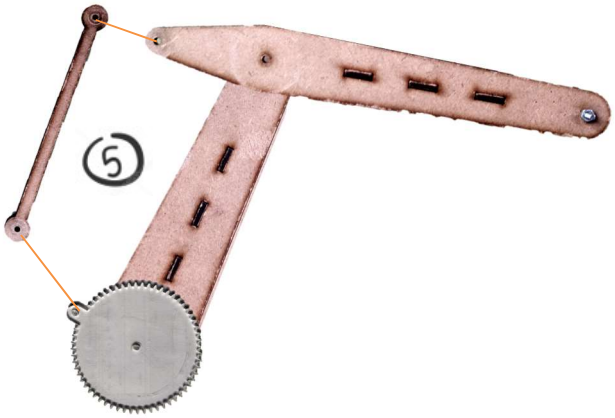


Una cosa importante es aclarar que en un mismo lado deben coincidir el engranaje que tiene un saliente con un agujero para fijarlo con la parte del antebrazo que también tiene un saliente correspondiente



④ → Conectar el brazo al antebrazo. Para ello usaremos cuatro arandelas, dos tornillos y dos tuercas frenadas.

⑤ → Aquí usamos el elemento que conecta el engranaje encargado del movimiento del antebrazo. Al tener los motores a la misma altura y con ella la tracción alejada del elemento que queremos mover es necesario el uso de esta pieza.



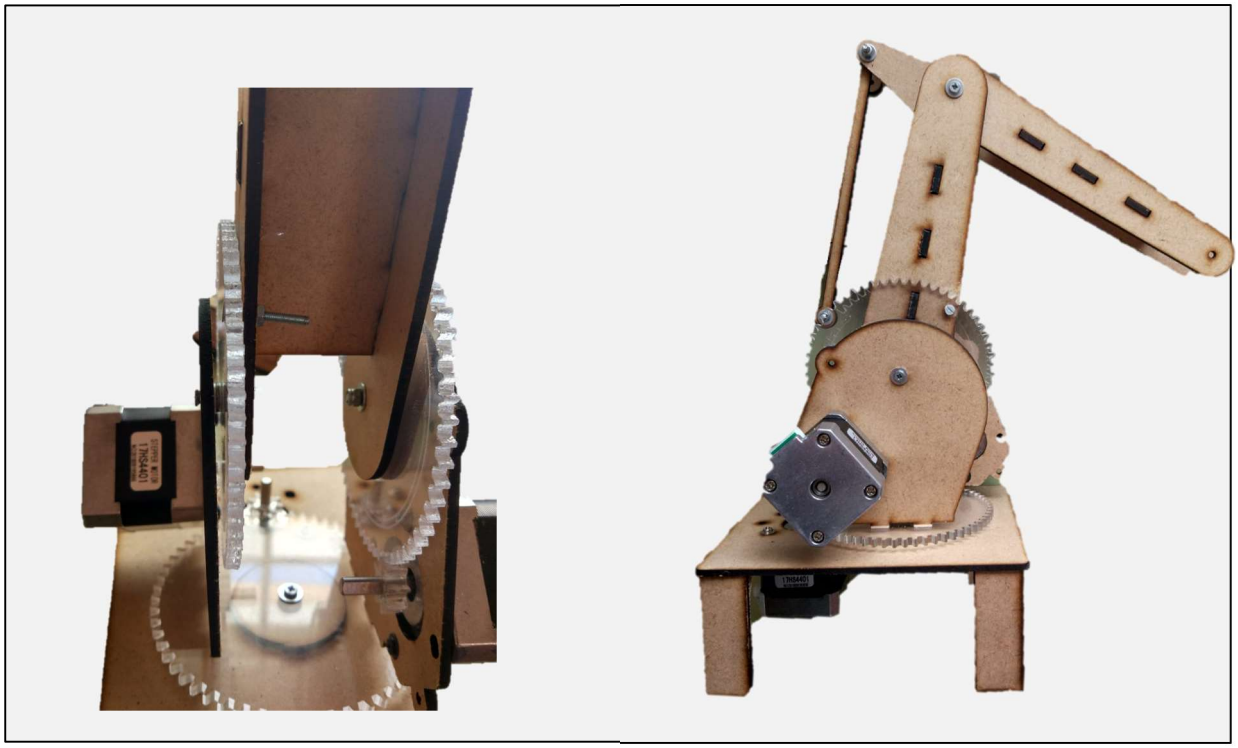


Fig. Anexo D-2: Montaje final

Para el efector final se diseña un soporte para el bolígrafo que usaremos. Diseñado en Catia V5

Unas últimas recomendaciones serían pegar las piezas que tienen que fijarse unas a otras con uso de algún pegamento, cola blanca. Esto aportará mayor estabilidad, además, el uso de un eje común que atornille las dos sujeciones de motor.

ANEXO E: PRESUPUESTO

A continuación, se evalúa el coste del desarrollo de este proyecto:

Elementos Electrónicos

	Cantidad (unidades)	*Precio unitario (€)*	*Total*
Motor PAP (Nema17)	3	15	45
Arduino Mega	1	13,99	13,99
Interruptor de fin carrera	3	0,70	2,10
Joystick	2	1,62	3,24
Driver A3967LSB	3	3.75	11,25

Estructura y engranajes

	Cantidad (m^2)	*Precio unitario (€)*	*Total*
Madera MDF	0,5	5.19	2.59
Metacrilato	0,5	11,98	5,99

Otros

Tornillos, tuercas, arandelas, conectores, protoboard, piezas 3D de PLA	10 €

TOTAL	*94.16 €*