UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño, implementación y control de un prototipo de robot SCARA de 4 grados de libertad

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

AUTOR/A: Lorente Romanos, Roberto

Tutor/a: Casanova Calvo, Vicente Fermín

CURSO ACADÉMICO: 2022/2023

ÍNDICE

Índice de Figuras

Índice de Tablas

1 Introducción

1.1 Motivación

En la actualidad, la robótica se ha convertido en una disciplina de gran relevancia en numerosos campos de aplicación. Los robots desempeñan un papel fundamental en la automatización de procesos industriales, la exploración espacial, la asistencia médica, la investigación científica y muchas otras áreas. Entre los diversos tipos de robots existentes, los brazos robóticos SCARA han adquirido un notable peso debido a su versatilidad y eficiencia en una amplia gama de tareas.

Los robots SCARA (Selective Compliance Assembly Robot Arm), que se pueden observar en la *figura 1,* se caracterizan por tener una estructura articulada que les permite movimientos en forma de arco y un alto grado de precisión en un plano horizontal. Su diseño se basa en una combinación de juntas rotativas y prismáticas, lo que les confiere la capacidad de realizar movimientos rápidos y repetitivos con una excelente precisión. Estas características hacen que los robots SCARA sean ampliamente utilizados en aplicaciones industriales que requieren manipulación, ensamblaje y posicionamiento precisos.

En este contexto, el presente trabajo de fin de máster se enfoca en el diseño e implementación de un prototipo de brazo robótico SCARA de 4 grados de libertad.

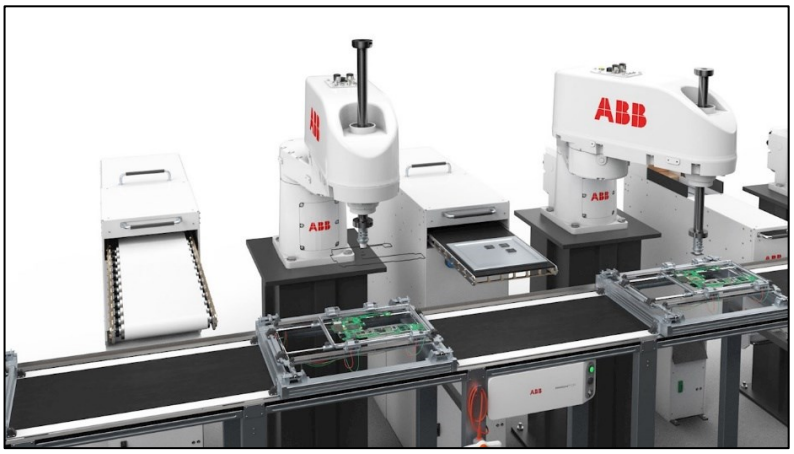


Figura 1 Robot SCARA Industrial

El diseño y la implementación de este prototipo implica no solo la integración de elementos mecatrónicos y la fabricación en 3D, sino también la simulación en ordenador y el desarrollo de programas de control que permitan su funcionamiento eficiente. El correcto diseño y la buena implementación de estos programas son fundamentales para garantizar el funcionamiento del brazo robótico y su integración en entornos de trabajo reales.

Habida cuenta de lo anteriormente expuesto, este proyecto representa una oportunidad excepcional para aplicar los conocimientos adquiridos a lo largo del máster de Mecatrónica, llevándolos a la práctica en un proyecto tangible y de relevancia en el campo de la robótica.

1.2 Objetivos

El objetivo principal del presente trabajo consiste en implementar un prototipo de robot de tipo SCARA de 4 grados de libertad que, a través de una interfaz gráfica en el ordenador, facilite al usuario su control en modo manual y automático.

El modo manual permite operar todas las articulaciones de forma independiente (modo JOG) y también situar el efector final en una posición XYZ definida (Cinemática Inversa). En el modo automático, el usuario debe situar una pieza sobre una pantalla táctil y, el robot, tras recibir las coordenadas, calcula la trayectoria y recoge adecuadamente la pieza.

Para lograr este objetivo general, se han determinado los siguientes objetivos parciales:

- Analizar los modelos matemáticos que definen la cinemática del robot SCARA y que permiten posicionar el prototipo de forma precisa y controlada.
- Analizar los componentes mecánicos y estructurales de los que se compone el robot, así como los mecanismos de transmisión utilizados para generar cada movimiento.
- Realizar un estudio del funcionamiento y el control de los motores paso a paso de cara a dimensionarlos adecuadamente y controlarlos de forma segura y eficaz.
- Modelar el sistema en *SIMSCAPE* de forma que, utilizando las herramientas de *MATLAB* y *SIMULINK*, el modelo permita realizar las simulaciones necesarias previas al funcionamiento real del prototipo.
- Fabricar las piezas por impresión 3D y realizar el ensamblaje mecánico y eléctrico del prototipo.
- Desarrollar los dos programas de control, estos son, el programa embebido en el microcontrolador del robot y el programa encargado de la visualización del panel de mando en el PC.

1.3 Resumen del Documento

El primer capítulo de este trabajo es esta introducción donde se ha descrito con detalle la motivación que ha llevado a considerar importante llevar a cabo un proyecto mecatrónico real con todas sus etapas. Además, se han planteado los objetivos detallados de este trabajo y se incluye este resumen.

El segundo capítulo es el encargado de fijar los conceptos matemáticos necesarios para controlar en todo momento la posición del robot, tanto de sus articulaciones como del efector final. Con este objetivo, se definirá la nomenclatura utilizada y los métodos de cálculo cinemático, presentando las distintas opciones que existen y justificando el método seleccionado.

En el tercer capítulo, se realizará una presentación de todos los componentes que conforman el prototipo de robot SCARA. Este sistema se compone de manera específica de cinco subensamblajes o partes principales: la base, la torre de elevación, el brazo 1, el brazo 2 y la pinza. Además de detallar cada una de estas partes, se destacará su función principal, el mecanismo de transmisión que integran, así como los motores, finales de carrera y rodamientos que albergan. De esta manera, se brindará una visión completa y detallada de la estructura y los elementos clave que conforman el robot SCARA.

En el cuarto capitulo se analizará el funcionamiento y naturaleza de los motores paso a paso híbridos que se han utilizado para generar el movimiento en las articulaciones del robot. Además, se presentará detalladamente el proceso de cálculo necesario para dimensionar los motores y, tras analizar cuidadosamente los resultados, se seleccionará el tamaño de motor adecuado para el prototipo.

En el quinto capítulo se desarrollará la simulación realizada del prototipo de robot SCARA utilizando la herramienta Simscape en Matlab. En él, además de ensamblar el modelo de simulación y resaltar las utilidades más representativas de esta herramienta, se abordarán aspectos importantes como la interacción entre objetos (fuerzas de contacto) y el control de las articulaciones. El objetivo de todo esto será conseguir una simulación del prototipo de robot SCARA de cara a rediseñar y desarrollar el sistema, permitiendo evaluar configuraciones, analizar el comportamiento y prever posibles problemas previa implementación física.

En el sexto apartado se llevará a cabo un detallado análisis de los aspectos relacionados con la fabricación de las piezas y ensamblaje mecánico y eléctrico del prototipo. Todo este proceso estará respaldado por un estudio exhaustivo de los componentes electrónicos y la pantalla táctil resistiva. Se realizará un análisis detallado del esquema eléctrico y de todas las conexiones, y se presentarán diversos programas de prueba que permiten verificar de manera individual el correcto funcionamiento de los diferentes sistemas y componentes. De esta forma, se proporcionará una visión completa y rigurosa del proceso de fabricación, ensamblaje y validación del prototipo, asegurando su funcionamiento óptimo y confiable.

En el séptimo apartado se presentarán los aspectos del proyecto relacionados con la programación del Robot y del Panel de Mando. Se detallará el funcionamiento de todos los modos de marcha, el modo de utilización de los botones de mando y seguridad y la filosofía que se ha utilizado para realizar las comunicaciones y el envío de información. Además, se presentarán las características de ciertas partes del código que por su importancia, dificultad u originalidad merecen ser remarcadas.

Los últimos capítulos incluyen, por orden, los resultados reales, las propuestas de mejora, las conclusiones del proyecto, el presupuesto, el pliego de condiciones, la relación con los objetivos ODS y los anexos referentes al código, los planos y los datasheet.

2 Modelado Matemático

En este apartado se van a fijar los conceptos matemáticos necesarios para controlar en todo momento la posición del robot, tanto de sus articulaciones como del efector final. Con este objetivo, se definirá la nomenclatura utilizada y los métodos de cálculo cinemático, presentando las distintas opciones que existen y justificando el método seleccionado.

2.1 Nomenclatura del Robot

A continuación, se presentan las variables geométricas del robot y el sistema de coordenadas utilizado. Estos conceptos son clave para entender de forma adecuada el desarrollo matemático y se van a utilizar a lo largo de todos los apartados posteriores del documento. Cabe destacar que se ha tomado la rotación antihoraria como sentido de giro positivo de las articulaciones.

En la *figura 2* se representan de forma gráfica dichas variables siendo L_1 la longitud del brazo 1, L_2 la longitud del brazo 2, h_z la altura desde el suelo hasta el efector final, θ_1 el ángulo que forma el brazo 1 con el eje x, θ_2 el ángulo que forma el brazo 2 con la prolongación del brazo 1, θ_3 el ángulo que forma la pinza con la prolongación del brazo 2.

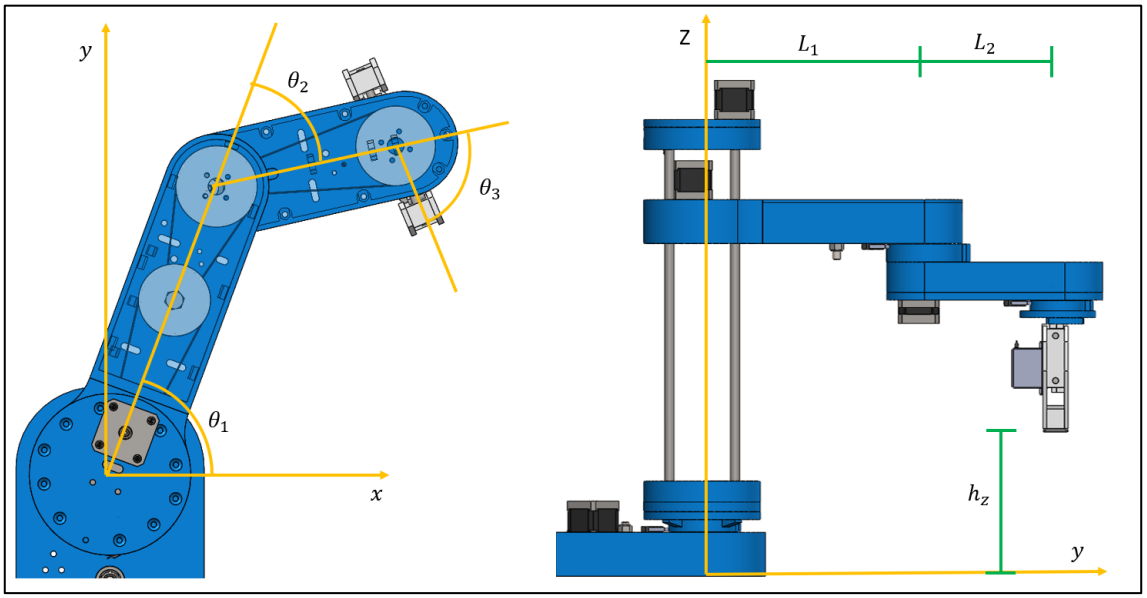


Figura 2 Nomenclatura y Sistema de Coordenadas

2.2 Estudio Cinemático

La cinemática de un robot se encarga del estudio de los movimientos y posiciones de sus diferentes elementos o articulaciones. Se centra en el análisis matemático de las relaciones geométricas y cinemáticas que rigen el comportamiento del robot en el espacio. En otras palabras, la cinemática

permite entender cómo se mueve un robot y cómo se relacionan las variables que determinan su posición y orientación.

Este estudio es esencial para diseñar el sistema de control y planificar trayectorias, ya que proporciona las bases teóricas para entender cómo los distintos grados de libertad del robot se relacionan entre sí y cómo evoluciona su posición y orientación en el tiempo.

Existen dos tipos principales de cinemática en robótica: la cinemática directa y la cinemática inversa.

La cinemática directa se refiere al cálculo de la posición y orientación del extremo final del robot (efector final) en función de las variables de los grados de libertad del robot. De esta forma, dado un conjunto de ángulos o posiciones articulares, la cinemática directa permite determinar la posición y orientación del efector final en el espacio de trabajo del robot.

La cinemática inversa, por otro lado, se refiere al proceso de determinar los valores de las variables articulares del robot necesarios para lograr una posición y orientación deseada del efector final. Es decir, dado un punto o una posición objetivo en el espacio de trabajo, la cinemática inversa permite calcular los ángulos o posiciones de las articulaciones del robot necesarios para alcanzar esa posición. Ambos conceptos se pueden apreciar en la *figura 3*.

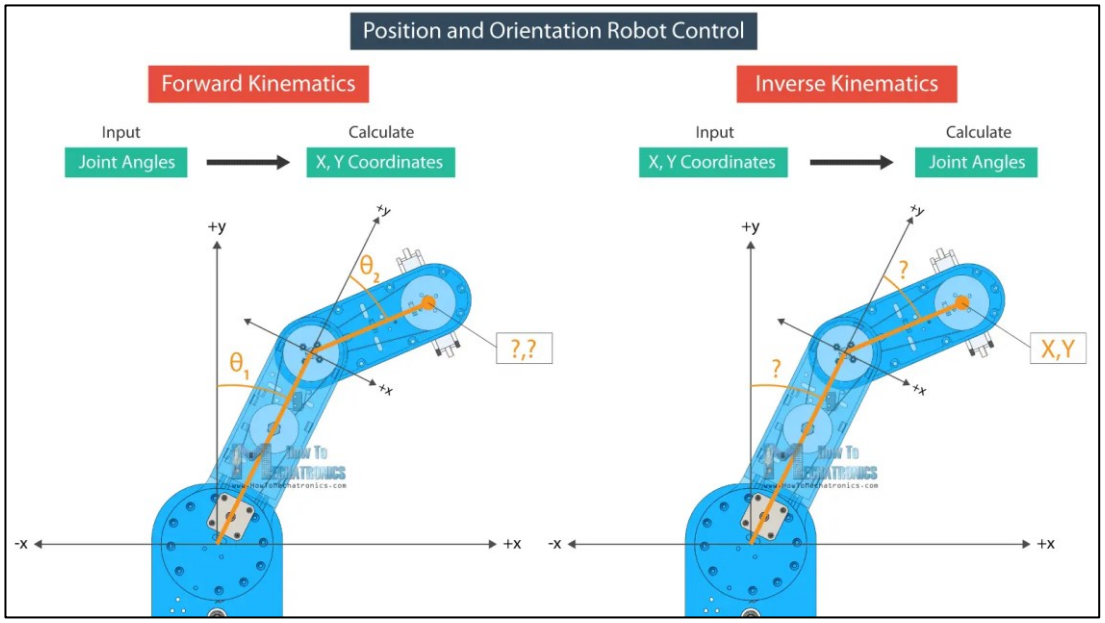


Figura 3 Cinemática Directa e Inversa del Robot SCARA

Para el cálculo cinemático existen varios métodos que permiten determinar la posición y orientación del efector final en función de las variables articulares o viceversa. Aunque el prototipo de brazo robótico en cuestión posee 4 grados de libertad (GDL), la elección del método para calcular su cinemática se ha basado en la simplicidad y facilidad de implementación.

Tradicionalmente, el método de Denavit-Hartenberg es ampliamente utilizado para el cálculo de la cinemática en robots de múltiples grados de libertad, tanto directa como inversa. Este método se basa en asignar sistemas de referencia a cada articulación del robot y utilizar parámetros geométricos y de desplazamiento para describir las transformaciones entre estos sistemas de referencia. Sin embargo,

dado que el objetivo principal es desarrollar un prototipo funcional y práctico, se ha optado por emplear un método más sencillo: el método geométrico.

El método geométrico consiste en utilizar relaciones geométricas y trigonométricas para determinar la posición del efector final. Este método permite calcular los ángulos adecuados de las dos primeras articulaciones, es decir, θ_1 y θ_2.

Para el control de la altura en el eje Z y el giro de la pinza (este último determinará la orientación del efector final), se ha considerado un enfoque independiente, lo que permite controlar estas variables de forma separada y ajustarlas según sea necesario.

2.3 Cinemática Directa

La cinemática directa permite conocer cuál es la posición y orientación del extremo del robot a partir de sus coordenadas articulares y las ecuaciones que permiten calcularla por el método geométrico se presentan a continuación.

$$x = l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) \quad (1)$$

$$y = l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \quad (2)$$

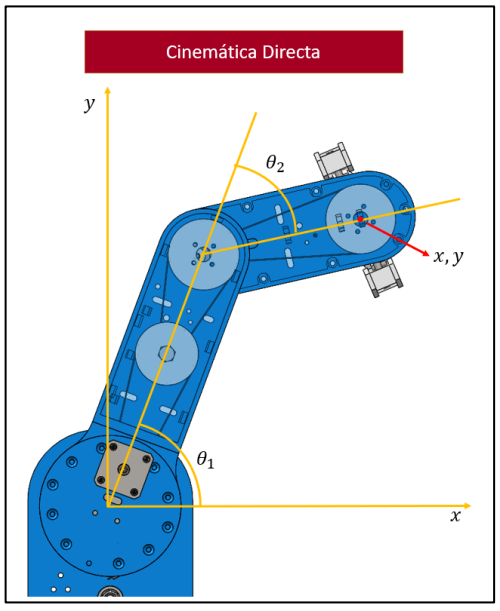


Figura 4 Cinemática directa del Robot SCARA

2.4 Cinemática Inversa

El procedimiento para calcular la cinemática inversa del robot SCARA mediante el método geométrico implica definir el sistema de coordenadas, obtener las ecuaciones de posición mediante igualdades y conceptos trigonométricos y resolverlas para obtener los valores de las variables articulares correspondientes a una posición deseada del efector final. Este enfoque permite determinar cómo mover las articulaciones del robot para lograr una configuración específica del efector final en el espacio de trabajo.

Basadas en el esquema de la *figura 5*, las ecuaciones que permiten calcular la cinemática inversa por el método geométrico se presentan a continuación. En ellas, se ha utilizado la variable auxiliar "D" para simplificar las ecuaciones. Además, tanto en la ec.(4) como en la ec.(5) se utiliza la función "$atan2(a,b)$", que permite operar en todos los cuadrantes del plano cartesiano devolviendo el ángulo correcto incluso si las coordenadas son negativas.

$$D = \frac{x^2 + y^2 - {L_2}^2 - {L_1}^2}{2L_1L_2} \quad (3)$$

$$q_2 = \text{atan2}(\pm\sqrt{1 - D^2}, D) \quad (4)$$

$$q_1 = \text{atan2}(y, x) - \text{atan2}(L_2 \sin(q_2), L_1 + L_2 \cos(q_2)) \quad (5)$$

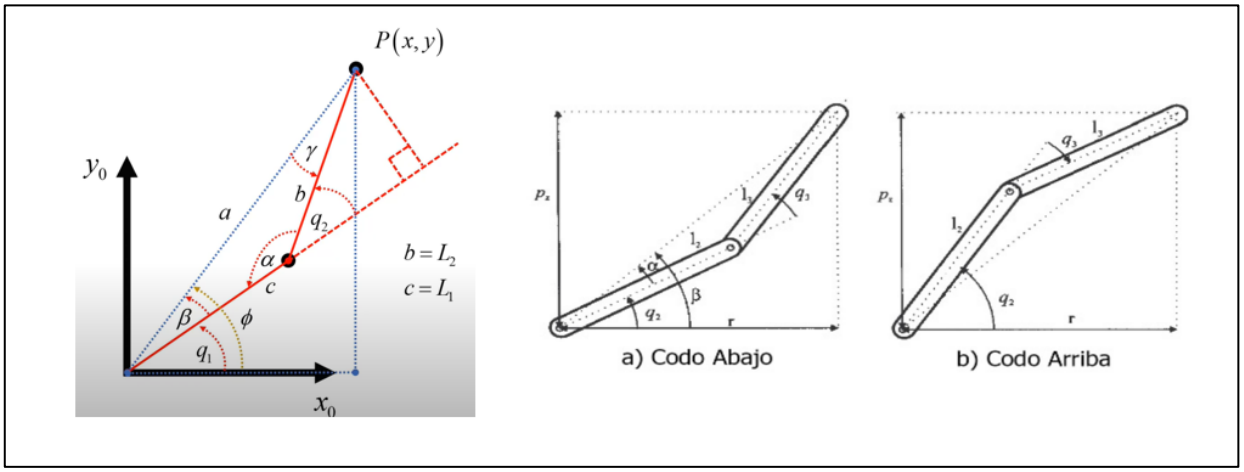


Figura 5 Cinemática inversa del Robot SCARA y configuraciones de codo arriba y abajo

Por último, se debe tener en cuenta que para cada punto x, y del plano, el robot puede adoptar 2 configuraciones distintas denominadas codo arriba y codo abajo. En las ecuaciones anteriores, este aspecto viene definido en el operador " $\pm$ " que va delante de la raíz de la ec.(4). En la *figura 5* queda ilustrado este aspecto para un punto del primer cuadrante.

3 Diseño del Robot y Componentes

En este apartado se van a presentar detalladamente todas las partes de las que se compone este prototipo de robot SCARA. Aunque la mayoría de estos robots pueden parecer similares constructivamente, hay ciertos aspectos que varían de unos a otros, como pueden ser el orden del tipo de articulación (Prismáticas o de Revolución) o los sistemas mecánicos encargados de generar cada movimiento.

El prototipo está compuesto por 5 subensamblajes o partes principales: la base, la torre de elevación, el brazo 1, el brazo 2 y la pinza. Todos ellos se pueden distinguir en la *figura 6.*

En los próximos apartados, se van a presentar detalladamente todas estas partes, resaltando su función principal, el mecanismo de transmisión que incorporan, así como los motores, finales de carrera y rodamientos que alojan.

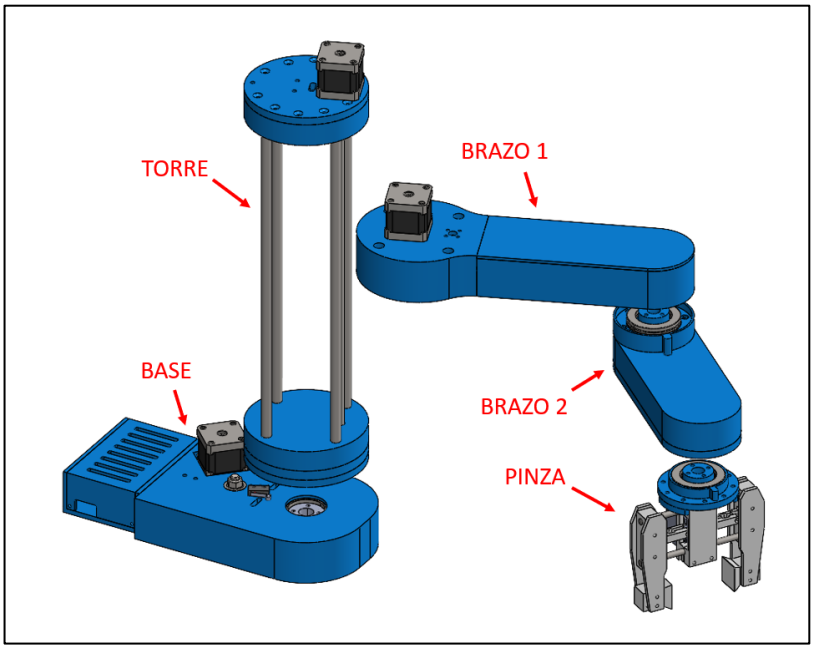


Figura 6 Subensamblajes principales del prototipo

3.1 Base

La base es la parte del robot que está en contacto con el suelo y actúa como bancada. Se trata de una pieza sólida y resistente puesto que tiene que soportar todo el peso del robot.

Esta pieza sirve como soporte para la primera transmisión mecánica, es decir, la encargada de que todo el robot gire. Acorde con la nomenclatura del *Apartado 2.1*, este mecanismo permite generar el movimiento θ_1. Esta es una transmisión formada por correas y poleas dentadas y ofrecen una reducción 20:1 a través de dos etapas de reducción. Esta accionada por el *Motor 1 (M1)* e incorpora el *final de carrera 1 (FC1)*.

Además, tal y como se puede apreciar en la parte izquierda de la *figura 7*, este subensamblaje incorpora una caja donde va alojada toda la electrónica.

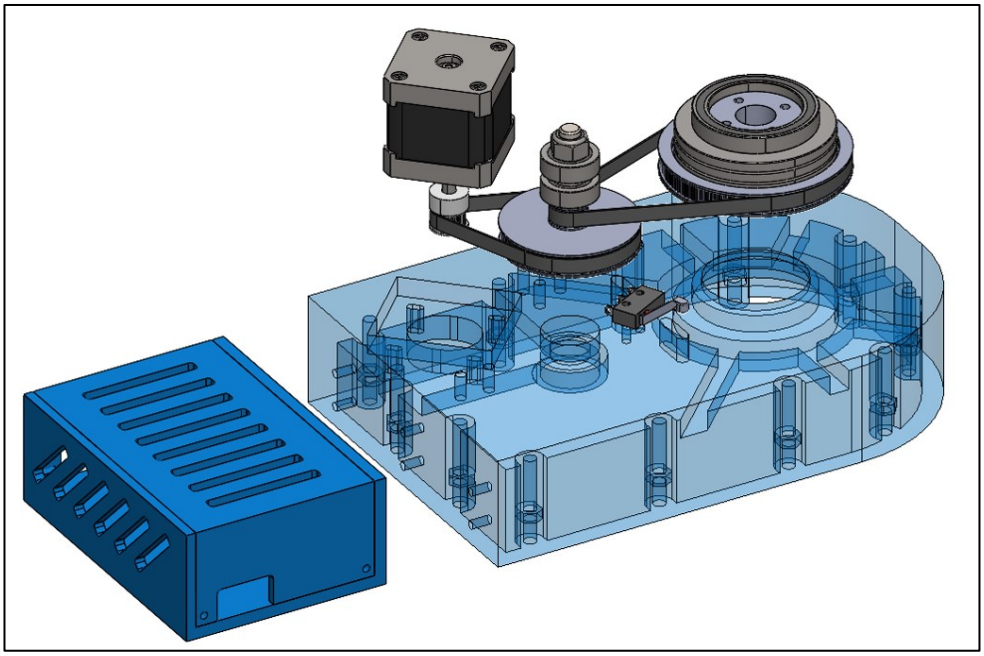


Figura 7 Subensamblaje Base del robot

La base del robot va conectada con la torre de elevación formando la primera junta de revolución (J1). Esta articulación se compone de dos rodamientos axiales y uno radial, que, trabajando conjuntamente, permiten soportar todas las cargas generadas en el robot, tanto en el funcionamiento dinámico como en el estático y, además, ofrecen movimientos suaves y precisos. En la *figura 8* se puede observar la forma constructiva de esta junta.

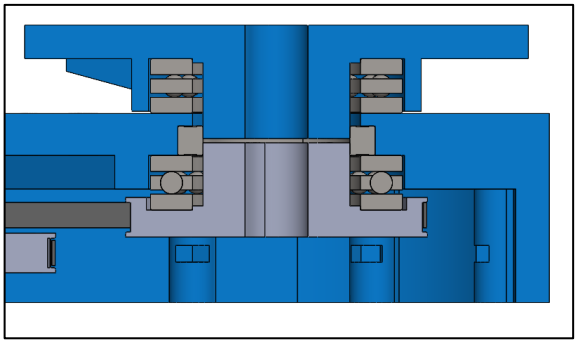


Figura 8 Rodamientos de la Articulación J1

3.2 Torre de elevación

La torre de elevación actúa de nexo entre la base y el brazo 1 y es la estructura encargada de permitir y generar el movimiento del sistema en el eje Z.

Este movimiento en el eje Z (h_z según la nomenclatura del A*partado 2.1*) se consigue gracias a una transmisión tuerca-husillo. En este caso, el motor *MZ* va directamente acoplado a la carga, es decir, el índice de reducción es de 1. Además, tal y como se puede apreciar en la *figura 9*, el sistema está formado por 4 barras de acero que tienen función estructural y cinemática, esto es, conectan y rigidizan la torre de elevación y sirven como guías para los 4 rodamientos lineales que incorpora el *brazo 1*.

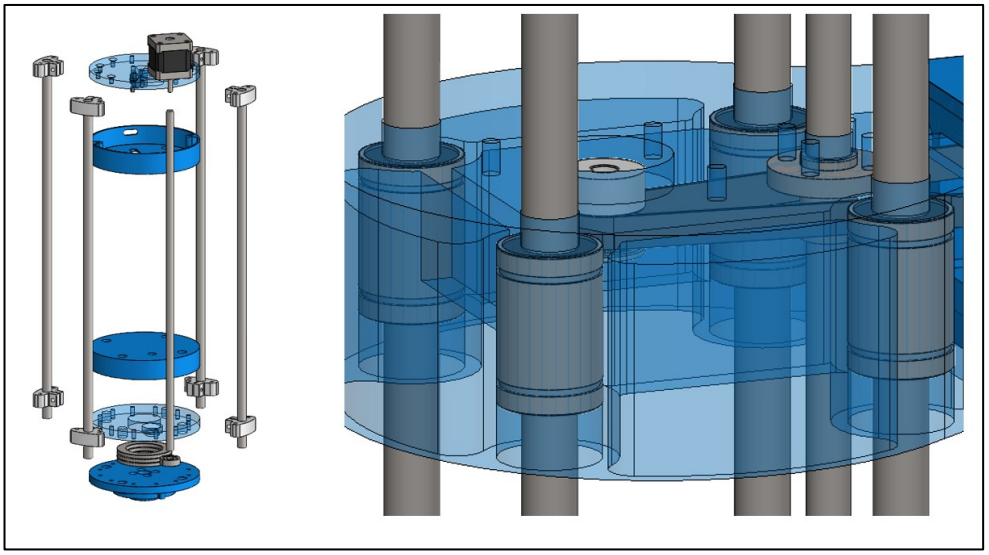


Figura 9 Torre de elevación y conexión con brazo 1

3.3 Brazo 1

El brazo 1, cuya longitud es de *228 mm*, es la parte que conecta la torre con el brazo 2. Este elemento es el encargado de alojar el sistema de transmisión que moviliza al brazo 2, es decir, se encarga de generar movimiento θ_2. Esta transmisión está compuesta por 2 etapas de correas y poleas dentadas y permite obtener un ratio de 16:1.

En la *figura 10*, además de apreciar el sistema de transmisión comentado, se puede observar el *motor M2* y *el final de carrera FC2*.

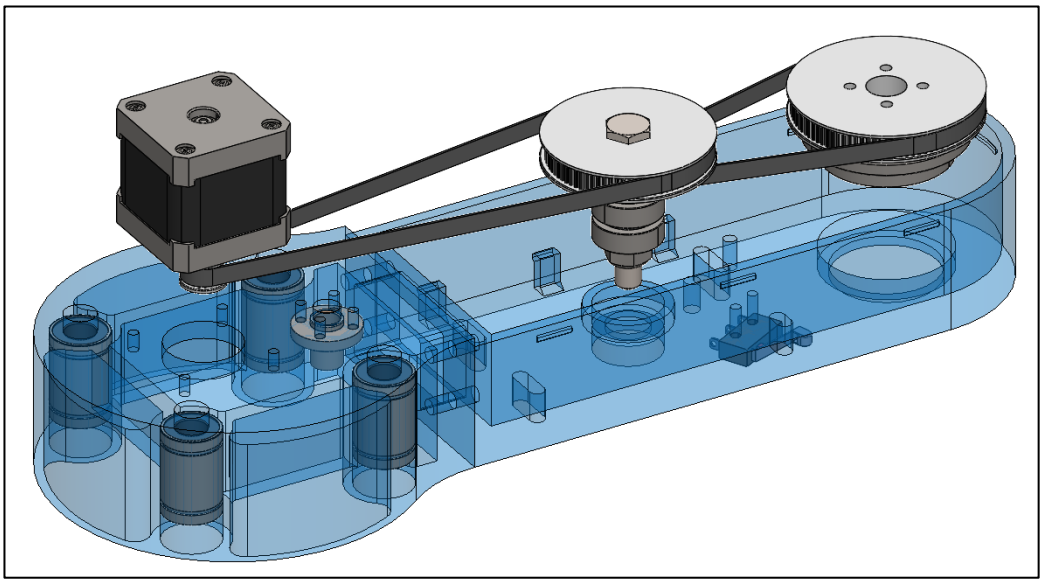


Figura 10 Subensamblaje del Brazo 1

Para la conexión entre los dos brazos se ha optado por utilizar dos rodamientos axiales y uno radial que, trabajando conjuntamente, permiten soportar todas las cargas generadas en el robot, tanto en el funcionamiento dinámico como en el estático. Además, ofrecen movimientos suaves y precisos. En la *figura 11* se puede observar la forma constructiva de esta junta.

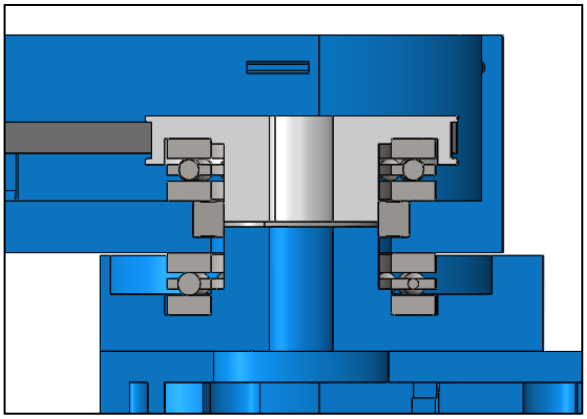


Figura 11 Rodamientos de la Articulación J2

3.4 Brazo 2

El brazo 2, cuya longitud es de *136.5 mm*, es la parte que conecta el brazo 1 con la pinza. Sus componentes pueden apreciarse en la *figura 12* y este elemento es el encargado de alojar el sistema de transmisión que permite el giro θ_3 de la pinza.

El conjunto está accionado por el *motor M3* y nuevamente es un mecanismo formado por poleas y correas dentadas que ofrece un ratio de 4:1 a través de una única etapa.

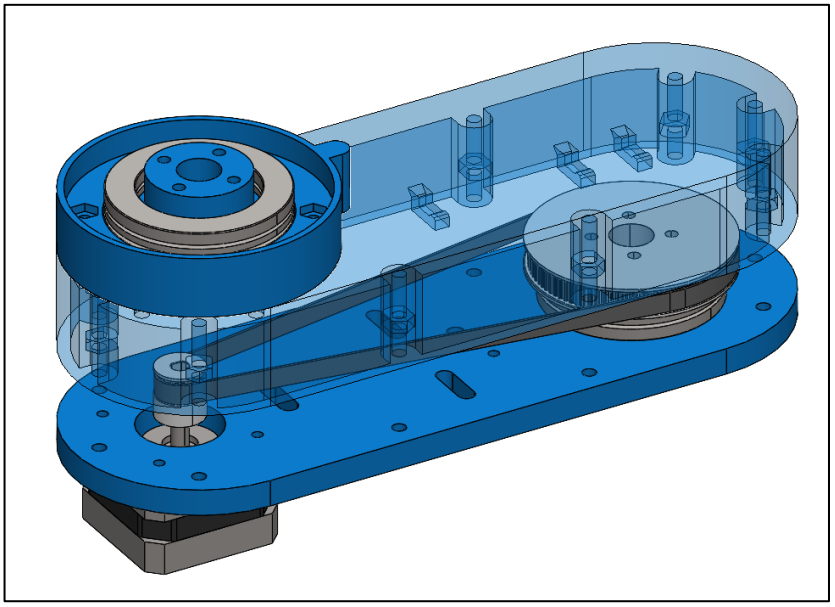


Figura 12 Subensamblaje del Brazo 2

Para la conexión con la pinza se ha optado por utilizar dos rodamientos axiales y uno radial que, trabajando conjuntamente, permiten soportar todas las cargas generadas en el robot, tanto en el funcionamiento dinámico como en el estático. Además, ofrecen movimientos suaves y precisos. En la *figura 13* se puede observar la forma constructiva de esta junta.

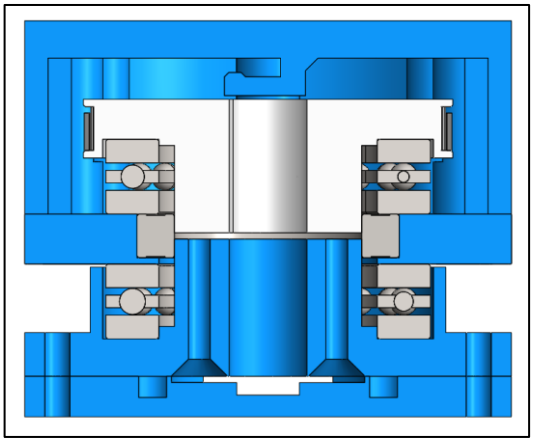


Figura 13 Rodamientos de la Articulación J3

3.5 Pinza

La pinza es el elemento del robot que le permite interactuar con su entorno de forma que el prototipo sea capaz de coger y transportar distintos tipos de objetos.

Tal y como se observa en la *figura 14,* el mecanismo principal es del tipo barra-corredera y está accionado por un servomotor. Este sistema permite transformar el movimiento rotatorio del Servomotor en un movimiento longitudinal de las pinzas.

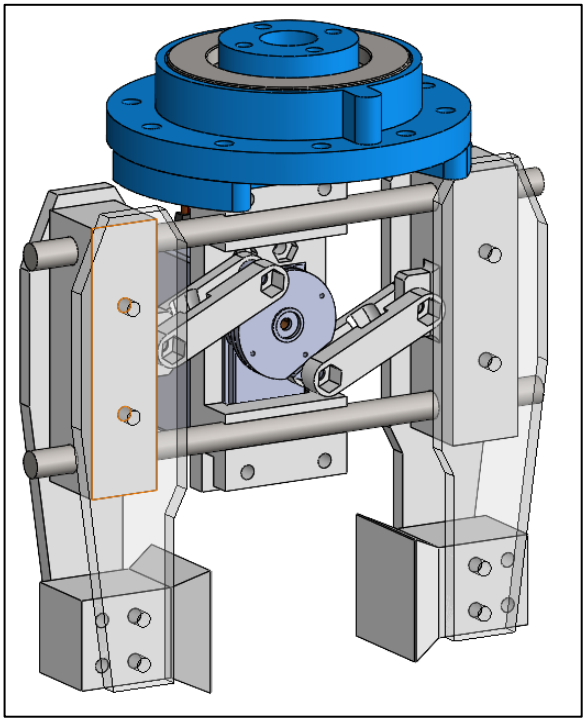


Figura 14 Subensamblaje de la Pinza

4 Estudio y Dimensionamiento de los Motores paso a paso

En este apartado se va a analizar el funcionamiento y naturaleza de los motores paso a paso híbridos que se han utilizado para generar el movimiento en las articulaciones del robot. Además, se presentará detalladamente el proceso de cálculo necesario para dimensionar los motores y, tras analizar cuidadosamente los resultados, se seleccionará el tamaño de motor adecuado para el prototipo.

4.1 Funcionamiento de los Motores paso a paso Híbridos

Los motores paso a paso híbridos son ampliamente utilizados en aplicaciones de robótica, incluyendo el control de las articulaciones de un robot. Estos motores se caracterizan por su precisión, facilidad de control y buena relación entre torque y tamaño. Su funcionamiento se basa en la generación de pasos discretos o incrementos angulares, lo que los hace especialmente adecuados para el control preciso de movimientos en un rango definido.

En términos generales, un motor paso a paso híbrido está compuesto por un rotor y un estator. El rotor está magnetizado y cuenta con imanes permanentes, mientras que el estator tiene devanados o bobinas electromagnéticas. El estator se energiza de forma secuencial para crear campos magnéticos que interactúan con los imanes del rotor, generando así un movimiento paso a paso.

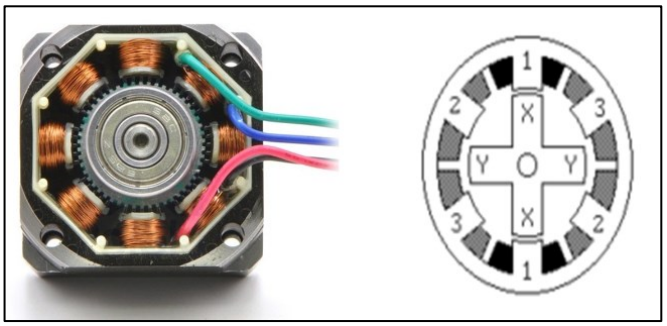


Figura 15 Motor paso a paso híbrido

El control del motor paso a paso híbrido se logra mediante la secuencia adecuada de corriente aplicada a las bobinas del estator. Esta secuencia determina la dirección y el número de pasos que el motor dará. El control se puede realizar a través de un controlador de motor paso a paso que envía las señales de activación a las bobinas según sea necesario. Sobre los controladores se hablará en profundidad en el *Apartado 6.2.4*.

4.2 Proceso de Cálculo de los Motores

El dimensionamiento de un motor se refiere al proceso de determinar las especificaciones adecuadas del motor necesario para cumplir con los requisitos de una aplicación específica. Esto implica considerar factores como el torque requerido, la velocidad de operación, el ciclo de trabajo, el tamaño y la eficiencia del motor. Para dimensionar un motor correctamente, es necesario realizar un análisis detallado de las cargas y demandas de la aplicación, teniendo en cuenta los factores de seguridad y los márgenes de funcionamiento. El objetivo final es seleccionar un motor que pueda proporcionar el rendimiento necesario sin sobredimensionar ni subdimensionar, asegurando un funcionamiento óptimo y confiable del sistema en el que se integrará el motor.

Aunque la forma de dimensionar un motor siempre responde a estos conceptos, dependiendo de si el sistema que se desea implementar posee unas especificaciones u otras, este cálculo puede abordarse de distintas maneras.

Por ejemplo, en situaciones donde se quiere diseñar un sistema que siempre vaya a desempeñar la misma secuencia de movimientos y en el que además se requieran unas condiciones de velocidad, aceleración y precisión de posicionamiento muy bien definidos, el motor puede ser dimensionado de forma muy precisa para satisfacer únicamente esos requisitos.

Por el contrario, si se tienen otro tipo de limitaciones o hay ciertos aspectos del proyecto que pueden adaptarse al desempeño que los motores puedan ofrecer, como es el caso de este proyecto, el dimensionamiento de los motores puede enfocarse de otra forma.

Este prototipo incorpora dos sistemas de transmisión diferentes en sus articulaciones, por husillo y por polea-correa, y, en los siguientes apartados se detallará el proceso de cálculo para cada tipo.

4.2.1 Consideraciones Previas

Frecuencia de pulsos

El primer factor limitante en este tipo de sistemas es la cantidad de pulsos que se pueden enviar al motor por segundo, es decir, la frecuencia de pulsos. Este valor puede estar limitado tanto por el microcontrolador como por el driver del motor.

Para este proyecto, se ha utilizado como microcontrolador una tarjeta Arduino MEGA, que dispone de una frecuencia de reloj de 16 MHz. Atendiendo a las especificaciones encontradas en el documento [7], la frecuencia máxima de pulsos que se puede enviar al motor a través de este microcontrolador es de $4000\ pul/s$. De este dato, se derivan los valores de velocidad y aceleración de la *tabla 1.*

Velocidad [pul/s]	4000
Aceleración [pul/s^2]	4000

Tabla 1. Velocidades y aceleraciones máximas de los motores

De esta forma, la velocidad y aceleración máxima de cada articulación vendrá definida por este valor, combinado con el valor de *microstepping* y la reducción del sistema de transmisión que incorpore.

Microstepping

El microstepping es una técnica utilizada en motores paso a paso para lograr movimientos más suaves y precisos. Permite generar incrementos de posición más pequeños que los pasos discretos tradicionales, lo que proporciona una mayor resolución y control en el posicionamiento del motor. El microstepping ofrece beneficios como una menor vibración, un movimiento más suave y una mayor precisión, aunque puede afectar el torque disponible del motor.

En general, cuanto menor sea el incremento del paso utilizado en el microstepping, menor será el torque máximo que el motor podrá proporcionar. Por lo tanto, es necesario encontrar un equilibrio entre la suavidad del movimiento y la capacidad de torque requerida para la aplicación específica.

Como se comentará en el *apartado 4.3*, los motores seleccionados están sobredimensionados. Por lo tanto, el par máximo sigue muy por encima del par requerido. Con este criterio y en base a valores de precisión que también se comentarán, el microstepping con el que se utilizarán los motores será de ¼ de paso.

Este valor es utilizado para calcular la variable θ_{step}, que será utilizada en las ecuaciones siguientes.

$$\theta_{step\,[pul/rev]} = \frac{360[°/rev]}{g_p[°/pul] * \mu_s} = \frac{360}{1.8 * \frac{1}{4}} = 800 \quad (6)$$

Resolución de posicionamiento

La resolución de posicionamiento es un valor característico de cada articulación y depende del avance del sistema de transmisión, la ratio de reducción y el valor de microstepping. Representa el movimiento en unidades de distancia, tales como [mm] o [°] que avanza la articulación por cada pulso que recibe el motor. La ecuación (7) permite calcularla.

$$L_{\vartheta\,[uds_{dist}/pul]} = \frac{Avance\,[uds_{dist}/rev]}{i * \theta_{step\,[pul/rev]}} \quad (7)$$

4.2.2 Cálculo del Par Motor en una Articulación con Husillo

Las siguientes ecuaciones permiten determinar el par motor necesario para mover la carga en las condiciones especificadas en una transmisión por husillo, teniendo en cuenta factores como la inercia, la fuerza externa, la fricción y la inclinación.

La ecuación (8) calcula el par motor total necesario, que es la suma del par de aceleración y el par de funcionamiento.

$$T_{mot\,[Nm]} = T_{acc} + T_{run} \quad (8)$$

El par de aceleración se calcula utilizando la inercia total y el cambio de velocidad multiplicado por un cambio de unidades para convertirlo a radianes por segundo.

$$T_{acc\,[Nm]} = J_{tot\,[kg\,m^2]} + \frac{\Delta_{vel\,[rpm]}}{\Delta_{t\,[s]}} * \frac{2\pi}{60} \quad (9)$$

La inercia total se obtiene sumando la inercia del motor con la suma de la inercia del husillo y de la carga, divididas por el cuadrado de la relación de transmisión.

$$J_{tot\,[kg\,m^2]} = J_{mot} + \frac{J_{hu} + J_w}{i^2} \quad (10)$$

Para la inercia de la carga se utilizan datos como el peso, la gravedad, la eficiencia y el paso del husillo.

$$J_{w\,[Kgm^2]} = \frac{W\,[kg]}{e * g\,[m/s^2]} * \left(\frac{1}{Paso\,[^{rev}/_{m}] * 2\pi}\right)^2 \quad (11)$$

La inercia del husillo se determina utilizando la longitud del husillo, la densidad de su material, el radio del husillo y la aceleración gravitacional.

$$J_{hu\ [Kgm^2]} = \frac{(\pi * L_{hu\,[m]} * \rho_{a\,\left[\frac{Kg}{m^3}\right]} * r^4)}{2 * g\,[m/s^2]} \quad (12)$$

El par de funcionamiento se calcula dividiendo la fuerza total entre el paso del husillo y la relación de transmisión.

$$T_{run\,[Nm]} = \frac{F_{tot\,[N]}}{Paso\,[^{rev}/_{m}] * 2\pi * i} \quad (13)$$

La fuerza total se obtiene sumando la fuerza externa, la fuerza de rozamiento y la fuerza gravitacional.

$$F_{tot\,[kg]} = F_{ext} + F_{roz} + F_{grav} \quad (14)$$

La fuerza de rozamiento se calcula multiplicando el coeficiente de fricción entre husillo-tuerca por el peso y el coseno del ángulo de inclinación.

$$F_{roz\,[N]} = \mu * W[kg] \cos\theta * 10 \quad (15)$$

La fuerza gravitacional se obtiene multiplicando el peso por el seno del ángulo de inclinación.

$$F_{grav[Kg]} = W[kg] \sin\theta * 10 \quad (16)$$

Los valores del sistema utilizados para calcular las anteriores ecuaciones se reúnen en la *tabla 2.* Simplemente comentar que se ha mayorado la carga a 10 Kg en aras de prever posibles aplicaciones del robot que incorporen mayor peso. Además, los valores del rozamiento y de la eficiencia se han obtenido del catálogo [3].

Peso de la carga [kg]	10
Avance del husillo [mm/rev]	2
Paso del husillo [rev/m]	500
Angulo de inclinación del husillo [rad]	pi/2
Coef. rozamiento husillo-tuerca	0.15
Eficiencia del husillo	0.5
Longitud del husillo [m]	0.38
Radio del husillo [m]	0.004
Indice de reducción	1
Inercia del motor [kgm^2]	5.70E-06
Densidad del acero [kg/m^3]	7900

Tabla 2. Datos de la transmisión por husillo

4.2.3 Cálculo del Par Motor en una Articulación con Polea-Correa

Las siguientes ecuaciones permiten calcular el par motor necesario en una transmisión por correa y poleas, considerando tanto la inercia del sistema como la eficiencia de la transmisión.

La ecuación (17) proporciona el par motor total necesario, que se obtiene sumando el par de aceleración y el par de funcionamiento, y luego se divide por la eficiencia de la transmisión de poleas.

$$T_{mot\,[Nm]} = (T_{acc} + T_{run})\,\frac{1}{\eta_p} \qquad (17)$$

El par de funcionamiento se establece en cero. Esto implica que no se requiere un par adicional para mantener el movimiento de la carga una vez que ha alcanzado la velocidad deseada.

$$T_{run\,[Nm]} = 0 \qquad (18)$$

El par de aceleración se calcula utilizando la inercia total y el cambio de velocidad. Al igual que en el caso anterior, se multiplica el cambio de velocidad por (2π/60) para convertirlo a radianes por segundo.

$$T_{acc\,[Nm]} = J_{tot\,[kg\,m^2]} + \frac{\Delta_{vel\,[rpm]}}{\Delta_{t\,[s]}} * \frac{2\pi}{60} \qquad (19)$$

La inercia total se obtiene sumando la inercia del motor con la inercia del robot dividida por el cuadrado de la relación de transmisión. Esto tiene en cuenta la inercia tanto del motor como del sistema mecánico que se está impulsando.

$$J_{tot\,[kg\,m^2]} = J_{mot} + \frac{J_{robot}}{i^2} \qquad (20)$$

Los valores del sistema utilizados para calcular las anteriores ecuaciones se reúnen en la tabla 3. Comentar que, dado que las poleas dentadas han sido impresas en 3D, se ha creído conveniente tomar un valor de 0.5 para el rendimiento de cada etapa. Además, para calcular la inercia del robot se ha utilizado el programa SolidWorks.

Avance por revolución [º/rev]	360
Rendimiento 1 etapa polea-correa	0.5
Número de etapas	2
Rendimiento total	0.25
Reducción total	20
Inercia del motor [kgm^2]	5.70E-06
Inercia del robot [kg*m^2]	0.0793

Tabla 3. datos de la transmisión por polea-correa

4.3 Análisis de los Resultados y Selección de Motores

El prototipo está formado por cuatro articulaciones. Tres de ellas son de polea-correa y la otra es de husillo. Con el objetivo de reducir los costes del proyecto, los cuatro motores seleccionados deberán ser iguales, lo que implica que el motor elegido deberá satisfacer los requisitos de la articulación que sea más exigente.

Las dos articulaciones que requieren mayor esfuerzo son la articulación 1 (J1), formada por una transmisión de polea-correa y la articulación 2 (J2), constituida por un sistema de husillo.

Tras realizar todos los cálculos de las ecuaciones anteriores utilizando los valores expuestos, los resultados para las dos articulaciones más exigentes se muestran en la *tabla 4.*

Articulación	Transmisión	Vel. Max [rpm]	Par Motor [Nm]
1	Polea-Correa	300	0.025
2	Husillo	300	0.032

Tabla 4. Par motor de las articulaciones más exigentes

De esta forma, se puede concluir que la articulación más exigente, y que por tanto va a determinar el motor necesario, es la articulación 2, formada por un sistema de transmisión de husillo. Estos cálculos suponen un par motor calculado de $T_{mot} = 0.32\ Nm$.

Este valor debe tomarse con sumo cuidado, puesto que las ecuaciones solamente son un primer punto de partida.

Se debe tener muy en cuenta que todas las partes estructurales del prototipo han sido impresas en 3D con una impresora de gama baja. Además, los componentes comerciales involucrados en las transmisiones también son de baja calidad. Esto implica que el funcionamiento real del prototipo estará expuesto a multitud de factores adversos que estas ecuaciones no han tenido en cuenta y que pueden producir grandes desalineamientos, vibraciones o mala lubricación entre muchos otros.

Además, este tipo de motores, si no están convenientemente sobredimensionados, pueden perder pasos si se les exigen pares instantáneos que no son capaces de proporcionar. Esta pérdida de pasos supondría un funcionamiento final inadecuado y, dado que los motores no disponen de encoder, no habría forma de detectar este mal funcionamiento.

Por todo esto, y en base a valores recomendados en el *catálogo [3]*, se ha decidido multiplicar este valor de par por un coeficiente de seguridad de $C_s = 3$, por lo que el par motor requerido será de $T_{mot} = 0.1\ Nm$.

Llegados a este punto y en base a todo lo anteriormente citado, se ha decidido seleccionar un motor NEMA 17 cuya curva de par-velocidad se representa en la *figura 16*.

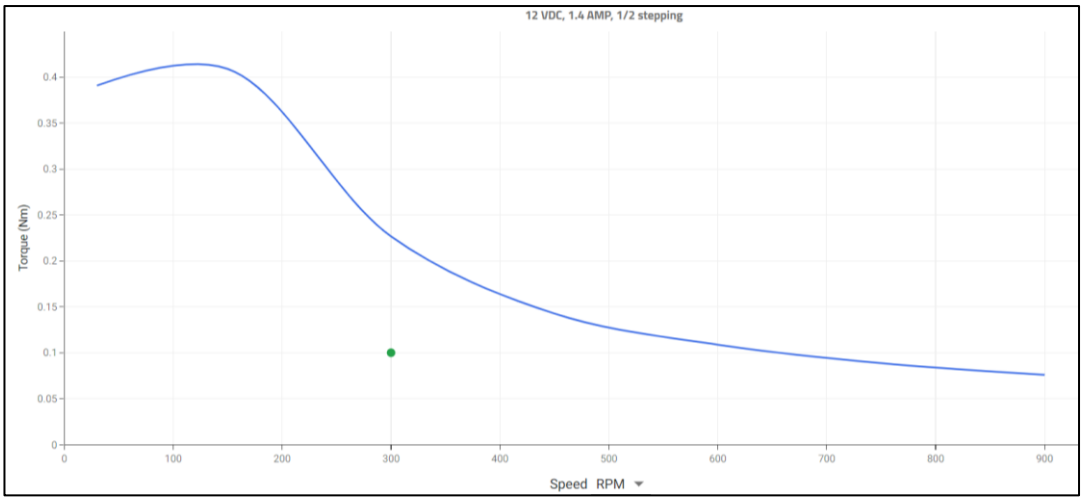


Figura 16. Curva Par-Velocidad del motor seleccionado

Este motor, a la velocidad de 300 rpm y alimentado a 12V y 1.4 A, es capaz de ofrecer un par de $T_{mot} = 0.22\ Nm$. Esto implica que el punto de operación que se ha calculado supone el 50% del rendimiento máximo que puede dar el motor en estas condiciones. Atendiendo a las especificaciones de dimensionamiento del *catálogo [3]*, esto supone un dimensionamiento adecuado del motor para esta aplicación.

Por último, se debe comentar que los drivers con los que se van a comandar los motores permiten regular la intensidad máxima que le puede llegar a cada uno. Dado que el motor seleccionado sí que está sobredimensionado para las demás articulaciones se ha optado por regular adecuadamente el límite de intensidad de cada driver, permitiendo así un funcionamiento más adecuado y eficiente de cada motor. Los valores de par motor y de corriente máxima de las demás articulaciones se encuentran

recopiladas en la *tabla 5.* Además, se ha aprovechado para añadir las resoluciones de posicionamiento de cada una.

Art.	Transmisión	Vel. Max [rpm]	Resolución [uds_dist/pul]	Par Motor [Nm]	Par mayorado [Nm]	Límite de Intensidad [A]
1	Polea-Correa	300	0.0225	0.025	0.075	1
2	Husillo	300	0.0025	0.032	0.096	1.4
3	Polea-Correa	300	0.0281	0.011	0.033	0.5
4	Polea-Correa	300	0.1125	0.015	0.045	0.65

Tabla 5. Valores finales de par, resolución e intensidad de las 4 articulaciones

5 Simulación del Prototipo

En este capítulo se presenta la simulación realizada del prototipo de robot SCARA utilizando la herramienta Simscape en Matlab. En él, además de ensamblar el modelo de simulación y resaltar las utilidades más representativas de esta herramienta, se han abordado aspectos importantes como la interacción entre objetos (fuerzas de contacto) y el control de las articulaciones. El objetivo de todo esto es conseguir una simulación del prototipo de robot SCARA de cara a rediseñar y desarrollar el sistema, permitiendo evaluar configuraciones, analizar el comportamiento y prever posibles problemas antes de la implementación física.

5.1 Herramienta Simscape Multibody

En el ámbito de la ingeniería, las herramientas de simulación desempeñan un papel fundamental en el diseño y desarrollo de sistemas complejos. Una de estas herramientas ampliamente utilizada es Simscape, una biblioteca de modelado físico disponible en el entorno de programación Matlab.

La principal fortaleza de Simscape radica en su enfoque basado en componentes físicos. En lugar de enfocarse únicamente en ecuaciones matemáticas, esta herramienta se centra en los elementos físicos que componen el sistema, como elementos estructurales, los grados de libertad de cada uno, motores, engranajes, entre otros. Estos componentes físicos se modelan mediante relaciones matemáticas que representan las leyes fundamentales de la física, lo que permite una simulación más realista y precisa.

La simulación con Simscape ofrece una serie de ventajas significativas para el diseño e implementación de prototipos. En primer lugar, permite la evaluación de diferentes configuraciones y parámetros del sistema antes de llevar a cabo la construcción física del prototipo. Esto implica un ahorro considerable de recursos, como tiempo y dinero, al evitar iteraciones costosas y errores en las etapas iniciales del proceso de desarrollo.

Además, Simscape brinda la posibilidad de realizar análisis detallados y exhaustivos del comportamiento del sistema en diferentes condiciones y escenarios. Esto permite identificar posibles limitaciones, optimizar el rendimiento y prever posibles fallas o problemas que podrían surgir durante la implementación real.

5.2 Diseño del Modelo de Simulación

El primer paso consiste en ensamblar en Simscape un modelo funcional del prototipo. Para ello, se ha optado por seguir la idea de que el modelo debe ser lo más simplificado posible, pero al mismo tiempo, debe satisfacer todos los requisitos que se le exijan al proyecto. Algunos de estos requisitos pueden ser: respetar las cotas principales del prototipo, representar todas las articulaciones, tener en cuenta la inercia real de los elementos del sistema, etc.

5.2.1 Bloques Principales y Ensamblaje del Modelo

El enfoque utilizado para el diseño del modelo implica la combinación de bloques de la librería de Simscape Multibody con bloques genéricos de las librerías de Simulink, definiendo sus propiedades y relaciones. A continuación, se presentarán de manera concisa los bloques clave de Simscape Multibody utilizados, considerando que podrían resultar menos familiares para el lector.

Bloque Solid:

Este bloque permite la definición de figuras tridimensionales junto con sus propiedades de inercia, su apariencia durante la simulación y su posición y orientación en el espacio de simulación en relación con otros sólidos. En el contexto de este trabajo, las diferentes piezas que conforman el brazo robótico ya han sido diseñadas en SolidWorks. Por lo tanto, se procede a cargar el archivo correspondiente en formato .SLDPRT y se definen las características mencionadas.

A modo de ejemplo, en la *figura 17* se presenta la configuración de la base del Robot. En ella se puede apreciar que se ha introducido una densidad de $1.24\ kg/m^3$, que corresponde a la densidad del PLA, material del que están fabricadas las piezas.

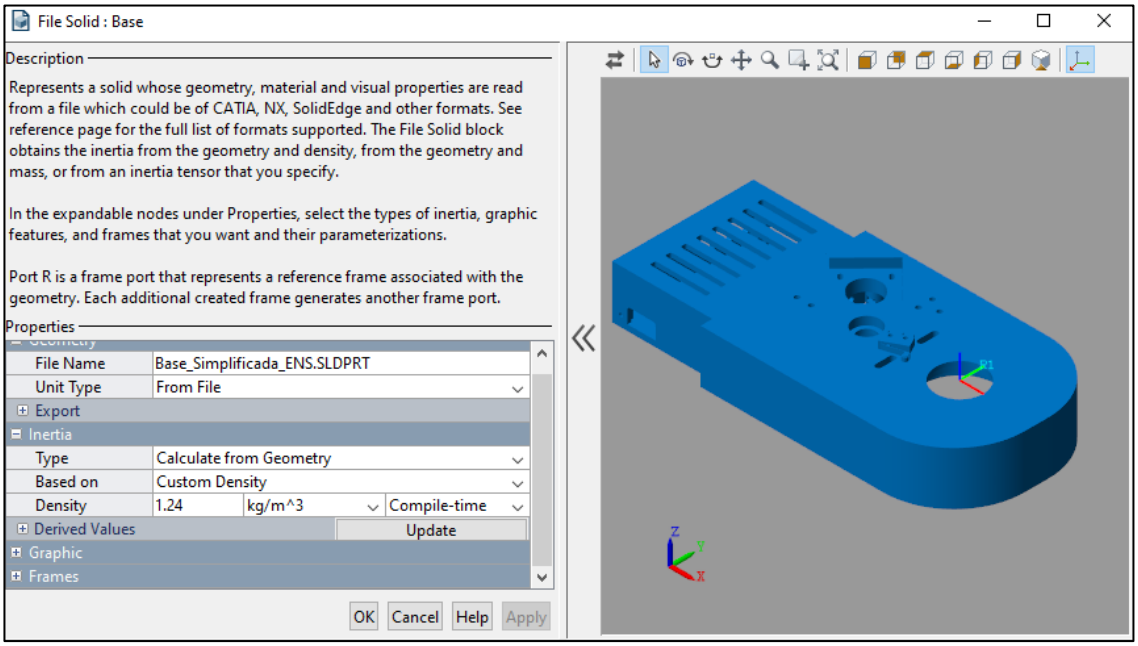


Figura 17. Configuración de la base del Robot en Simscape

Para conectar dos piezas consecutivas del robot, se deben situar dos sistemas de coordenadas con la misma posición y orientación que pertenezcan cada uno a una de las piezas. Para facilitar este propósito, todas las piezas han sido previamente tratadas en SolidWorks para que al importarlas en Simscape tengan el sistema de coordenadas principal en la posición deseada. No obstante, hay ciertos componentes que no ha sido posible introducirlos de esta forma y por tanto se ha tenido que hacer uso de la siguiente herramienta.

Bloque Rigid Transform:

La funcionalidad del bloque "Rigid Transform" es esencial para definir las relaciones de traslación y orientación entre los sólidos de manera precisa y eficiente. A menudo, resulta más conveniente y efectivo que el bloque "Solid". Mediante este bloque, es posible establecer numéricamente la distancia y la orientación entre dos sólidos o entre un sólido y el origen del espacio de simulación, lo que proporciona una mayor flexibilidad en la configuración de los componentes del sistema.

De esta forma, gracias a este bloque ha sido posible situar tanto los elementos sólidos como la posición concreta de las articulaciones. En la *figura 18* se aprecian las opciones de configuración de este tipo de bloques.

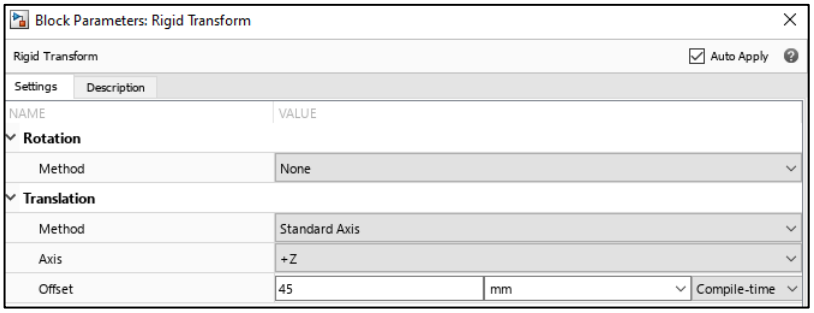


Figura 18. Ejemplo de uso de los bloques Rigid Transform

Bloque Joint:

Las "Joints" son bloques genéricos que representan las diferentes articulaciones que pueden unir los bloques "Solid". En el contexto de este estudio, se han utilizado tanto juntas de revolución como de traslación, ya que son las necesarias para representar el funcionamiento del Brazo Robot en cuestión.

Estos bloques proporcionan una amplia variedad de opciones para el diseño, permitiendo definir propiedades tales como los parámetros mecánicos de la articulación (rigidez y rozamiento), el método de actuación (siendo estos el Par o la Posición), además de ofrecer distintos sensores para recopilar datos sobre las magnitudes más relevantes (posición, velocidad, aceleración, par…). Todos estos aspectos y muchos otros se pueden observar en la *figura 19.*

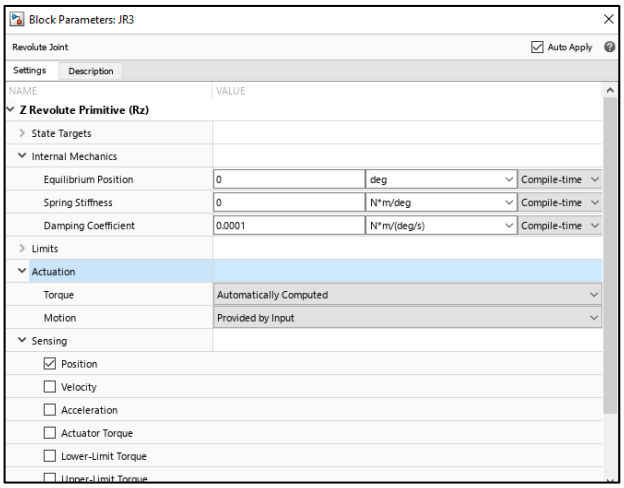


Figura 19. Ejemplo de configuración del bloque Revolute Joint

Bloques de Simulación:

Este conjunto de bloques está formado por tres partes, el "*Solver Configuration*", que permite establecer los parámetros de la simulación, el "*World Frame*", que representa el origen de todo el ensamblaje y el "*Mechanism Configurator*", que permite introducir fuerzas externas tales como la gravedad.

Todas las simulaciones de Simscape Multibody deben incorporar estos 3 bloques, que se presentan en la *figura 20.*

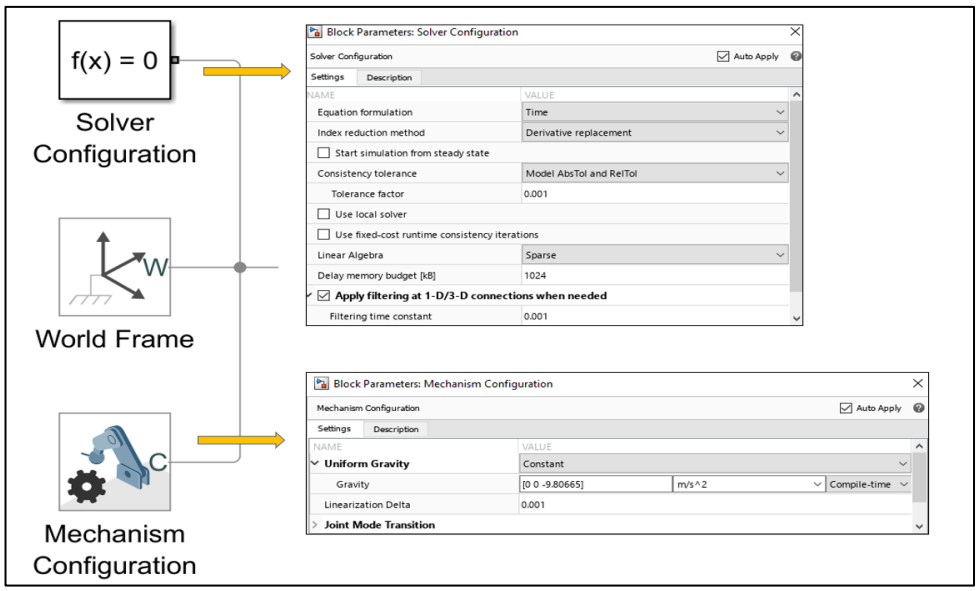


Figura 20. Bloques de configuración de Simscape Multibody

Haciendo uso de todas las herramientas anteriormente citadas, se puede realizar el ensamblaje completo del prototipo.

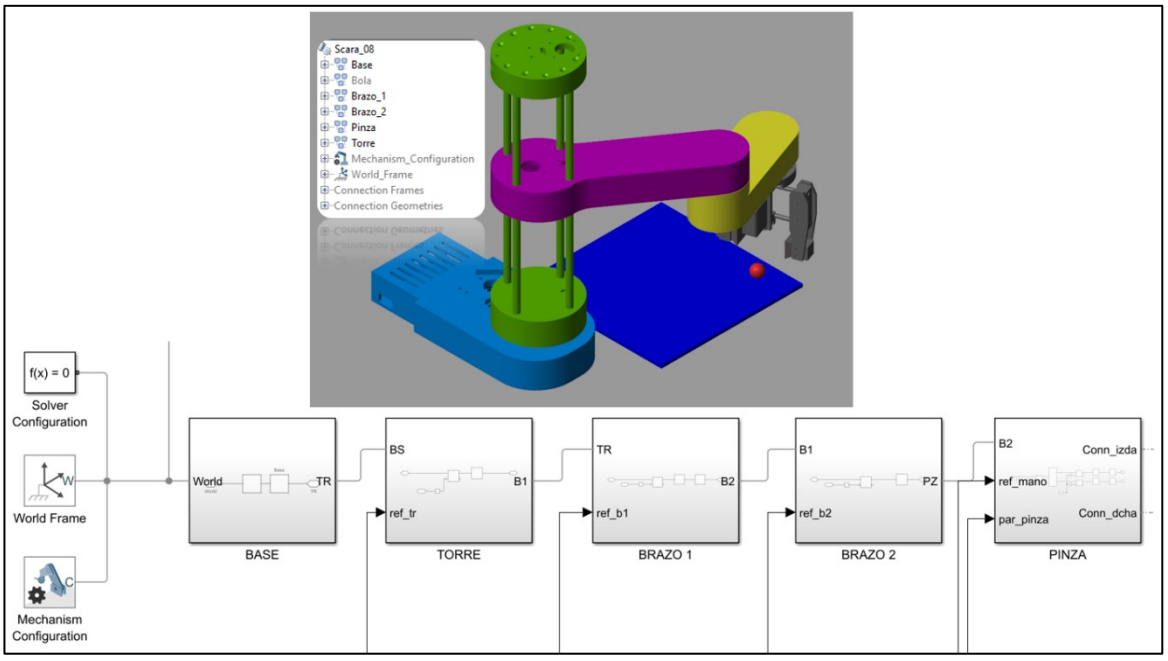


Figura 21. Esquema y Ensamblaje de todos los elementos del robot

De todos ellos, el único bloque que merece la pena resaltar, dada su laboriosa formación, es el de la pinza. En la *figura 22* se representa el esquema de Simulink utilizado y en este se puede apreciar que la pinza se ha formado con tres subconjuntos, la base principal y las dos partes que forman la garra. Se observa que toda la pinza gira gracias a una junta de revolución (JR3), mientras que las dos partes de la garra se desplazan sobre unas juntas prismáticas.

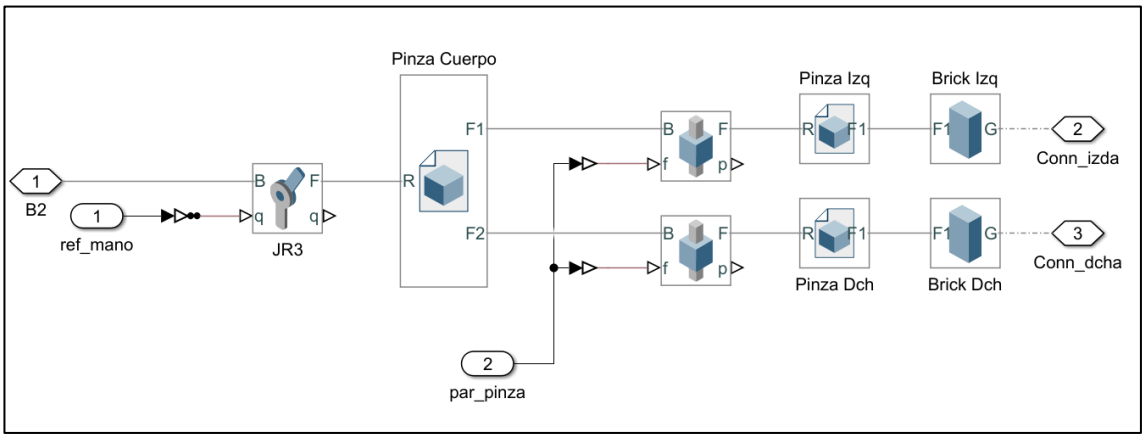


Figura 22. Ensamblaje de la pinza en Simscape

5.2.2 Interacción entre Objetos y Fuerzas de Contacto

Uno de los objetivos principales de este proyecto es que el robot sea capaz de coger una bola situada aleatoriamente sobre una pantalla táctil. Esto implica que el modelo debe simular adecuadamente el contacto entre los componentes que van a interactuar y con este propósito, la librería ofrece un bloque denominado *"Spatial Contact Force"*.

Este bloque se encarga de aplicar fuerzas de contacto normales y de fricción entre las dos geometrías a las que está conectado el bloque. Las fuerzas aplicadas son iguales y opuestas y se sitúan a lo largo de una línea de acción común. Tal y como se aprecia en la *figura 23*, este bloque necesita que se le definan los siguientes coeficientes: rigidez, amortiguamiento, rozamiento dinámico y rozamiento estático.

Todos estos parámetros se han ajustado mediante ensayo-error, hasta encontrar los parámetros con los que la simulación funciona y los componentes contactan de forma adecuada.

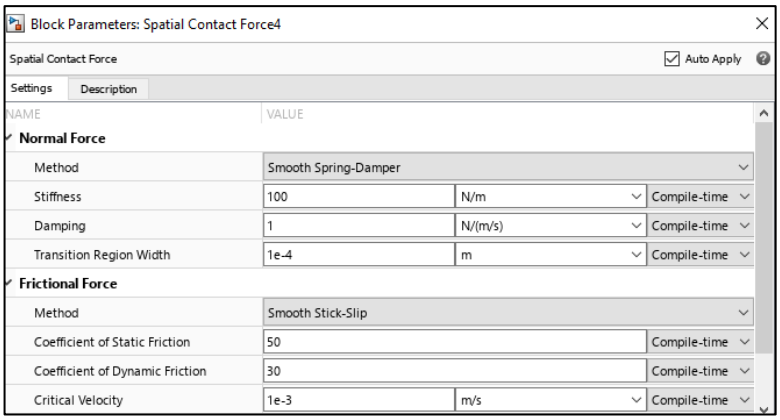


Figura 23. Configuración de los bloques Spatial Contact Force

En la simulación, la bola contacta con cuatro elementos: la pantalla táctil, el soporte final y las dos garras de la pinza. De esta forma, tal y como se aprecia en la *figura 24*, ha sido necesario utilizar cuatro de estos bloques.

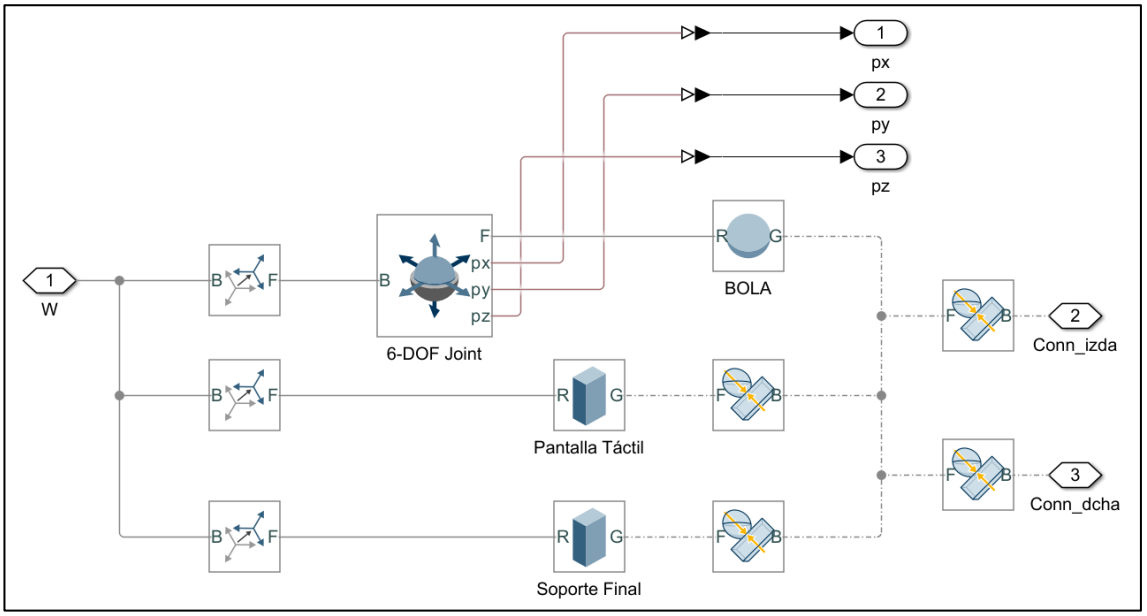


Figura 24. Modelado del contacto de la bola en Simscape

Por último, hay que añadir que, dado que la bola puede moverse libremente por el espacio, es necesario que disponga de una "junta" con 6 grados de libertad de movimiento (6-DOF Joint).

5.3 Control de las Articulaciones

Como ya se ha comentado previamente, el movimiento del robot se genera actuando sobre los bloques de articulación. Dado que este prototipo es accionado mediante motores paso a paso y que estos no necesitan de un control en bucle cerrado, se ha optado por enviar los comandos de movimiento a través de la opción denominada "Motion" de cada junta.

De esta forma, a cada articulación del robot se le envían los puntos objetivo que debe alcanzar en forma de referencias en escalón que se suceden en el tiempo y se introducen unos bloques de "*rate limiter*" para que se generen unos movimientos suaves y coordinados.

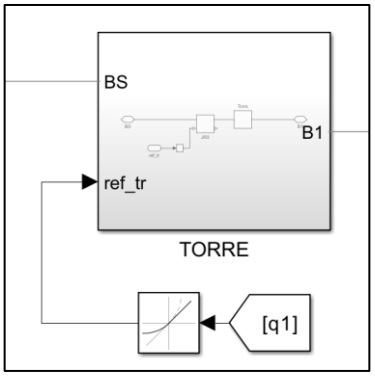
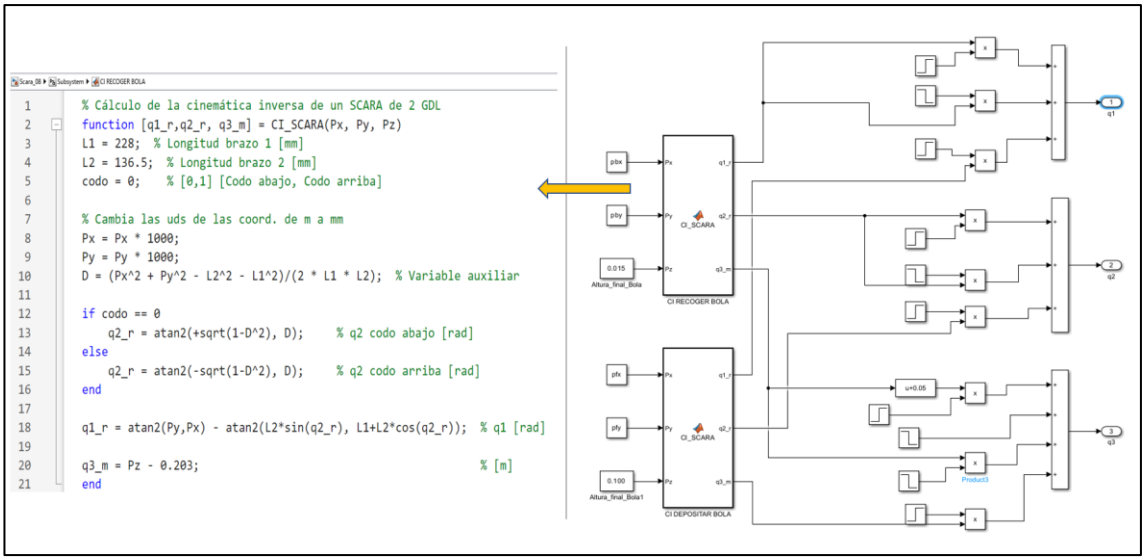


Figura 25. Control del movimiento de la primera articulación del robot

También se ha comentado que la bola puede caer en cualquier parte de la pantalla táctil y que el robot debe cogerla y depositarla en la superficie final, de la cual también se puede elegir la posición en el espacio. Por este motivo, en el esquema de *Simulink* se ha introducido un bloque denominado "Control Articulaciones" que, mediante las ecuaciones de cinemática inversa expuestas en el *apartado 2*, se encarga de calcular todos los puntos de la trayectoria que debe seguir el robot, tanto para coger la bola como para depositarla. En la *figura 26* se representa el subsitema encargado de calcular los puntos y generar la secuencia de movimientos en el tiempo.



```
% Cálculo de la cinemática inversa de un SCARA de 2 GDL
function [q1_r,q2_r, q3_m] = CI_SCARA(Px, Py, Pz)
L1 = 228;  % Longitud brazo 1 [mm]
L2 = 136.5;  % Longitud brazo 2 [mm]
codo = 0;    % [0,1] [Codo abajo, Codo arriba]

% Cambia las uds de las coord. de m a mm
Px = Px * 1000;
Py = Py * 1000;
D = (Px^2 + Py^2 - L2^2 - L1^2)/(2 * L1 * L2);  % Variable auxiliar

if codo == 0
    q2_r = atan2(+sqrt(1-D^2), D);     % q2 codo abajo [rad]
else
    q2_r = atan2(-sqrt(1-D^2), D);     % q2 codo arriba [rad]
end

q1_r = atan2(Py,Px) - atan2(L2*sin(q2_r), L1+L2*cos(q2_r));  % q1 [rad]

q3_m = Pz - 0.203;                                           % [m]
end
```

Figura 26. Generación de la trayectoria de simulación

5.4 Esquema Completo y Simulaciones

Atendiendo a todo lo expuesto anteriormente, el modelo completo de simulación se presenta en la *figura 27.*

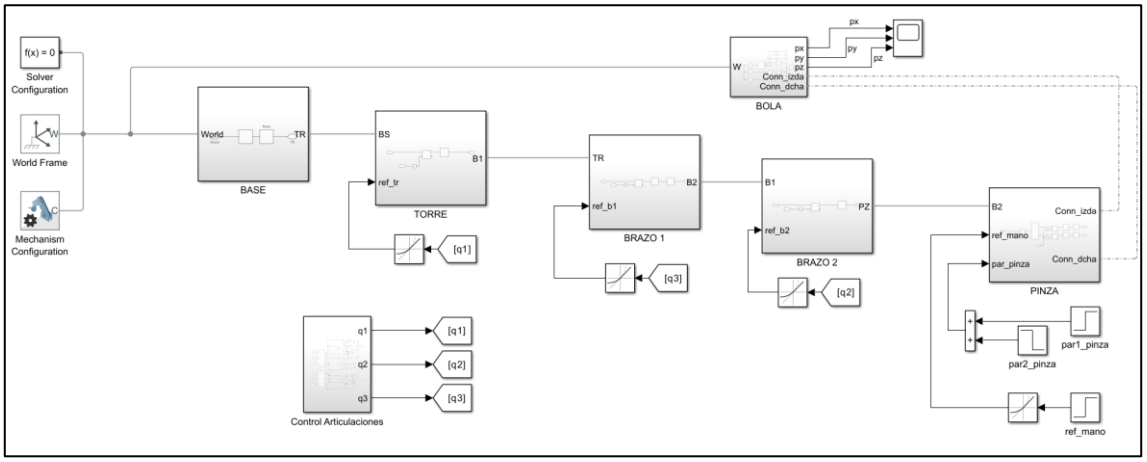


Figura 27. Esquema Completo del Robot en SIMSCAPE

Una vez completado todo el modelo y habiendo verificado su correcto funcionamiento, se puede pasar a realizar las simulaciones correspondientes.

El objetivo principal de esta simulación es realizar, de forma idéntica a la realidad, el funcionamiento del robot en modo automático, es decir, recoger, trasladar y depositar una bola que cae de forma aleatoria sobre una pantalla táctil.

Para poder ejecutar adecuadamente esta simulación se ha hecho uso de un script de *Matlab* para dar valor a ciertos parámetros de la simulación. Además, como se observa en la *figura 28*, se ha utilizado la función ***"rand()"*** para generar unas coordenadas aleatorias para la posición en la que cae la bola, simulando así la acción humana de depositar la bola sobre la pantalla táctil. También se han añadido una serie de valores de tiempo que se utilizan para formar una trayectoria temporal de puntos objetivo para el robot.

```
%% Parámetros de Simulación
% Coordenadas de la placa táctil
ptx = 0;    pty = 0.217;    ptz = 0.0025; % [m]

% Coordenadas aleatorias de la bola [m]
pbx = (-1 + 2 * rand) * 0.1175
pby = 0.0995 + rand * 0.235
pbz = 0.05;

% Coordenadas de la placa final [m]
pfx = 0.200;    pfy = -0.250;   pfz = 0.050;

% Secuencia de tiempos [s]
t0 = 0.2;  t1 = 1;  t2 = 1.3;   t3 = 2.3;  t4 = 3.5;
t5 = 4.5;  t6 = 5;  tsim = t6;

m = 3;  % Rampa de aceleración
```

Figura 28. Script de Matlab con los parámetros adicionales de la simulación

En las siguientes figuras e imágenes se pueden observar las diferentes posiciones por las que pasa el robot, las coordenadas donde cae la bola y la posición, velocidad y aceleración tanto de las cuatro articulaciones como del efector final.

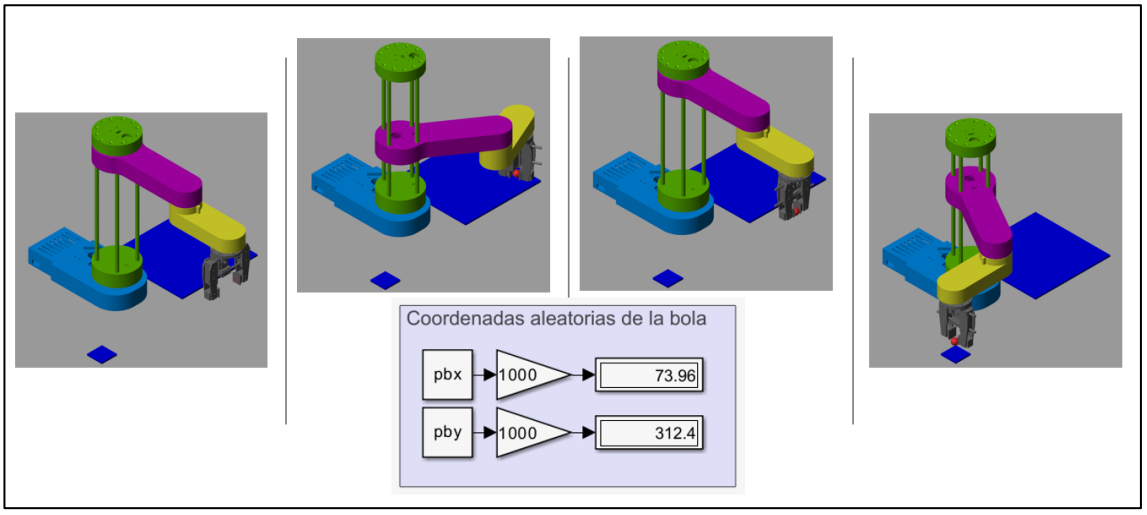


Figura 29. Coordenadas iniciales de la bola y simulación del movimiento completo

En la *figura 30* se representan las posiciones, velocidades y aceleraciones con respecto al tiempo. En ella se puede apreciar el punto de partida en el que el robot empieza, y la secuencia de movimientos que alcanza en los tiempos establecidos para completar toda la trayectoria. También se puede observar la evolución de las velocidades respondiendo a los cambios de aceleración.

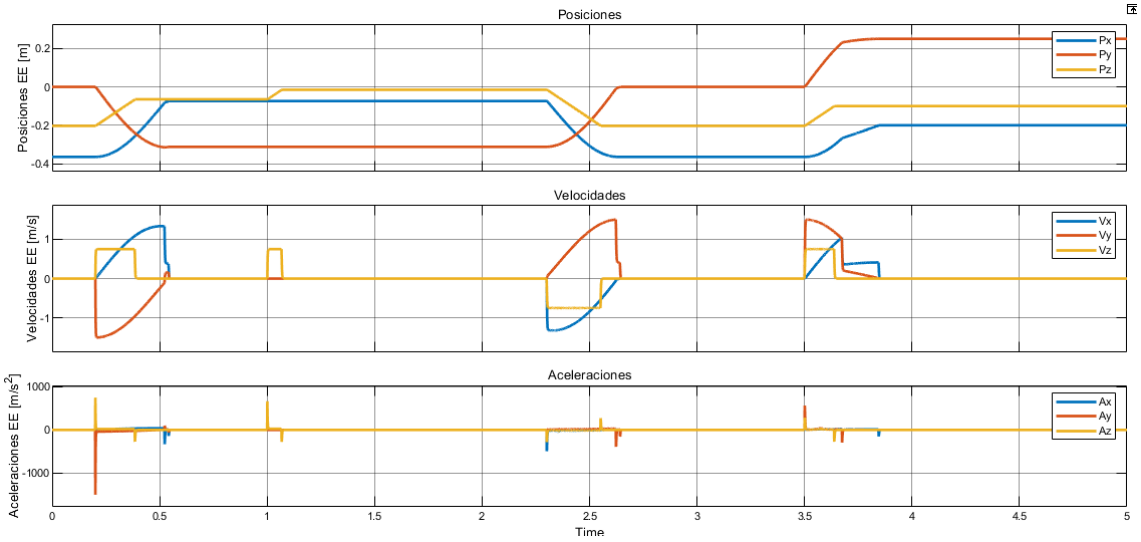


Figura 30. Posiciones, velocidades y aceleraciones del efector final

En las siguientes figuras se representan las posiciones, velocidades y aceleraciones de las tres juntas más representativas y, tras comprobar en simulación que todo funciona correctamente, se puede pasar a ensamblar y configurar el prototipo real.

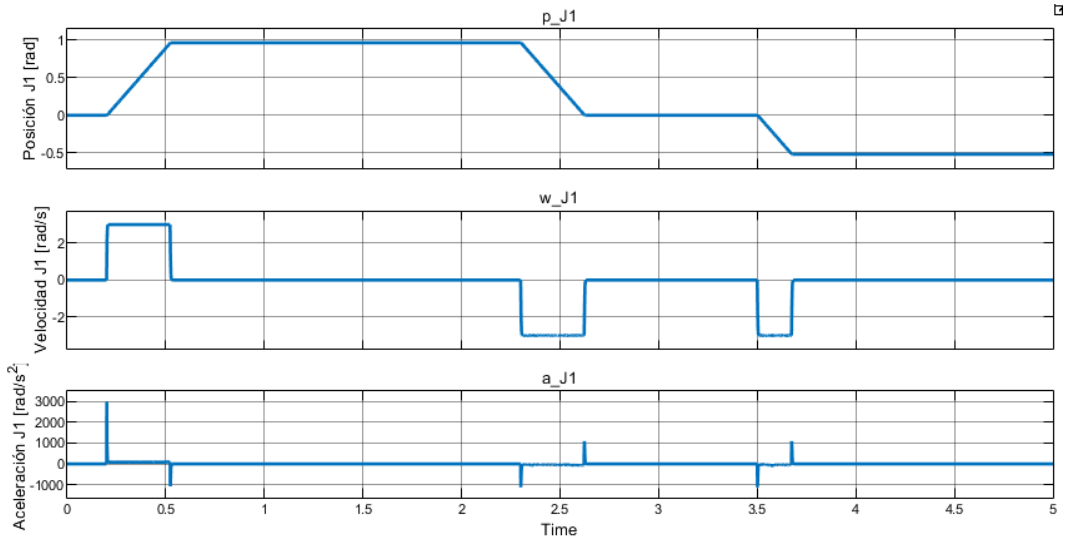


Figura 31. Posición, velocidad y aceleración de la primera junta de revolución (J1)

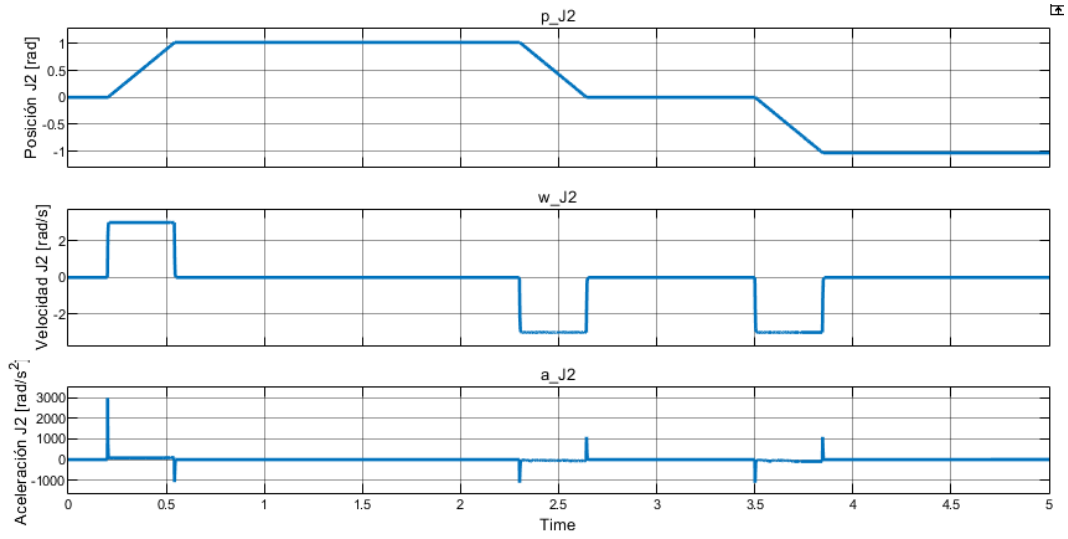


Figura 32. Posición, velocidad y aceleración de la segunda junta de revolución (J2)

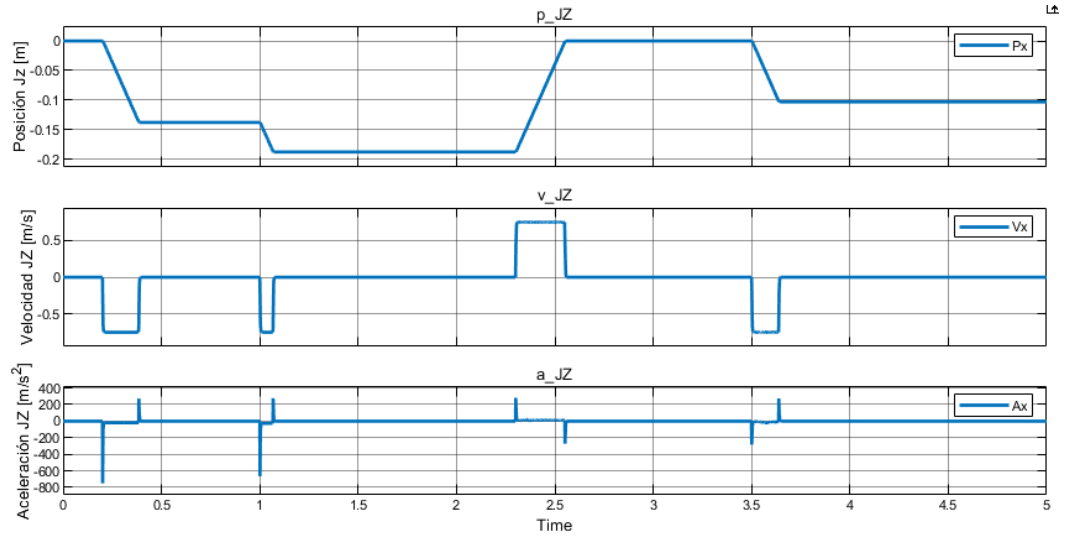


Figura 33. Posición, velocidad y aceleración de la junta prismática (JZ)

6 Implementación Real

En esta sección, se llevará a cabo un detallado examen de los aspectos relacionados con la fabricación de las piezas y el completo ensamblaje mecánico y eléctrico del prototipo. Todo este proceso estará respaldado por un estudio exhaustivo de los componentes electrónicos y de la pantalla táctil resistiva. Se realizará un análisis detallado del esquema eléctrico y todas las conexiones, y se presentarán diversos programas de prueba que permiten verificar de manera individual el correcto funcionamiento de los diferentes sistemas y componentes. De esta forma, se proporcionará una visión completa y rigurosa del proceso de fabricación, ensamblaje y validación del prototipo, asegurando su funcionamiento óptimo y confiable.

6.1 Componentes Mecánicos y Estructurales

En primer lugar, es importante destacar que la fabricación de todas las piezas estructurales se llevó a cabo de manera propia y simultánea a la elaboración de los apartados anteriores, utilizando una impresora 3D modelo Ender 3 y el material PLA (ρ=1.24 kg/m^3). Para este propósito, se utilizó el programa CURA, que permite preparar los modelos de impresión y generar el GCODE correspondiente a cada pieza.

Los parámetros de impresión se ajustaron gradualmente, pieza por pieza. En líneas generales, se optó por velocidades de impresión bajas (v=20 mm/s) para garantizar una buena calidad, y se utilizó un porcentaje de relleno considerable (40%) para asegurar la resistencia de las piezas.

Uno de los aspectos más desafiantes fue ajustar la tolerancia de impresión de las piezas. Después de numerosas pruebas de calibración, se decidió utilizar el parámetro "expansión horizontal de orificios". Este ajuste permitió obtener tolerancias de ± 0.3 mm en todas las dimensiones, un valor notable considerando que se trata de una impresora de gama baja. En la *figura 34* se muestran algunas de estas piezas.

Figura 34. Fabricación propia de las piezas mediante impresión 3D

Además, se adquirieron componentes comerciales como husillos, rodamientos, tornillería y correas a través distintos distribuidores de internet. Una vez que se disponía de todos los componentes y piezas necesarias, se procedió al montaje del prototipo.

Es importante destacar que este trabajo no tiene la intención de proporcionar una guía detallada sobre los aspectos relacionados con el ensamblaje mecánico. Por lo tanto, solo se mencionará brevemente la secuencia de montaje de las diferentes partes, acompañada de imágenes que ilustren el proceso.

La secuencia de montaje es la siguiente: en primer lugar, se montó la base, seguida de la torre de elevación. A continuación, se ensamblaron los brazos 1 y 2, y por último, se instaló la pinza.

En la *figura 35*, se encuentran imágenes que ilustran el proceso de montaje de las distintas partes.

Figura 35. Montaje mecánico de los distintos subensamblajes del robot

Encuanto a la tornillería utilizada, la *figura 36* recopila todas las métricas que se han utilizado.

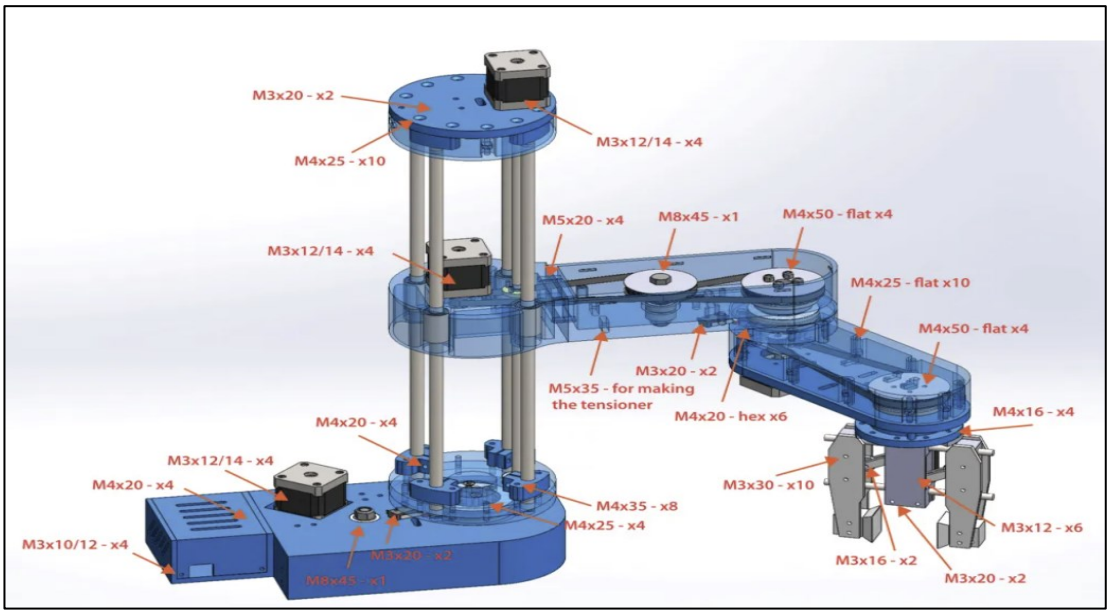


Figura 36. Tornillería utilizada en el prototipo

6.2 Componentes electrónicos

A continuación, se enumeran todos los componentes electrónicos que forman parte del robot. El ensamblaje de algunos de estos elementos, como los motores, el cableado y los finales de carrera, se realizó simultáneamente al montaje mecánico. Esta práctica fue necesaria debido a que el robot incorpora ciertas partes a las que no se puede acceder una vez que el ensamblaje está completo.

6.2.1 Arduino MEGA

El Arduino MEGA se erige como una plataforma de desarrollo versátil y potente para el control y coordinación de los diferentes componentes electrónicos del brazo robótico SCARA. Diseñado con una arquitectura basada en microcontrolador, el Arduino MEGA destaca por sus características avanzadas y su amplia capacidad de procesamiento.

Una de las características principales del Arduino MEGA es su alta capacidad de entrada/salida digital y analógica, lo cual permite conectar y controlar eficientemente los diferentes componentes electrónicos del brazo robótico. Gracias a sus 54 pines digitales y 16 pines analógicos, ofrece una versatilidad excepcional para interactuar con sensores, actuadores y otros dispositivos periféricos. Esta característica ha resultado determinante para seleccionar este modelo de Arduino, puesto que la placa Arduino Uno no disponía de suficientes pines para controlar todos los elementos.

Además, el Arduino MEGA dispone de una memoria flash de 256 KB, lo que brinda un amplio espacio para almacenar el programa de control del robot y los datos necesarios para su correcto funcionamiento. Esto resulta especialmente relevante en proyectos de mayor envergadura, como el diseño e implementación de un brazo robótico con múltiples grados de libertad.

Otra característica destacada del Arduino MEGA es su capacidad de procesamiento, incorporando un reloj interno de 16 MHz que resulta adecuado para este proyecto. También incorpora puertos de comunicación serial UART, SPI e I2C, que permiten la interacción con otros dispositivos y sistemas en red. Asimismo, gracias a la conexión USB integrada y al soporte de comunicación a través de RS-232, es posible establecer una interfaz entre el robot y el panel de control del PC.

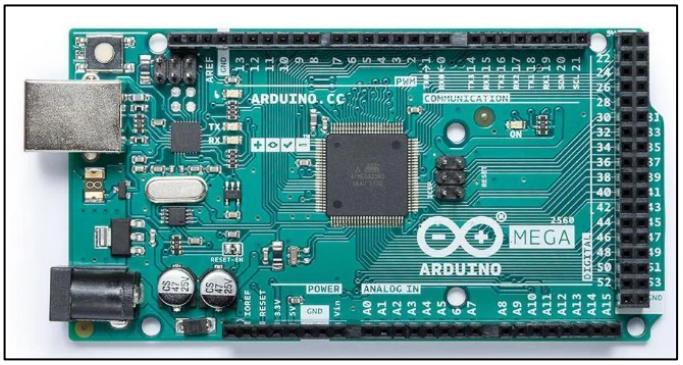


Figura 37. Microcontrolador Arduino MEGA

6.2.2 CNC-Shield

La CNC-Shield es una placa de control altamente utilizada en el ámbito de la robótica y la mecatrónica y desempeña un papel clave en el diseño e implementación de este brazo robótico. Esta placa proporciona una interfaz entre el Arduino MEGA y los componentes electrónicos esenciales para el control y movimiento preciso del robot.

La CNC-Shield se destaca por su capacidad para controlar hasta cuatro motores paso a paso, que son utilizados en el brazo robótico para generar los movimientos articulados. Estos motores, conocidos por su precisión y torque, son ideales para aplicaciones que requieren movimientos controlados y repetibles.

Además de facilitar el control de los motores paso a paso, la CNC-Shield también se encarga de la alimentación de los motores, asegurando que reciban la potencia adecuada para su funcionamiento. Asimismo, la placa permite conectar y gestionar los drivers A4988, los cuales actúan como amplificadores de corriente para los motores, brindando la capacidad de controlar la velocidad y la dirección de estos.

Otra funcionalidad importante de la CNC-Shield es su capacidad para conectar los finales de carrera. Estos dispositivos, también conocidos como interruptores de límite, se utilizan para detectar y limitar el movimiento del brazo robótico en puntos específicos. Al detectar la posición límite de cada articulación, los finales de carrera garantizan la seguridad y evitan daños al robot y a su entorno.

En conjunto con el Arduino MEGA, la CNC-Shield se convierte en un enlace crucial que une la capacidad de procesamiento y control del microcontrolador con la funcionalidad específica de los motores y los finales de carrera, permitiendo así el funcionamiento adecuado del prototipo.

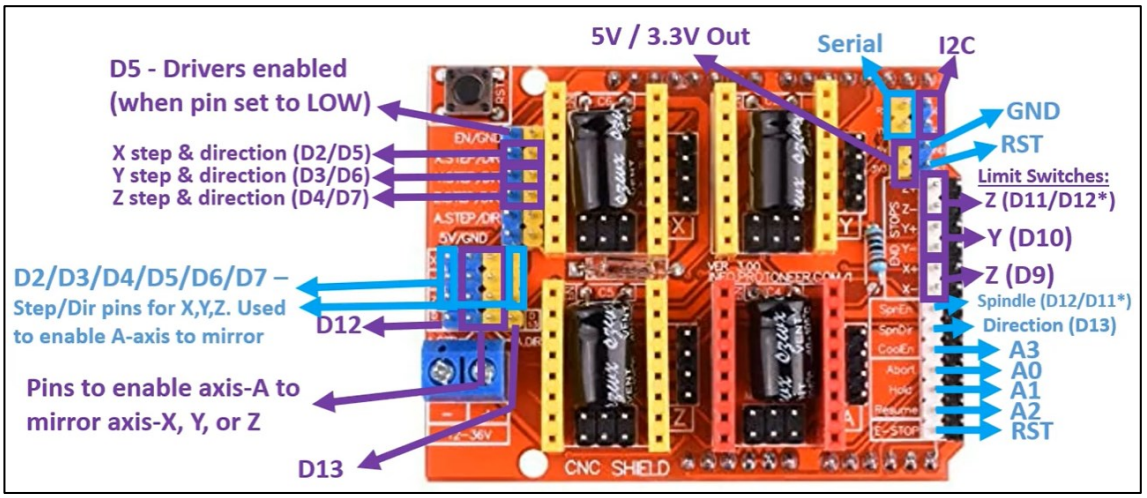


Figura 38. Esquema CNC-Shield con explicación de pines

6.2.3 Motores paso a paso NEMA 17

Los motores paso a paso NEMA 17 son ampliamente utilizados en aplicaciones de robótica y automatización debido a sus características de precisión y control de movimiento. Estos motores se caracterizan por su tamaño estandarizado, conforme a la norma NEMA, con una dimensión de 1.7 x 1.7 pulgadas.

Una de las principales ventajas de los motores paso a paso NEMA 17 es su capacidad para realizar movimientos angulares con alta precisión. Estos motores se basan en un diseño electromecánico que convierte señales eléctricas en movimientos discretos y medibles, denominados "pasos". Cada paso corresponde a un ángulo específico de rotación, lo que permite un control preciso sobre la posición y el desplazamiento del brazo robótico.

Otra característica destacada de los motores paso a paso NEMA 17 es su alto torque en relación con su tamaño compacto. Esto los hace especialmente adecuados para aplicaciones que requieren fuerza y resistencia, permitiendo al brazo robótico realizar movimientos precisos incluso cuando está cargado con objetos o realizando tareas que requieren fuerza.

En cuanto a su control, los motores paso a paso NEMA 17 requieren de drivers específicos, como los drivers A4988 de los cuales se hablará en el siguiente apartado.

Dado que se ha realizado un estudio exhaustivo de este tipo de motores en el *apartado 3*, no se cree necesario comentar nada más.

Figura 39. Motores paso a paso NEMA 17

6.2.4 Drivers A4988

Los drivers A4988 son los encargados de realizar el control de los motores paso a paso NEMA 17. Estos drivers son dispositivos de control de corriente que permiten alimentar y gestionar el movimiento de los motores de manera precisa y eficiente.

Una de las principales características de los drivers A4988 es su capacidad para proporcionar una corriente ajustable a los motores paso a paso. Esto es especialmente importante ya que permite adaptar la corriente suministrada a las necesidades específicas de cada motor y optimizar su rendimiento. Un ajuste adecuado de la corriente contribuye a prevenir sobrecalentamientos y asegura un funcionamiento seguro y eficaz.

Tal y como se ha calculado en el *apartado 3,* los valores de ajuste de esta corriente, atendiendo a las necesidades de par de cada articulación, se presenta en la siguiente tabla.

Art.	Transmisión	Vel. Max [rpm]	Par motor [Nm]	Límite de Intensidad [A]
1	Polea-Correa	300	0.075	1
2	Husillo	300	0.096	1.4
3	Polea-Correa	300	0.033	0.5
4	Polea-Correa	300	0.045	0.65

Tabla 6. Ajuste de la intensidad máxima de cada driver

Además de su capacidad para controlar la corriente, los drivers A4988 también ofrecen otras funcionalidades importantes. Por ejemplo, permiten controlar la dirección del movimiento de los motores, lo que facilita el control de las articulaciones del brazo robótico en distintas direcciones. Asimismo, ofrecen la posibilidad de ajustar la velocidad de los motores, lo que permite adaptarla a las necesidades específicas de cada tarea.

Otro aspecto destacado de los drivers A4988 es su compatibilidad con el Arduino MEGA y la CNC-Shield utilizada en este proyecto. Estos drivers se pueden conectar fácilmente a la CNC-Shield y se integran de manera fluida con el Arduino MEGA, lo que simplifica el proceso de configuración y control de los motores paso a paso.

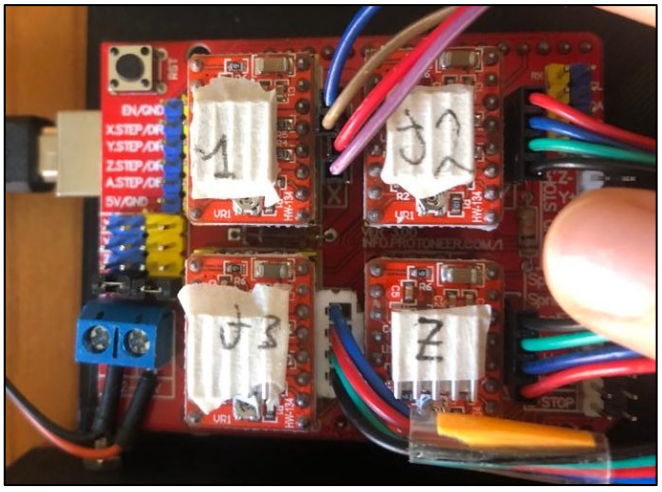


Tabla 7. Drivers A4988 conectados a la CNC-Shield

6.2.5 Fuente de Alimentación

Se ha seleccionado una fuente de alimentación de 12 V y 6 A para generar el suministro de energía necesaria para el funcionamiento adecuado de los motores. Esta fuente de alimentación proporciona una tensión continua de 12 voltios y una capacidad de corriente de hasta 6 amperios, lo que garantiza una alimentación estable y suficiente para los motores y otros componentes electrónicos involucrados en el sistema.

La fuente de alimentación se encarga de suministrar energía a través de la CNC-Shield, actuando como un vínculo crucial en la cadena de suministro de energía. Conectada a la CNC-Shield, la fuente de alimentación permite que la corriente eléctrica fluya hacia los motores paso a paso NEMA 17 de manera controlada y segura.

Es importante destacar que los motores paso a paso requieren una cantidad significativa de corriente para su correcto funcionamiento, especialmente cuando se enfrentan a cargas o tareas que demandan un mayor esfuerzo. La fuente de alimentación de 12 V y 6 A cumple con este requisito al proporcionar una corriente suficiente para que los motores funcionen de manera eficiente y puedan generar el torque necesario para los movimientos articulados del brazo robótico.

Además de alimentar los motores, la fuente de alimentación también suministra energía a otros componentes electrónicos del sistema, como la propia CNC-Shield y los finales de carrera, pero es importante remarcar que la placa Arduino MEGA se alimenta de forma independiente a través del cable USB conectado al ordenador.

Figura 40. Fuente de alimentación de 12 V y 6 A

6.2.6 Servomotor MG996R

El Servomotor MG996R se encarga de controlar el movimiento de apertura y cierre de la pinza, permitiendo así la capacidad de agarrar y soltar objetos de manera precisa y controlada.

Los servomotores, en general, son dispositivos electromecánicos que se utilizan ampliamente en aplicaciones de robótica y automatización debido a su capacidad para controlar el ángulo de giro y la posición de un eje.

El rango de movimiento de 180 grados permite abrir y cerrar la pinza con precisión, lo que facilita la tarea de agarrar y soltar objetos de diferentes tamaños y formas. Además, su alto torque garantiza un agarre seguro y estable, evitando posibles caídas o deslizamientos indeseados de los objetos manipulados.

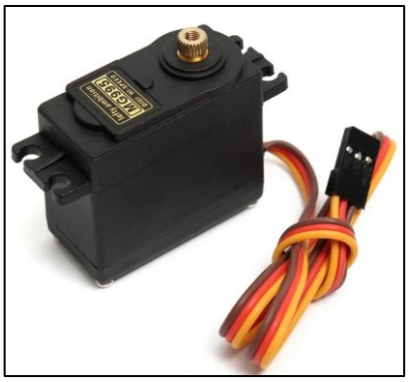


Figura 41. Servomotor MG996R

6.2.7 Finales de Carrera

Los finales de carrera desempeñan un papel importante en el control y la seguridad del brazo robótico. Cada articulación del brazo robótico incorpora un solo final de carrera, es decir, se controla el límite de movimiento solo en una de las direcciones. Por este motivo, es muy importante controlar adecuadamente y de forma segura el movimiento en la otra dirección a través de software.

Todos los finales de carrera se han conectado en la configuración de contacto normalmente cerrado. Esto significa que cuando el final de carrera no está pulsado, se recibe un valor lógico 1, y cuando se pulsa, se recibe un valor lógico 0.

Además, tal y como se verá posteriormente, se ha introducido por software un sistema de lectura con antirrebotes, que añade robustez al sistema de seguridad.

Al configurar los finales de carrera de esta manera, se establece un mecanismo de detección de límites que permite al sistema saber cuándo una articulación ha alcanzado su posición límite en una dirección determinada. Cuando el final de carrera correspondiente se activa, es decir, se pulsa, se envía una señal al controlador del brazo robótico para que detenga el movimiento en esa dirección y evite daños o situaciones inseguras.

Además, estos elementos se utilizan principalmente en el ciclo de homing del robot. Dado que los motores paso a paso no incorporan encoder, el uso de los finales de carrera es el único método disponible para calibrar la posición de cada articulación.

Figura 42. Final de Carrera

6.2.8 Módulo de Comunicaciones FT232RL

El módulo FT232RL Mini USB a TTL Conversor Serial es un dispositivo que actúa como un convertidor de señal entre la interfaz USB y el protocolo de comunicación TTL (Transistor-Transistor Logic). Su función principal es permitir la comunicación bidireccional entre un dispositivo USB, como el ordenador, y un dispositivo TTL, en este caso, el Arduino.

Este módulo de comunicaciones ha desempeñado un papel crucial al abrir un segundo puerto de comunicación serie entre el Arduino y el ordenador. Esta solución se volvió necesaria debido a que el puerto serie principal de la placa Arduino MEGA ya estaba asignado para mostrar datos en la pantalla, informando al usuario sobre los movimientos del robot. Por lo tanto, el módulo FT232RL se utilizó específicamente para establecer un puerto adicional que se dedicó exclusivamente a la transmisión de comandos de movimiento desde la interfaz del panel de mando, lo que mejoró significativamente la comunicación y el control del brazo robótico.

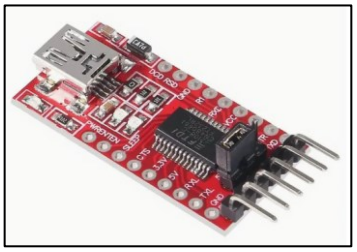


Figura 43. Módulo de comunicaciones FT232RL

6.2.9 Pantalla Táctil Resistiva

La pantalla táctil se ha utilizado para determinar la posición en la que cae la bola. Se trata de una pantalla táctil resistiva de cinco hilos que consta de dos placas resistivas transparentes separadas por espaciadores aislantes. La placa superior contiene un contacto metalizado y sirve como nodo de detección de voltaje. Las cuatro esquinas de la placa inferior se utilizan para producir gradientes de tensión en las direcciones “x” e “y” (ver *figura 44*). Se utiliza una configuración de polarización específica para cada eje, el “x” y el “y”, tal y como se aprecia en la tabla 8. Una vez generados los gradientes adecuados, la pantalla envía una señal analógica por el quinto hilo, que se utiliza para determinar las coordenadas de la bola.

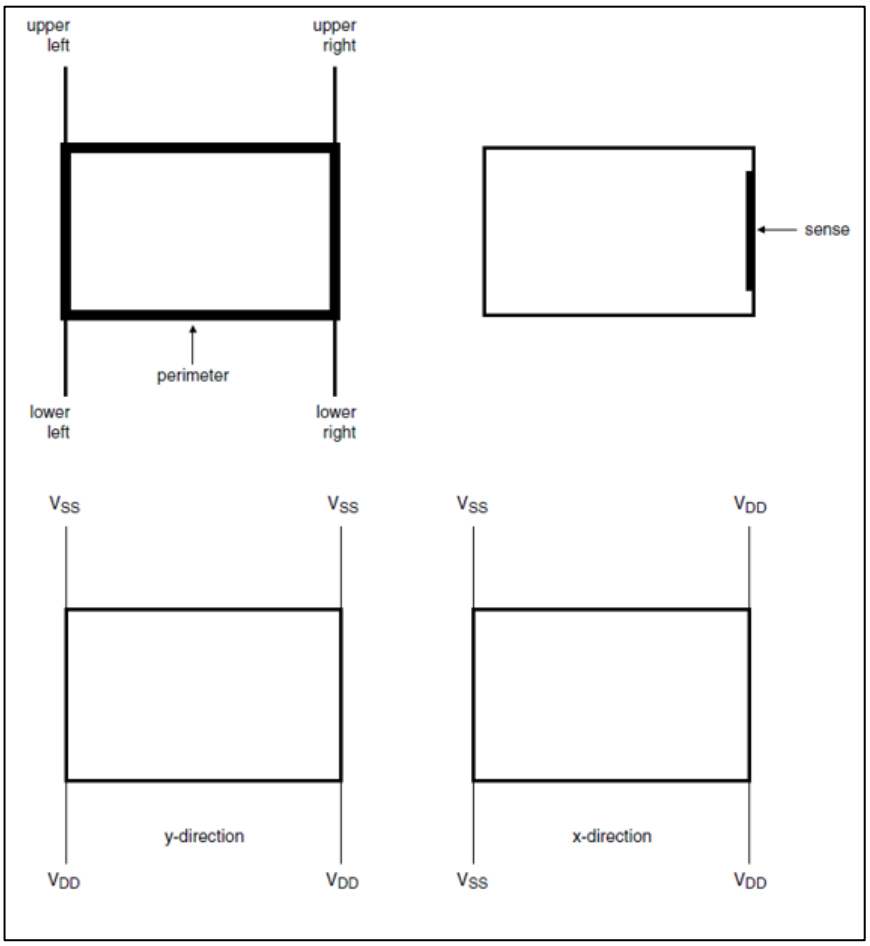


Figura 44. Esquema de la pantalla táctil resistiva de 5 hilos

	touchscreen signal				
function	**upper left**	**lower left**	**upper right**	**lower right**	**sense**
hardware touch detection	Vss	Vss	Vss	Vss	logic zero interrupt
read x-position	Vss	Vss	V_{DD}	V_{DD}	voltage measurement
read y-position	Vss	V_{DD}	Vss	V_{DD}	voltage measurement

Tabla 8. Valores de voltaje para lecturas de posición en los ejes X e Y de la pantalla

6.2.10 Esquema Completo de Conexiones

Con el objetivo de facilitar la comprensión del esquema, este se ha dividido en dos partes.

En la *figura 45* se reúnen todos los elementos que se conectan directamente con la CNC-Shield. Estos son: los 4 motores paso a paso, el servomotor, los cuatro finales de carrera y la fuente de alimentación de los motores.

En la *figura 46* se representan los elementos que van directamente conectados a los pines del Arduino MEGA y que son el módulo de comunicación serie y la pantalla táctil resistiva.

Además, se debe conectar la CNC-Shield con la placa Arduino MEGA, montándolas una sobre la otra.

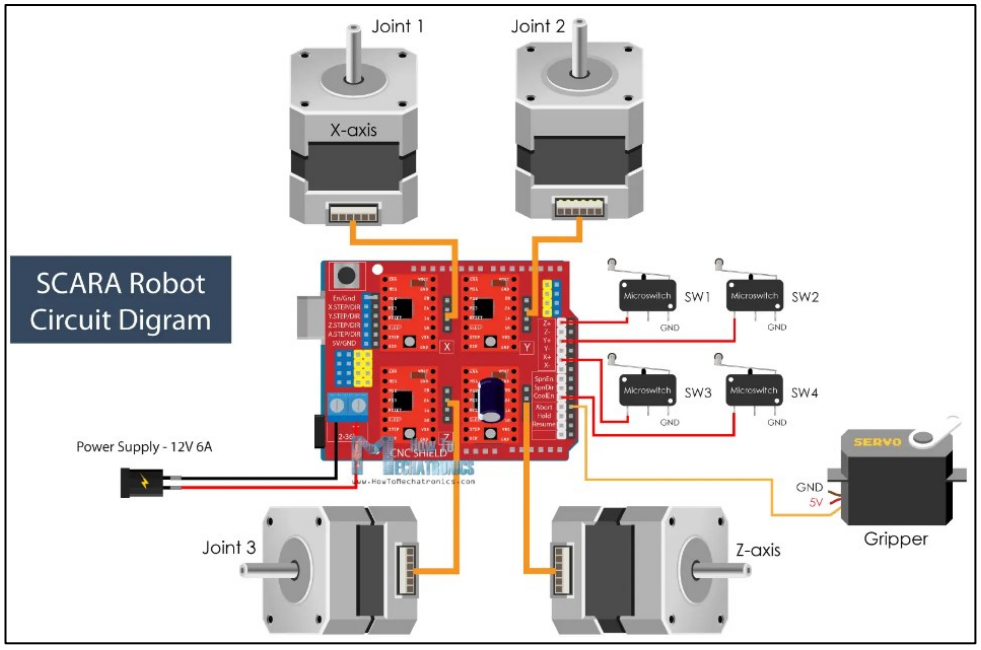


Figura 45. Esquema de conexión de los componentes de la CNC-Shield

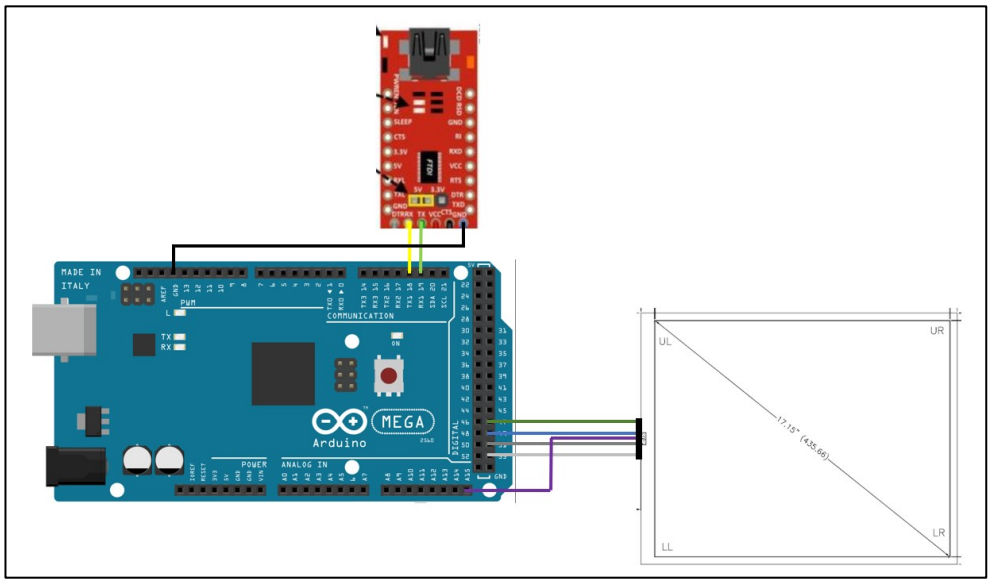


Figura 46. Esquema de conexiones de los componentes del Arduino MEGA

6.3 Montaje Completo del Prototipo

Teniendo claro cuáles son todos los elementos que componen el robot, tanto mecánicos como eléctricos y habida cuenta de las pertinentes conexiones, tan solo queda realizar el ensamblaje completo. En la siguiente figura se muestra el resultado final del prototipo de robot SCARA de 4 grados de libertad.

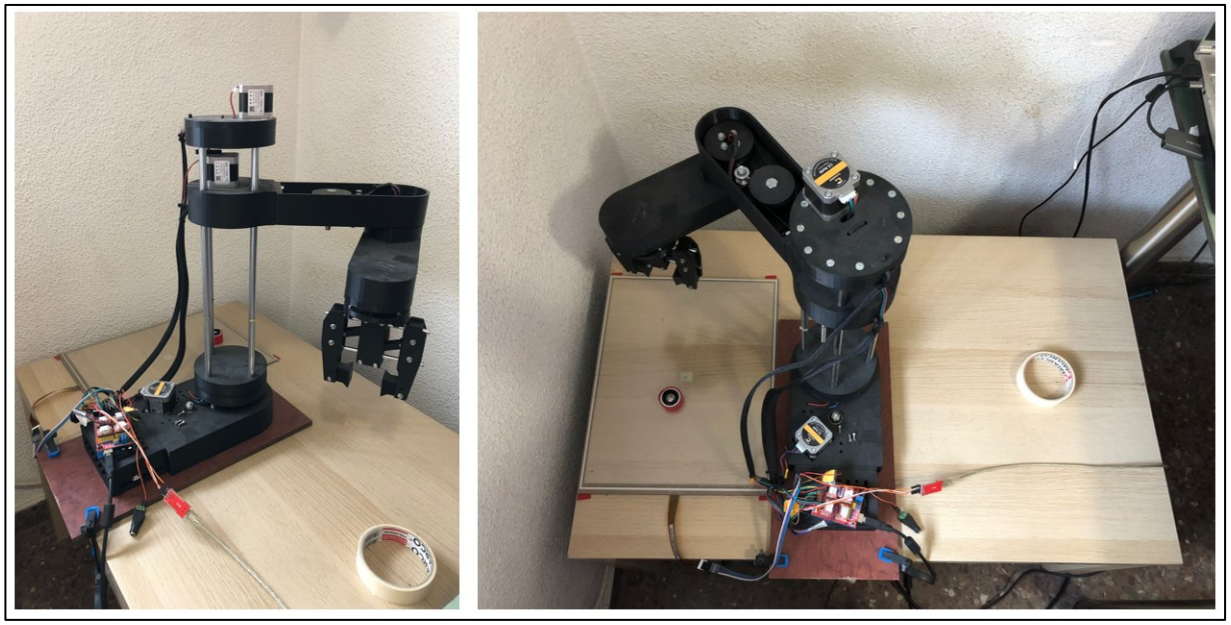


Figura 47. Montaje completo del prototipo

6.4 Pruebas de funcionamiento de los sistemas

Una vez ensamblado todo el prototipo se deben comprobar todos los sistemas de forma individual. Es muy importante ir generando los programas y las funciones de control de cada una de las partes de forma individual para testear que todo funciona correctamente.

Con este objetivo, en los siguientes apartados se van a presentar pequeños programas de Arduino que se encargan de controlar los sistemas individuales.

6.4.1 Motores paso a paso

Para comprobar que los 4 motores paso a paso funcionan, se ha creado un pequeño programa que los mueve en ambas direcciones.

```
// Programa: Control de un stepper sin librería

// #defines
#define enDrivers 8  // Pin para habilitar los drivers
#define stepPin 3    // Pin para enviar los pulsos
#define dirPin 6     // Pin para indicar la direción
#define tiempo 3000

void setup() {
  Serial.begin(9600);
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  pinMode(enDrivers, OUTPUT);
  digitalWrite(enDrivers, LOW);  // Habilita el Driver
}

void loop() {

  // Movimiento del motor 200 pulsos en dirección horaria
  digitalWrite(dirPin, HIGH);
  for (int x = 0; x < 200; x++) {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(tiempo);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(tiempo);
  }
  delay(1000);

  // Movimiento del motor 200 pulsos en dirección anti-horaria
  digitalWrite(dirPin, LOW);
  for (int x = 0; x < 200; x++) {
    Serial.println(x);
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(tiempo);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(tiempo);
  }
  delay(1000);
}
```

Figura 48. Programa de prueba de los motores paso a paso

6.4.2 Finales de Carrera

Para testear el funcionamiento de los finales de carrera se ha generado una pequeña función que, implementando un sistema anti-rebotes, comprueba si los finales de carrera se han activado.

```
// PROGRAMA: Funcionamiento de los Finales de Carrera

// #defines
#define FC_Z 0
#define FC_J1 1
#define FC_J2 2
#define FC_J3 3

// Variables globales
int vFC [4] = {A3, 9, 10, 11};
int estado_anterior [4] = {0, 0, 0, 0};

void setup() {
  Serial.begin(9600);
  // se de claran enmodo "pull-up" para su correcta lectura
  pinMode(vFC[FC_Z], INPUT_PULLUP);
  pinMode(vFC[FC_J1], INPUT_PULLUP);
  pinMode(vFC[FC_J2], INPUT_PULLUP);
  pinMode(vFC[FC_J3], INPUT_PULLUP);
}

void loop() {
  Serial.println(compruebaFC(FC_Z));
}

// Comprobación del FC con Anti-Rebotes
int compruebaFC(char FC) {
  int estado_actual = digitalRead(vFC[FC]);
  if (estado_actual != estado_anterior[FC]) {
    delay(5);
    estado_actual = digitalRead(vFC[FC]);
    if (estado_actual != estado_anterior[FC]) {
      estado_anterior[FC] = estado_actual;
    }
  }
  return estado_anterior[FC];
}
```

Figura 49. Programa de prueba de los Finales de Carrera

6.4.3 Servomotor

Para comprobar el servomotor se ha utilizado una librería de Arduíno denominada “servo.h” que permite generar una señal pwm en el pin de control del motor.

Tras realizar varias pruebas, se comprobó que el funcionamiento óptimo del servo (debido a la naturaleza del mecanismo en el que está incorporado) está entre 0 y 82º.

```
1   // PROGRAMA: Movimiento del servomotor
2
3   #include <Servo.h>  // Incluye la librería servo
4
5   Servo myservo;      // Crea una clase servo
6
7   void setup() {
8     myservo.attach(A0);  // Indicar el pin de control
9   }
10
11  void loop() {
12    cerrarPinza();
13    delay(1500);
14    abrirPinza();
15    delay(1500);
16  }
17
18  void cerrarPinza() {
19    for (int pos = 0; pos <= 82; pos ++) {myservo.write(pos); delay(15);}
20  }
21
22  void abrirPinza() {
23    for (int pos = 82; pos >= 0; pos --) {myservo.write(pos); delay(15);}
24  }
```

Figura 50. Programa de prueba del servomotor

Con estos fundamentos básicos, se ha desarrollado una función en el programa real llamada “***moverPinza(porcentajeObjetivo)”.*** Esta función (véase la línea 152 del programa de Arduíno) recibe un porcentaje de apertura deseado y lo transforma a los grados de movimiento del servo.

6.4.4 Pantalla táctil

Tal y como se ha comentado previamente, la pantalla táctil resistiva necesita que se alimenten sus cuatro extremos de una determinada manera dependiendo de si se quiere obtener la lectura del eje X o la del Y. El siguiente programa se encarga de dicha implementación.

Es muy importante tratar la señal en crudo que se obtiene. En este caso, la señal obtenida se mapea entre los valores en crudo máximos y mínimos de cada eje y los valores en milímetros que determinan la posición de cada eje de la pantalla en función del origen del robot.

```
1   // PROGRAMA: Lectura de la Pantalla táctil
2
3   // #defines
4   #define UR 47                 // UP RIGTH
5   #define LR 49                 // LOW RIGTH
6   #define UL 51                 // UP LEFT
7   #define LL 53                 // LOW LEFT
8   #define signal A15            // PIN de señal
9
10  // VARIABLES GLOBALES
11  float Pxb, Pyb;
12
13  void setup() {
14
15    Serial.begin(115200) ;
16
17    // PINES DIGITALES COMO SALIDAS
18    pinMode(UR,OUTPUT) ; pinMode(LR,OUTPUT) ;
19    pinMode(UL,OUTPUT) ; pinMode(LL,OUTPUT) ;
20
21    // SE ACTIVAN ADECUADAMENTE LOS QUE SIEMPRE TIENEN EL MISMO VALOR
22    digitalWrite(LR, LOW); digitalWrite(UL,HIGH );
23  }
24  void loop() {
25
26    // LECTURA DEL EJE X
27    digitalWrite(UR,HIGH) ; digitalWrite(LL,LOW); delay(1);
28    Pxb = map(analogRead(signal), 182, 821, -37429.0, -10189.0) / 100.0;
29
30    //  LECTURA DEL EJE Y
31    digitalWrite(UR,LOW) ; digitalWrite(LL,HIGH) ;
32    delay(1);
33    Pyb = map(analogRead(signal), 204, 808, -170.0 * 100.0, 170.0 * 100.0) / 100.0;
34
35    // VISUALIZACIÓN DE LECTURAS
36    Serial.print(Pxb) ; Serial.print("\t") ;
37    Serial.print(Pyb) ; Serial.println("\t") ;
38  }
```

Figura 51. Programa de prueba de la pantalla táctil

Para implementarlo en el programa general, se han creado dos funciones, ***"leerPx()" y "leerPy()"***, que son llamadas cuando el robot se encuentra en el modo de funcionamiento automático (véanse líneas 502 y 510 del código de arduíno).

7 Programación

En este apartado se van a presentar los aspectos del proyecto relacionados con la programación del Robot y del Panel de Mando. Dada su gran extensión, los programas completos se han incluido en el ANEXO I de tal forma que este apartado pretende servir como guía para una mejor comprensión de los programas. A pesar de esto, ambos códigos pueden ser fácilmente comprendidos por el lector puesto que incluyen numerosos comentarios a lo largo del propio código.

7.1 Funcionamiento General

El objetivo principal del presente trabajo consiste en implementar un prototipo de robot de tipo SCARA de 4 grados de libertad que, a través de una interfaz gráfica en el ordenador, facilite al usuario su control en modo manual y automático (véase la *figura 52*).

El modo manual permite operar todas las articulaciones de forma independiente (modo JOG) y también situar el efector final en una posición XYZ definida (Cinemática Inversa). En el modo automático, el usuario debe situar una pieza sobre una pantalla táctil y, el robot, tras recibir las coordenadas, calcula la trayectoria, recoge y transporta adecuadamente la pieza.

De esta forma, resulta necesario desarrollar 2 programas distintos. El primer programa está embebido en el microcontrolador del robot y se ha programado en código de Arduíno. El segundo programa es el que implementa el panel de mando y para ello se ha utilizado el software Processing, que utiliza lenguaje Java.

Otro aspecto general que se debe remarcar es el tema relacionado a las comunicaciones. Concretamente se han utilizado dos puertos de comunicación serie RS-232 con envío de información en una sola dirección. El primer puerto se utiliza para la comunicación desde el Panel de Mando (que se ejecuta en el PC) hacia el Arduíno y se utiliza para enviar todas las órdenes de movimiento, incluidas las emergencias. El segundo puerto se utiliza para enviar información sobre el estado en el que se encuentra el robot. De esta forma, utilizando el puerto serie principal de Arduíno MEGA, se envía información a un monitor serie que permite conocer en todo momento el estado del Robot, la orden que está ejecutando y si hay emergencias o no.

7.2 Programa del Panel de Mando en Processing

Processing es un entorno de programación creativo y de código abierto que se utiliza ampliamente en el ámbito del arte digital, diseño interactivo y la creación de visualizaciones y animaciones. Diseñado específicamente para artistas y diseñadores, Processing simplifica la tarea de crear aplicaciones y proyectos multimedia, permitiendo a los usuarios concentrarse en la expresión visual y la interactividad.

Una de las características más destacadas de Processing es su capacidad para crear interfaces de usuario personalizadas de manera sencilla y efectiva. Aquí es donde entra en juego la librería ControlP5. Esta es una biblioteca de procesamiento que facilita la creación de interfaces gráficas de usuario (GUI) de forma rápida y eficiente. Permite a los programadores crear paneles de mando, controles interactivos y elementos de interfaz intuitivos para sus proyectos.

La librería ControlP5 proporciona una amplia gama de elementos de interfaz, como botones, cajas de texto, barras de desplazamiento y controles deslizantes, entre otros. Estos elementos se pueden personalizar y adaptar a las necesidades estéticas y funcionales del proyecto. Además, ControlP5 ofrece una gestión sencilla de eventos, lo que permite a los usuarios capturar y responder a las interacciones del usuario, como clics de botón o cambios de valor en los controles. La programación y el uso de todos estos elementos es muy sencilla y se puede apreciar en el *programa del panel de mando en processing del ANEXO I.*

Con el fin de proporcionar una referencia visual para las explicaciones siguientes, se presenta en la *figura 52* una imagen completa del diseño final del Panel de Mando. Esta imagen incorpora los modos de marcha, los controles de mando y seguridad, así como la información detallada sobre la posición del robot y la posibilidad de modificar las velocidades y aceleraciones de los movimientos del robot. Estos aspectos serán descritos en detalle en las siguientes secciones.

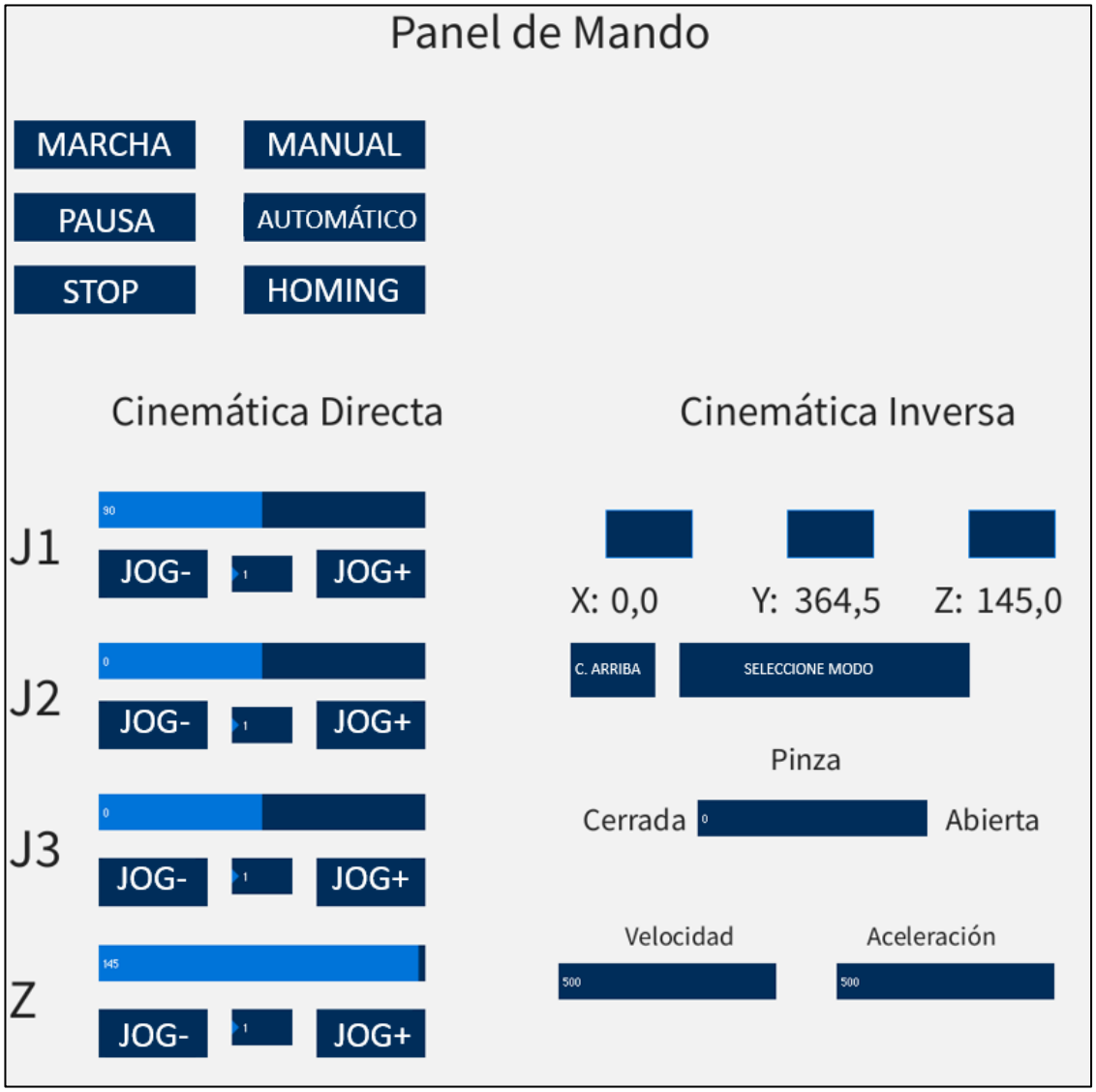


Figura 52. Panel de Mando del Robot

7.3 Modos de Marcha

Son tres los modos de marcha que se pueden seleccionar: MANUAL, AUTOMÁTICO y HOMING.

7.3.1 Modo MANUAL

Con el modo manual activado, es posible mover el robot con los dos modos de marcha que se presentan a continuación y que se corresponden con la cinemática directa y la inversa.

Sliders + JOG:

Permite introducir el valor angular de las articulaciones o la pinza mediante los sliders o los botones de JOG. Se debe tener en cuenta que el robot no se moverá hasta que no se pulse el botón multipropósito, que tendrá la etiqueta de "MOVER A POSICIÓN". Mediante la cinemática directa, el programa recalcula y muestra automáticamente la posición XYZ del efector final.

Posición XYZ:

Permite introducir un punto deseado mediante coordenadas XYZ y, a través de la cinemática inversa, el programa calcula los ángulos de las articulaciones necesarios para conseguir el punto introducido, actualizando automáticamente los valores en los sliders.

El movimiento no se ejecuta hasta que no se pulse el botón multipropósito, que tendrá la etiqueta de "MOVER A POSICIÓN". Además, se dispone de un botón para seleccionar las posiciones codo arriba y codo abajo.

7.3.2 Modo AUTOMÁTICO

El modo automático permite que el robot interactúe con la pantalla táctil. De esta forma, tras situar la bola en la pantalla, el usuario debe pulsar el botón multipropósito que tendrá la etiqueta de "BOLA SITUADA" y el robot calculará automáticamente la trayectoria necesaria para recogerla y transportarla hasta el punto objetivo. Una vez depositada la bola, el robot volverá a la posición inicial y, si se desea, se puede repetir el ciclo situando la bola en una nueva posición.

7.3.3 Modo HOMING

Esta función permite realizar el proceso de homing de todas las articulaciones en el momento que se desee. Por razones de seguridad, se ha decidido que el robot no ejecute el ciclo de homing al inicio del programa y, por ese motivo, queda como obligación del usuario ejecutarlo manualmente. También es posible realizarlo en cualquier momento, por ejemplo, tras haber movido alguna de las articulaciones del robot de forma manual mediante un movimiento humano.

Dada la naturaleza de los motores paso a paso que movilizan las articulaciones, se debe tener en cuenta que, si el robot es movido mediante fuerza humana o si se apaga el microcontrolador, se pierde completamente la posición exacta del robot y por eso resulta necesario realizar un ciclo de homing con el objetivo de situar de nuevo los puntos de referencia y poder continuar con el funcionamiento normal.

7.4 Botones de Mando y Seguridad

Resulta de vital importancia configurar de forma adecuada los botones de mando y seguridad, puesto que de ellos va a depender que el programa de control del robot sea robusto y responda adecuadamente ante todas las emergencias, se presenten en el momento que se presenten. De esta forma, se anima al lector a revisar la configuración y jerarquía de estos botones en el *programa del panel de mando en processing del ANEXO I, líneas 423-561.*

7.4.1 MARCHA / PARO

Este es el botón principal que permite que el sistema funcione o no funcione. Actúa sobre todos los modos de marcha, es decir, si se pulsa paro, se desconecta cualquier modo de marcha que hubiera seleccionado y se envía un comando de parada al robot que detiene cualquier movimiento de forma instantánea.

Dado que no existe una comunicación bidireccional entre el Robot y el Panel de Mando, si se pulsa PARO mientras el Robot está en movimiento, los valores de los ángulos que aparezcan en el panel de mando no coincidirán con los valores reales del robot. Sin embargo, dado que las paradas de emergencia se han programado de forma adecuada y que por tanto en ningún momento se pierde ningún paso en los motores, el Robot sabe constantemente en qué posición exacta se encuentra cada articulación de forma que, si se quiere reanudar el movimiento, el robot responderá adecuadamente a cualquier comando que se le envíe.

Por último, hay que añadir que todos los parámetros del panel pueden ser modificados sin activar MARCHA, pero nada funcionará hasta que no se active.

7.4.2 PAUSA / CONTINÚA

Este botón permite pausar cualquier movimiento del robot que se esté ejecutando en los modos MANUAL y HOMING, pero no estará disponible en el modo AUTOMÁTICO.

Esto se debe a que el modo AUTOMÁTICO posee una trayectoria de movimientos muy extensa y programar de forma adecuada la continuación tras pausa en todas las situaciones posibles resulta bastante complicado. Por este motivo, en el modo AUTOMÁTICO únicamente se podrá utilizar el botón de STOP / REARME.

Al pulsar este botón no se desconecta el modo de marcha en el que se encuentra el sistema. El movimiento se reanuda tras volver a pulsar el botón, que esta vez contendrá la etiqueta CONTINÚA.

7.4.3 STOP / REARME

Este botón está disponible para los modos MANUAL y AUTOMÁTICO y detiene por completo el movimiento que se esté realizando. Desconecta completamente el modo de marcha en el que se

encuentre el sistema. Tras pulsar REARME, el robot vuelve a la posición inicial y el programa estará dispuesto para volver a entrar a cualquier modo de funcionamiento que se desee.

Se ha decidido eliminarlo del modo HOMING puesto que al pulsar REARME, el robot vuelve a la posición inicial, pero, si se detiene el proceso de HOMING, el robot no tiene las referencias adecuadas y no podría volver a la posición inicial, pudiendo incluso causar la rotura de alguno de sus componentes.

7.4.4 BOTÓN MULTIPROPÓSITO

Este botón se utiliza para "ejecutar" cualquier modo y básicamente está programado para que, al pulsarlo, se envíe el vector de datos a Arduíno. Según el modo de marcha seleccionado, el botón tendrá las siguientes apariencias y funciones:

- **MANUAL**: Tendrá la etiqueta de "Ejecutar Movimiento" y, al pulsarlo, el robot se moverá hasta la posición deseada.
- **AUTOMÁTCIO**: Tendrá la etiqueta de "Bola Situada" y, al pulsarlo, el robot iniciará la lectura de la pantalla táctil y su posterior movimiento en modo automático.
- **HOMING**: Tendrá la etiqueta de "Iniciar Homing" y, al pulsarlo, el robot iniciará el ciclo completo de homing para todos sus ejes.
- **NINGUNO**: Tendrá la etiqueta de "Seleccione Modo" y, dado que no hay ningún modo seleccionado, el programa no enviará ningún dato.

7.5 Otros aspectos del Programa del Panel de Mando

7.5.1 Envío de Datos

Con el objetivo de simplificar el envío y la posterior recepción de datos, todos los valores de los ángulos de las articulaciones que se envían del Panel de Mando al Arduíno son valores enteros, es decir, ninguno es decimal. Esto implica que, en el modo MANUAL, no se pueden alcanzar las posiciones que requieran de valores decimales. De esta forma, el programa sí que trabaja con números reales, pero a la hora de enviarlos, los redondea a números enteros.

Sin embargo, el modo AUTOMÁTICO sí que permite trabajar con números reales. Esto se debe a que los cálculos de la cinemática inversa que se utilizan para calcular la posición en la que cae la bola se ejecutan en el propio Arduíno y, dado que no tienen que ser enviados, no resultan un problema.

La estructura del vector de datos se puede observar en la *figura 53*. De esta forma, se envían 8 datos separados por comas y en formato string. El primero corresponde al estado del sistema, es decir, el modo de marcha. Los cuatro siguientes corresponden a los valores de las 4 articulaciones. El siguiente es el porcentaje de apertura de la pinza y los dos últimos son la velocidad y la aceleración que se desea para el movimiento.

Se consideran como emergencia las siguientes acciones:

- Que se pulse el botón de PARO.
- Que se pulse el botón de PAUSA.
- Que se pulse el botón de STOP.
- Que durante el movimiento de robot en modo MANUAL (mientras se está ejecutando el movimiento), se envíe un movimiento a una posición distinta.

En todos estos casos, el robot se detiene completamente y, una vez parado, responde según la emergencia de la que se trate. En apartados posteriores se entrará en detalle de cómo se detectan las emergencias y cómo se ejecutan las paradas de forma segura.

Figura 53. Estructura del vector de datos

7.5.2 Control del Rango Máximo de Movimiento

Es muy importante tener en cuenta que el robot nunca puede recibir una orden de movimiento que supere su propio rango físico disponible. Para evitar esto, se ha utilizado una característica propia de los elementos interactivos "sliders", los cuales permiten delimitar el rango de valores que se puede introducir. De esta forma, es imposible que el usuario introduzca un punto que el robot no pueda alcanzar, quedando así satisfecha la condición de seguridad del movimiento.

7.6 Programa del Robot en Arduíno

El programa principal se encuentra en la función ***void loop()*** y básicamente lo que hace es esperar a que haya datos por el puerto serie y recibirlos. Si es una emergencia, para el robot, si no lo es, ejecuta la orden de la que se trate (véanse las líneas 142-179 del código del Robot en Arduíno del ANEXO I).

A continuación, se van a comentar algunas de las partes del código que por su importancia, dificultad u originalidad merecen ser remarcadas.

7.6.1 Control de los Motores Paso a Paso y Librería AccelStepper

La librería AccelStepper es una poderosa herramienta que facilita el control y la gestión de motores paso a paso en proyectos electrónicos. Con un conjunto de funciones versátiles, esta librería proporciona una interfaz sencilla para controlar simultáneamente múltiples motores paso a paso de forma precisa y eficiente.

Entre las funciones principales que ofrece la librería AccelStepper, se encuentran:

1. Control de velocidad y aceleración: La librería permite establecer la velocidad deseada para el motor, así como la aceleración y desaceleración suave. Esto garantiza un movimiento fluido y controlado, evitando oscilaciones indeseadas.

2. Control de pasos y dirección: AccelStepper permite definir la cantidad de pasos que debe dar el motor en una dirección específica. Además, también es posible cambiar la dirección del motor en cualquier momento durante el movimiento.

3. Posicionamiento preciso: Con la librería, es posible establecer la posición exacta a la que se desea que el motor se desplace. Esto resulta especialmente útil en aplicaciones donde se requiere un posicionamiento preciso, como en brazos robóticos o máquinas CNC.

4. Funcionalidad de aceleración y desaceleración suave: AccelStepper incorpora algoritmos avanzados de aceleración y desaceleración suave, lo que reduce el estrés mecánico y mejora la precisión del motor. Esto resulta fundamental para evitar vibraciones y garantizar un movimiento suave y controlado.

5. Personalización de perfiles de movimiento: La librería ofrece la flexibilidad de personalizar los perfiles de movimiento según las necesidades específicas del proyecto. Es posible ajustar la aceleración, velocidad máxima, desaceleración y otros parámetros para lograr un rendimiento óptimo del motor.

Gracias a todas estas funciones, la librería AccelStepper se ha convertido en una herramienta ampliamente utilizada en proyectos que requieren el control preciso de motores paso a paso. Su facilidad de uso y su capacidad de personalización la hacen ideal para una amplia gama de aplicaciones, desde proyectos de robótica y automatización hasta máquinas CNC y equipos de posicionamiento.

Son muchas las funciones que incorpora esta librería y tan solo se van a exponer algunas de ellas, puesto que todas las que se han utilizado pueden observarse en el ANEXO I, a lo largo de todo el código del Robot en Arduíno.

7.6.2 Movimiento de los Motores Paso a Paso

Para exponer un ejemplo representativo, se va a utilizar el movimiento en modo manual (véase la función **void modoManual()** del código del Robot en Arduíno, ANEXO I).

En esta función se asignan los valores de velocidad y aceleración a los motores, se calculan los pulsos que cada uno debe dar y se ejecuta el movimiento.

La librería ofrece distintas formas de ejecutar estos movimientos, pero, en este proyecto, se ha decidido utilizar la función **run()** puesto que no bloquea el microcontrolador. De esta forma, aunque los motores estén en movimiento, el programa puede comprobar en todo momento si llega un comando de parada de emergencia. En la figura 54 se puede observar el bucle de movimiento.

```
// Ejecuta el movimiento de las 4 Articulaciones
MZ.enableOutputs();
while (MZ.currentPosition() != posJZ || M1.currentPosition() != posJ1 || M2.currentPosition() != posJ2 || M3.currentPosition() != posJ3) {
  MZ.run();    M1.run();
  M2.run();    M3.run();

  // Si recibe datos, comprueba emergencias
  if (Serial1.available()) {
    emergencia();
    break;
  }
}
```

Figura 54. Movimiento de los 4 motores con comprobación de parada de emergencia

7.6.3 Detección de Emergencias y Parada del Robot

La acción de detectar una emergencia y parar el robot instantáneamente ha sido una de las funcionalidades que más ha costado de conseguir.

En primer lugar, se debe tener muy claro que las emergencias pueden darse mientras el robot está en funcionamiento, es decir, durante el movimiento, sea cual sea el modo en el que se encuentre. Esto implica que el programa debe ser capaz de mover el robot ejecutando el funcionamiento normal, pero, además, debe estar comprobando en todo momento si se recibe alguna emergencia.

La primera estrategia que se probó fue implementar esa rutina de estar atento en todo momento a la recepción de datos que puedan suponer una emergencia. De esta forma, el programa, si detectaba que había datos en el buffer, pasaba a recibirlos todos (los 8) y actuaba en función de los comandos recibidos.

Este proceso tomaba mucho tiempo al microcontrolador y lo que producía era una parada muy brusca, que correspondía al momento en el que el micro dejaba de atender el movimiento de los motores y pasaba a recibir los datos. Tras recibirlos, el movimiento continuaba con una aceleración muy notoria y posteriormente atendía la emergencia y paraba los motores. Este funcionamiento es completamente inadecuado puesto que la parada del robot se producía con movimientos muy erráticos.

Dado que el único dato que es necesario leer con urgencia es el primero, puesto que indica si se trata de una emergencia o no, la segunda estrategia que se probó consistió en recibir únicamente el primer dato, comprobar si se trataba de una emergencia, actuar en consecuencia y posteriormente recibir el resto. Lamentablemente, este método tampoco funcionó y aunque sí que disminuyó el tiempo de reacción, la parada no se ejecutaba adecuadamente.

Tras analizar detenidamente el problema, se llegó a la conclusión de que, si el robot recibía datos durante la ejecución de cualquier modo de funcionamiento, se podía considerar como una parada de emergencia sin tener ninguna necesidad de invertir tiempo en leerlos.

De esta forma, el robot ejecuta directamente la parada y posteriormente recibe los datos y actúa en consecuencia, quedando conseguido el objetivo propuesto.

Habida cuenta de todo esto, se puede observar a lo largo de todo el código que tanto antes de empezar movimientos *(figura 55*) como durante la ejecución de estos (*figura 54*), existen multitud de llamadas a la función de comprobar si hay datos disponibles, de forma que, si los hay, el código sale adecuadamente de la parte donde se encuentre y llega rápidamente a la función de detener motores.

```
// Secuencia de movimientos del modo AUTOMÁTICO
if (Serial1.available()){emergencia(); return;} else {autoP1();}   // P1: ENCIMA BOLA
if (Serial1.available()){emergencia(); return;} else {autoP2();}   // P2: BAJA BOLA
if (Serial1.available()){emergencia(); return;} else {autoP3();}   // P3: COGE BOLA
if (Serial1.available()){emergencia(); return;} else {autoP3_2();} // P3_2: SUBE CON PIEZA
if (Serial1.available()){emergencia(); return;} else {autoP4();}   // P4: CENTRAL
if (Serial1.available()){emergencia(); return;} else {autoP5();}   // P5: ENCIMA OBJETIVO
if (Serial1.available()){emergencia(); return;} else {autoP6();}   // P6: BAJA OBJETIVO
if (Serial1.available()){emergencia(); return;} else {autoP7();}   // P7: SUELTA BOLA
if (Serial1.available()){emergencia(); return;} else {autoP8();}   // P8: CENTRAL
```

Figura 55. Ejemplo de comprobación de emergencia antes de iniciar cualquier movimiento del modo AUTOMÁTICO

El código encargado de detener los motores se puede apreciar en la *figura 56*. Básicamente lo que se hace es asignar a los motores una aceleración considerablemente alta (con el objetivo de que se detengan rápido), llamar a la función **MX.stop(),** que se encarga de calcular los pulsos que debe dar el motor para detenerse con seguridad teniendo en cuenta la aceleración anterior, y posteriormente se entra en el bucle donde se desaceleran los motores. Una vez detenidos, se reasigna la aceleración previa para cuando se reanude el movimiento. También se puede observar que el envío del mensaje de EMERGENCIA se realiza una vez que los motores están parados. Esto es por que el envío de datos por el puerto serie es relativamente lento y lo que se prioriza es la parada de los motores.

```
// Función que se ejecuta al recibir una EMERGENCIA. Detiene rápidamente los motores
void emergencia() {

  // Se detienen rápidamente todos los motores
  MZ.enableOutputs();
  // Se asignan aceleraciones máximas para detener el robot rápidamente
  MZ.setAcceleration(2000); M1.setAcceleration(2000); M2.setAcceleration(2000); M3.setAcceleration(2000);

  // Calcula las nuevas posiciones de parada y detiene motores
  MZ.stop();   M1.stop();
  M2.stop();   M3.stop();

  while (MZ.isRunning() || M1.isRunning() || M2.isRunning() || M3.isRunning()) {
    MZ.run();    M1.run();
    M2.run();    M3.run();
  }

  MZ.disableOutputs();

  // Se asignan los valores de aceleración previos
  MZ.setAcceleration(data[aceleracion]); M1.setAcceleration(data[aceleracion]);
  M2.setAcceleration(data[aceleracion]); M3.setAcceleration(data[aceleracion]);

  // Muestra INFO por pantalla
  Serial.println("EMERGENCIA: Motores detenidos");
}
```

Figura 56. Función de Parada de Emergencia

7.6.4 Configuración CNC-Shield

La configuración de la CNC-Shield generó bastantes problemas al inicio. Se debe tener en cuenta que la CNC-Shield incorpora un pin para habilitar todos los drivers de los motores. Además, este pin está invertido, es decir, se habilita el driver con un 0 lógico. Esto se debe configurar adecuadamente en la función **void setup()** (véase figura 57).

```
// Configuración de la CNC-shield
M1.setEnablePin(enDrivers);  M1.setPinsInverted(false, false, true);
M2.setEnablePin(enDrivers);  M2.setPinsInverted(false, false, true);
M3.setEnablePin(enDrivers);  M3.setPinsInverted(false, false, true);
MZ.setEnablePin(enDrivers);  MZ.setPinsInverted(false, false, true);
```

Figura 57. Configuración CNC-Shield

7.6.5 Recepción y Procesamiento de Datos

Como ya se ha comentado, el flujo de datos principal va desde el Panel de Mando del PC hacia el Arduíno del Robot. Lo que se envía y recibe es una cadena de caracteres que contiene los datos como números enteros separados por comas. De esta forma, el programa implementado detecta donde están las comas y transforma los números de formato caracter a formato entero (véase *figura 58*).

```
// Función que recibe los datos y los muestra por palntalla.
void recibeDatos() {
  content = Serial1.readString();                    // Lee los datos y los almacena en el estrig content
  for (int i = 0; i < 8; i++) {
    int index = content.indexOf(",");                // Busca el primer caracter  ","
    data[i] = atol(content.substring(0, index).c_str());  //Extrae el  primer número y lo convierte a entero
    content = content.substring(index + 1);          //Elimina el primer número y la primera coma para repetir el proceso
  }
```

Figura 58. Recepción y Procesamiento de Datos

7.6.6 Implementación del Ciclo de HOMING

El Robot, tras recibir la orden de ejecutar el ciclo de homing, moviliza los motores de uno en uno hasta detectar el final de carrera. Una vez detectado, se mueve en sentido contrario a una velocidad muy baja hasta que deja de detectar el final de carrera. De esta forma, el robot dispone de la referencia adecuada y mueve cada articulación hasta la posición inicial (véase la función **void homing()** del programa del Robot en Arduíno).

8 Resultados Reales

Una vez desarrollado todo el código, se puede pasar a probar todos los modos de funcionamiento y comprobar que todos ellos funcionen adecuadamente.

Dado que en el funcionamiento real el Robot se mueve de forma continua y lo más adecuado para mostrar su funcionamiento sería un archivo de vídeo, se va a intentar reflejar dicho funcionamiento mediante imágenes. De esta forma, en las siguientes figuras se puede observar al robot realizando la trayectoria del modo automático.

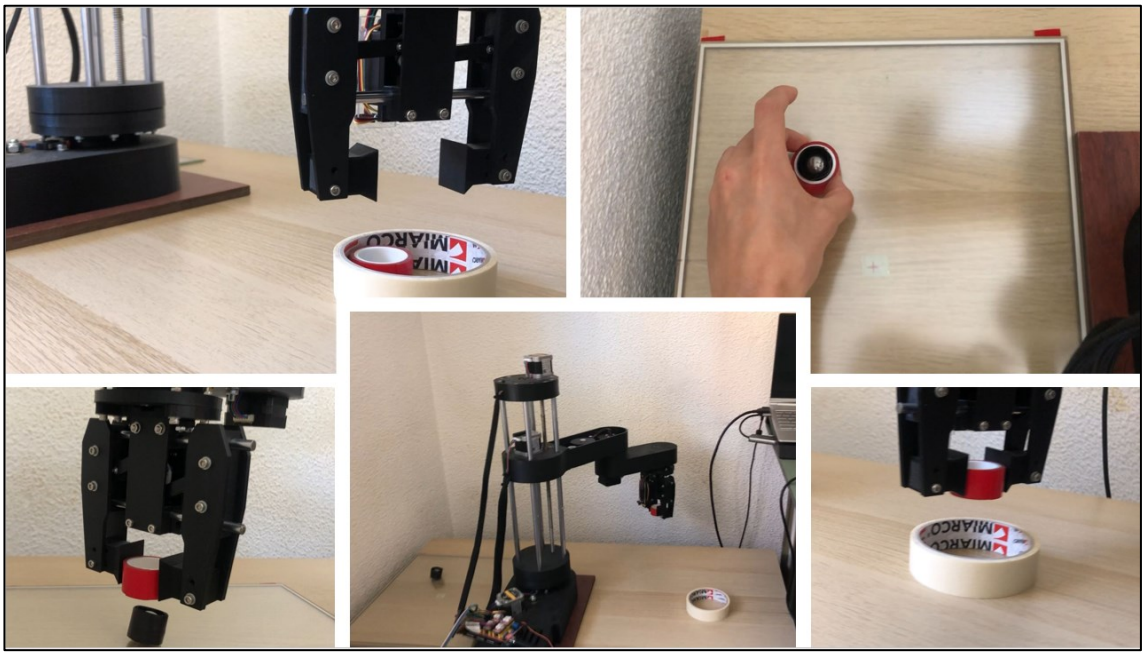


Figura 59. Funcionamiento Real en Modo AUTOMÁTICO

9 Propuestas de Mejora

Se proponen las siguientes mejoras para el proyecto:

1. Posibilidad de enviar números decimales o información en pulsos en lugar de grados: Se sugiere la incorporación de esta funcionalidad para evitar la pérdida de información y permitir el control más preciso del robot en el modo manual. Esto permitiría alcanzar todas las posiciones deseadas de forma más precisa y flexible. También podría solucionarse de forma simple enviando los valores en pulsos de motor en lugar de en grados o milímetros.

2. Comunicación bidireccional entre Arduino y Processing: Se propone establecer una comunicación bidireccional entre ambos sistemas, de modo que el robot pueda enviar comandos de confirmación (por ejemplo, un comando de OK) a Processing después de cada movimiento. Esta funcionalidad facilitaría la sincronización y el control de movimiento, así como la visualización de información en el panel, como los valores reales de posición durante las paradas de emergencia.

3. Implementación del comando PAUSA/CONTINÚA en el modo AUTOMÁTICO: Se sugiere incluir la opción de pausar y reanudar el modo automático para permitir una mayor flexibilidad en el control del robot. Esta funcionalidad sería útil para detener temporalmente el proceso automático en caso de necesidad y luego continuar desde el punto donde se dejó.

4. Implementación de movimientos más complejos, como trayectorias trapezoidales en el efector final: Se propone añadir la capacidad de realizar movimientos más complejos, como trayectorias trapezoidales, en el efector final del robot. Esto permitiría realizar movimientos suaves y precisos, mejorando la eficiencia y la calidad de las tareas realizadas por el robot.

5. Incorporar encoders absolutos o cambiar a servomotores con encoder: Se sugiere esta mejora para no tener que realizar el ciclo de homing al iniciar el programa.

Estas mejoras propuestas ampliarían la funcionalidad y el rendimiento del sistema, brindando mayor precisión, flexibilidad y control en las operaciones del robot.

10 Conclusiones

Tras finalizar el Trabajo de Fin de Máster y reflexionar sobre el camino recorrido, se puede afirmar que se han logrado la mayoría de las tareas y objetivos planteados al inicio. Por un lado, se ha puesto en práctica los conocimientos adquiridos en control automático y programación a lo largo del último año, ampliándolos significativamente y adaptando los conceptos al control de motores paso a paso. Esto ha permitido adquirir experiencia en el diseño de diversos sistemas de control y comprender mejor su funcionamiento y la interacción entre sus componentes. La combinación de esto con el diseño de productos utilizando herramientas CAD/CAM ha sido especialmente gratificante, ya que ha contribuido a la concepción de entornos y modelos de simulación. Aunque en ocasiones se encontraron desafíos, en general, el proyecto resultó ser interesante y estimulante.

Para el objetivo principal del Brazo Robótico, se pudo llevar a cabo un estudio exhaustivo del diseño del prototipo de robot SCARA, aprovechando los conocimientos mecánicos adquiridos a lo largo del máster. Se logró resolver tanto los problemas cinemáticos directos como inversos mediante métodos geométricos básicos. Dado que se trata de un tipo de robot ampliamente conocido, comprender y abordar la solución de estos problemas no supuso ninguna dificultad.

Asimismo, se logró desarrollar un modelo de simulación que permitió validar los cálculos realizados y verificar el correcto funcionamiento del diseño mecánico. Como se explicó en la memoria, este modelo simula de manera realista el comportamiento que se espera del robot en su implementación práctica.

Además, se logró una exitosa implementación física del robot, acompañada de una interfaz de usuario que facilita la comunicación entre el usuario y el microcontrolador. Se realizó una exhaustiva comprobación experimental que confirmó el correcto funcionamiento tanto del robot como de la interfaz, respaldando así las simulaciones realizadas con el modelo creado.

No obstante, el proceso de montaje ha demostrado ser el más desafiante, tanto por las circunstancias actuales mencionadas anteriormente como por la transición de la teoría a la práctica, del mundo simulado al mundo real. Durante esta fase, surgieron algunas incidencias no previstas, como la dificultad para retirar los soportes de agujeros o taladros en las piezas impresas en 3D, así como la necesidad de reemplazar algunos componentes por otros que ofrecieran mejores resultados en las condiciones de uso previstas, como las tuercas autoblocantes. Esto ha brindado una comprensión más profunda de la multitud de aspectos a tener en cuenta al fabricar un prototipo desde cero.

Por lo tanto, se considera que se han cumplido los objetivos establecidos para esta parte del proyecto.

Además, durante el desarrollo del proyecto, se ha comprendido la importancia de la metodología del Mínimo Producto Viable (MVP), que consiste en partir de un producto básico y a partir de él, ir realizando mejoras y refinamientos. Esta metodología se considera aplicable tanto en el ámbito laboral como en otros aspectos de la vida, donde es necesario cumplir con requisitos básicos y luego buscar la mejora continua. Además, se destaca otra metodología de trabajo implementada a lo largo de todo el proyecto, que consiste en descomponer una tarea compleja en varias tareas más simples, lo cual facilita su realización y permite evaluar el progreso realizado.

11 Presupuesto

Para realizar el cálculo del presupuesto se ha tenido en cuenta que el proyecto ha sido desarrollado exclusivamente por un único ingeniero, quien es el autor de esta memoria (sin considerar la colaboración del tutor universitario, que ha participado en diferentes aspectos en mayor o menor medida). Con el objetivo de proporcionar una mayor transparencia y facilitar la comprensión, el presupuesto se divide en diferentes categorías según la naturaleza de cada elemento.

Es importante destacar que en la elaboración del presupuesto no se ha considerado el costo de fabricación de todas las piezas impresas, ya que se utilizó una impresora propia. Tampoco se ha tenido en cuenta el costo de las licencias para el software matemático Matlab y el software de diseño SolidWorks, debido a que se utilizaron con la licencia de estudiante de la UPV.

11.1 Coste Material

En primer lugar, se presenta en detalle el presupuesto correspondiente a los materiales necesarios para la ejecución del proyecto. Con el fin de facilitar la identificación de los componentes, se ha dividido el presupuesto en diferentes categorías que corresponden a los subensamblajes que conforman el robot. Los presupuestos de los conjuntos se muestran en *las Tablas 9, 10, 11, 12 y 13* respectivamente, para el conjunto base, el conjunto de la torre de elevación, el conjunto del brazo 1, el conjunto del brazo 2 y el conjunto de la pinza.

Conjunto	Material	Unidades	Precio (€)
Base	Correa GT2 de 200 mm	1	1.65
	Correa GT2 de 300 mm	1	1.65
	Rodamiento axial - 40x60x13 mm	2	7.8
	Rodamiento radial - 35x47x7 mm	2	10
	Rodamiento radial 608 - 8x22x7 mm	2	1.125
	Tornillo M8 45 mm	1	0.10
	Arandela M8	1	0.02
	Tuerca M8 autoblocante	1	0.04
	Tornillo M4 55 mm	4	0.20
	Tuerca M4 autoblocante	4	0.06
	Motor Nema 17HD48002H-22B	1	5
	Polea de 5 mm de 20 dientes	1	1.21
	Final de carrera	1	0.7
Total conjunto base			29.528 €

Tabla 9. Presupuesto de la Base del Robot

Conjunto	Material	Unidades	Precio (€)
Elevación	Rodamiento radial 608 - 8x22x7 mm	1	0.5625
	Barra calibrada D10 mm L400 mm	4	12
	Barra roscada D8 mm L380 mm	1	10
	Acoplamiento de 5mm a 8mm	1	5
	Rodamiento radial 10x19x29 mm	4	8
	Final de carrera	1	0.7
	Motor Nema 17HD48002H-22B	1	5
	Polea de 5 mm de 20 dientes	1	1.21
Total conjunto elevación			42.4725 €

Tabla 10. Presupuesto de la torre de Elevación

Conjunto	Material	Unidades	Precio (€)
Brazo 1	Correa GT2 de 300 mm	1	1.65
	Correa GT2 de 400 mm	1	1.65
	Rodamiento axial - 35x52x12 mm	2	8
	Rodamiento radial - 30x42x7 mm	1	3.5
	Rodamiento radial 608 - 8x22x7 mm	2	1.125
	Tornillo M8 45 mm	1	0.10
	Arandela M8	1	0.02
	Tuerca M8 autoblocante	1	0.04
	Tornillo M4 50 mm	4	0.20
	Tuerca M4 autoblocante	4	0.06
	Polea de 5 mm	4	4.88
	Polea de 5 mm de 20 dientes	1	1.21
	Motor Nema 17HD48002H-22B	1	5
	Final de carrera	1	0.7
Total conjunto brazo 1			28.135 €

Tabla 11. Presupuesto del Brazo 1

Conjunto	Material	Unidades	Precio (€)
Brazo 2	Correa GT2 de 400 mm	1	1.65
	Rodamiento axial - 35x52x12 mm	2	8
	Rodamiento radial - 30x42x7 mm	1	3.5
	Tornillo M4 55 mm	4	0.20
	Tuerca M4 autoblocante	4	0.06
	Polea de 5 mm	2	2.44
	Polea de 5 mm de 20 dientes	1	1.21
	Motor Nema 17Hs2408S 17	1	12.49
	Final de carrera	1	0.7
Total conjunto brazo 2			30.25 €

Tabla 12. Presupuesto del Brazo 2

Conjunto	Material	Unidades	Precio (€)
Pinza	Servomotor MG996R	1	5
	Tornillería	4	3.26
Total pinza			8.26 €

Tabla 13. Presupuesto de la Pinza

El presupuesto total de los materiales del proyecto se presenta en la *Tabla 14*.

Conjunto	Precio (€)
Base	29.528 €
Elevación	42.4725 €
Brazo 1	28.135 €
Brazo 2	30.25 €
Pinza	8.26 €
Electrónica	83.1 €
Total	221.7455 €

Tabla 14. Presupuesto total del coste de material

11.2 Coste Personal

A continuación, se presenta en la *Tabla 15* los honorarios establecidos por el autor del presente proyecto, según las horas dedicadas y el tipo de tarea realizada. Es importante destacar que las horas indicadas en dicha tabla no representan el número real de horas que el autor ha invertido en la ejecución del proyecto. Si se reflejaran las horas reales para mantener un precio asequible, sería necesario establecer un precio por hora significativamente bajo.

Dedicación	Precio por hora (€/h)	Horas de trabajo (h)	
Creación de modelo	20	40	800 €
Implementación real	15	70	1050 €
Documentación	10	40	400 €

Tabla 15. Coste Personal

11.3 Coste Total

Por último, en la *Tabla 16* se muestra el presupuesto final del presente proyecto, que incluye el costo total de los materiales utilizados, así como la remuneración que el autor del proyecto debe recibir por sus labores realizadas.

Concepto	Precio (€)
Materiales	221.75
Coste personal	2250
Total	2471.75

Tabla 16. Presupuesto total

12 Pliego de Condiciones

12.1 Definición y Alcance del Pliego

El presente documento establece las condiciones técnicas mínimas para el desarrollo e implementación de un brazo robótico SCARA de 6 grados de libertad, destinado a la ejecución de tareas de Pick & Place. Se enfoca en las fases de modelado, programación, montaje y pruebas de funcionamiento real, considerando que este proyecto tiene un carácter académico y educativo, sin contemplar su implementación en una línea de producción real.

12.2 Condiciones Generales

Es obligatorio cumplir con todas las normas y requisitos comúnmente observados en proyectos similares. Para salvaguardar la seguridad del proyectista y de aquellos que realicen modificaciones o pongan en marcha el proyecto en el futuro, se establece la obligatoriedad de cumplir con el Reglamento Electrotécnico de Baja Tensión para los componentes electrónicos o eléctricos. Este reglamento proporciona el marco normativo para trabajar con suministro eléctrico de menos de 1000 V CA y 1500 V CC.

12.3 Condiciones Particulares

12.3.1 Condiciones Facultativas

El contratista (autor del proyecto) asume una serie de derechos y obligaciones:

- Conocer la normativa aplicable.
- Familiarizarse con todos los aspectos del proyecto.
- Contar con la presencia o localización de los responsables durante las diferentes fases del proyecto.
- Elaborar un documento que refleje las indicaciones, aclaraciones o modificaciones del proyecto.
- Disponer de los medios auxiliares necesarios para garantizar el desarrollo correcto del proyecto, adquiriendo el material personalmente (costo del material a cargo del contratista).
- Recibir los pagos comprometidos en las fechas acordadas, si los hubiera.
- Percibir una compensación económica por los trabajos no especificados en los documentos del proyecto, pero necesarios para su correcta ejecución.

El Tutor de la Universidad, en su rol de dirección facultativa, tiene las siguientes obligaciones y facultades:

- Supervisar aspectos del proyecto que afecten a su fiabilidad, calidad y seguridad durante la ejecución.
- Estar presente en momentos clave del desarrollo del proyecto.
- Contribuir con soluciones técnicas a problemas imprevistos durante la ejecución.

- Realizar las ampliaciones necesarias del proyecto en función de las modificaciones introducidas respecto a las soluciones iniciales.

12.3.2 Condiciones Técnicas

El uso de dispositivos electrónicos y eléctricos sujetos al Reglamento Electrotécnico de Baja Tensión debe complementarse con la información técnica proporcionada por los fabricantes. Se deben seguir las recomendaciones de uso especificadas en las hojas técnicas o datasheets de los componentes. Las piezas fabricadas mediante impresión 3D deben cumplir las normas aplicables a estas tecnologías y ser impresas con materiales apropiados bajo la supervisión de personal cualificado.

Los requisitos de software son los siguientes:

- Sistema operativo Windows 10 o superior.
- Matlab 2022b o superior.
- SolidWorks 2021 o superior.
- Arduino IDE.
- Processing.
- Ultimaker Cura 5.2 o superior.

En cuanto al hardware, se recomienda contar con las siguientes especificaciones mínimas para garantizar un desarrollo fluido de las diferentes fases del proyecto (es posible realizar las actividades requeridas sin cumplir todos estos requisitos, aunque el proceso podría no ser óptimo):

- Procesador Intel Core i5 o superior.
- Memoria RAM de 8 GB o superior.
- Almacenamiento libre mínimo de 30 GB.
- Tarjeta gráfica GeForce GTX 750 o superior.

12.3.3 Condiciones Económicas y Legales

Dado el enfoque docente del proyecto, cuyo objetivo principal es desarrollar las habilidades ingenieriles del contratista en el marco de su Trabajo de Fin de Máster, así como para aquellos que utilicen procedimientos explicados o realicen modificaciones en el futuro, no se establecen condiciones económicas o legales específicas.

13 Relación del Trabajo con los Objetivos de Desarrollo Sostenible

A continuación, se presentarán las formas en las que este proyecto contribuye a la consecución de los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles		Alto	Medio	Bajo	No Procede
ODS 1.	Fin de la pobreza.		X		
ODS 2.	Hambre cero.		X		
ODS 3.	Salud y bienestar.	X			
ODS 4.	Educación de calidad.			X	
ODS 5.	Igualdad de género.			X	
ODS 6.	Agua limpia y saneamiento.				X
ODS 7.	Energía asequible y no contaminante.			X	
ODS 8.	Trabajo decente y crecimiento económico.		X		
ODS 9.	Industria, innovación e infraestructuras.	X			
ODS 10.	Reducción de las desigualdades.			X	
ODS 11.	Ciudades y comunidades sostenibles.			X	
ODS 12.	Producción y consumo responsables.			X	
ODS 13.	Acción por el clima.	X			
ODS 14.	Vida submarina.				X
ODS 15.	Vida de ecosistemas terrestres.				X
ODS 16.	Paz, justicia e instituciones sólidas.				X
ODS 17.	Alianzas para lograr objetivos.				X

El desarrollo de este prototipo de brazo robótico tipo SCARA presenta una clara relación con varios de los Objetivos de Desarrollo Sostenible (ODS) establecidos por las Naciones Unidas. En primer lugar, el objetivo de Fin de la Pobreza se vincula con el potencial de la robótica para mejorar la eficiencia y productividad en diferentes sectores, lo que a su vez puede generar empleos decentes y promover el crecimiento económico. Al automatizar tareas repetitivas y agotadoras, como el pick and place, se libera a los trabajadores humanos de labores monótonas y se les brinda la oportunidad de desarrollar habilidades en áreas más especializadas.

En relación al objetivo de Hambre Cero, el uso de la robótica puede contribuir a una producción más eficiente y sostenible en la cadena de suministro de alimentos. La automatización de tareas agrícolas y de procesamiento puede aumentar la productividad, reducir las pérdidas de alimentos y garantizar un suministro más seguro y accesible para todos.

Además, el objetivo de Salud y Bienestar se relaciona con la aplicación de la robótica en entornos médicos, donde los robots pueden desempeñar un papel importante en la asistencia a pacientes y la realización de procedimientos complejos. Esto no solo mejora la calidad de la atención médica, sino que también reduce los riesgos y la fatiga de los profesionales de la salud.

En cuanto a Industria, Innovación e Infraestructuras, la implementación de tecnologías robóticas promueve la automatización de procesos industriales, mejorando la eficiencia, la calidad y la seguridad en la producción. Esto impulsa el crecimiento económico y fomenta la innovación en la industria.

Por último, en el ámbito de Acción por el Clima, es relevante destacar que en el diseño y dimensionamiento de los motores del brazo robótico se ha buscado la eficiencia energética, ajustando la intensidad máxima de los motores para reducir el consumo de energía al mínimo necesario. Esto contribuye a la mitigación del cambio climático y a la conservación de los recursos naturales.

Por todo esto, el uso de la robótica, como se ha evidenciado en el desarrollo de este prototipo de brazo robótico SCARA, ofrece un gran potencial para abordar los desafíos de los ODS, promoviendo la eficiencia, la sostenibilidad y el bienestar en diversos sectores de la sociedad.

14 Bibliografía

[1] Carlos Balaguer y Rafael Aracil-Santoja Antonio Barrientos. Fundamentos de Robótica, volumen 2. McGraw-Hill Interamericana de España, 2007.

[2] – “Apuntes de la asignatura control Aplicado de sistemas Mecatrónicos” – V. Casanova Calvo.

[3] – “Catálogo Técnico SureStepperManual. Motores paso a paso”

[4] BOE. Resolución de 7 de octubre de 2019, de la Dirección General de Trabajo, por la que se registra y publica el XIX Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos.

[5] – “User’s Guide SolidWorks 2021”

[6] – “Help MATLAB R2022b”

[7] – “Documentación Librería AccelStepper”

[8] – “SCARA ROBOT. How to Mechatronics, 2020”

ANEXO I. Códigos de los Programas

1 Código del Robot en Arduíno

```
/*
  Programa: Código embebido en Arduíno: Control de las articulaciones y de la Pantalla
  táctil

  Desarrollado por: Roberto Lorente Romanos
  Fecha:  5/5/2023
*/
// INCLUDES
#include <AccelStepper.h>       // Librería para controlar los motores de pasos
#include <Servo.h>              // Librería para controlar el servo
#include <SoftwareSerial.h>     // Librería para utilizar un segundo puerto Serie

// DEFINES
#define microStepping 0.25      // Jumpers colocados para realizar 1/4 de paso
#define gp            1.8       // Grados que avanza el motor por cada pulso con
microStepping 1 (paso completo)
#define avanceHusillo 2.0       // Avence del husillo por revolución 2 [mm/rev]
#define avanceJRev    360.0     // Avance directo de las articulaciones de revolución 360
[º/rev]
#define L1            228.0     // Longitud brazo 1 [mm]
#define L2            136.5     // Longitud brazo 2 [mm]

#define enDrivers 8             // Pin para habilitar los drivers

// Defines para llamar a valores concretos de cada Final de Carrera dispuestos en vectores
específicos
#define FCZ 0        // Final de carrera de la articulación J1
#define FC1 1        // Final de carrera de la articulación J2
#define FC2 2        // Final de carrera de la articulación J3
#define FC3 3        // Final de carrera de la articulación Z

// Defines para llamar a valores concretos de cada articulación dispuestos en vectores
específicos
#define JZ  0       // Articulación Z
#define J1  1       // Articulación 1
#define J2  2       // Articulación 2
#define J3  3       // Articulación 3

// Defines para llamar a los datos del vector data
#define estado          0
#define alturaJZ        1
#define anguloJ1        2
#define anguloJ2        3
#define anguloJ3        4
#define aperturaPinza   5
#define velocidad       6
#define aceleracion     7

// Pto XY donde depositar la pieza al final del modo AUTO
#define xObjetivo  348.75
#define yObjetivo  0
```

```
// Valores de pulsos que determinan los rangos máximos de movimiento
#define PmaxZ -58688          // Pulsos máximos para el rango de movimiento de JZ (Mic_step
= 1/4)
#define Pmax1 14354           // Pulsos máximos para el rango de movimiento de J1 (Mic_step
= 1/4)
#define Pmax2 10880           // Pulsos máximos para el rango de movimiento de J2 (Mic_step
= 1/4)
#define Pmax3 2594            // Pulsos máximos para el rango de movimiento de J3 (Mic_step
= 1/4)
#define P0CI_1 3795           // Pulsos en J1 que sitúan el inicio de los ejes de coord. de
la CI
//#define Pcentral_1 7851     // Pulsos hasta posición central desde FC en J1
#define Pcentral_1 7795       // Pulsos hasta posición central desde FC en J1
#define Pcentral_2 5345       // Pulsos hasta posición central desde FC en J2
#define Pcentral_3 1413       // Pulsos hasta posición central desde FC en J3

// Defines de la pantalla táctil
#define UR 47                  // UP RIGTH
#define LR 49                  // LOW RIGTH
#define UL 51                  // UP LEFT
#define LL 53                  // LOW LEFT
#define signal A15             // PIN de señal
#define offsetX  0.0          // Corrección X -0.6
#define offsetY  0.0           // Corrección X  8.0

// Declaración de los motores de pasos (Typeof driver: with 2 pins, STEP, DIR)
AccelStepper MZ(1, 12, 13); AccelStepper M1(1, 2, 5);  AccelStepper M2(1, 3,
6);  AccelStepper M3(1, 4, 7);

// Declaración del Servo
Servo servoPinza;

// Declaración del Segundo puerto Serie
//SoftwareSerial Serial1(2, 3); // RX, TX SOLO PARA RDUÍNO UNO!!!!
// ARDUINO MEGA LLEVA PUERTOS SERIES INTEGRADOS

// VARIABLES GLOBALES
// Vector que reúne los pines a los que están conectados los Finales de Carrera
int vFC[4] = { A3, 9, 10, 11 };
// Vector que almacena el estado anterior de los FC. Utilizado para filtrar los
rebotes
int estado_anterior[4] = { 0, 0, 0, 0 };
// Vector que reúne los índices de reducción de cada
articulación
int vi[4] = { 1, 20, 16, 4 };
// Vector que reúne los avances por vuelta de cada
articulación
float vdLoad[4] = { avanceHusillo, avanceJRev, avanceJRev, avanceJRev };
// Coordenadas XY de la posición de la bola. Las mide la pantalla
float Pxb, Pyb;
long posJZ, posJ1, posJ2, posJ3, porcenPinza;
// Variables que se utilizan para calcular los movimientos que deben realizar los
motores
long q1p_CI, q2p_CI;
// Variables que calcula la función CI
[pulsos]
int velAnterior = 500;
```

```
int accAnterior = 500;

// Variables para la recepción de datos
// En este vector se recibirá toda la cadena de caracteres que haya en el puerto serie
String content = "";
// Vector en el que se almacenarán los datos recibidos en formato entero
int data[8];

// Cálculos iniciales
// Variable que representa el microStepping seleccionado
float thetaStep = 360.0/(gp * microStepping);
// [mm/pul] y [º/pul] Resuloción de cada Articulación
float vReso [4] = {(avanceHusillo/vi[JZ])/thetaStep, (avanceJRev/vi[J1])/thetaStep,
(avanceJRev/vi[J2])/thetaStep, (avanceJRev/vi[J3])/thetaStep };

// Variables de mando
int ESTADO;

void setup() {

  // Establece parámetros de comunicación serie para ambos puertos
  Serial.begin(115200);
  Serial1.begin(115200);

  // Configuración de la CNC-shield
  M1.setEnablePin(enDrivers);  M1.setPinsInverted(false, false, true);
  M2.setEnablePin(enDrivers);  M2.setPinsInverted(false, false, true);
  M3.setEnablePin(enDrivers);  M3.setPinsInverted(false, false, true);
  MZ.setEnablePin(enDrivers);  MZ.setPinsInverted(false, false, true);

  // Configuración de los finales de carrera como entradas PULL UP
  pinMode(vFC[FCZ], INPUT_PULLUP);  pinMode(vFC[FC1], INPUT_PULLUP);
  pinMode(vFC[FC2], INPUT_PULLUP);  pinMode(vFC[FC3], INPUT_PULLUP);

  // Parámetros motores de pasos
  MZ.setMaxSpeed(4000);  MZ.setAcceleration(500);  MZ.setCurrentPosition(58000);
  M1.setMaxSpeed(4000);  M1.setAcceleration(500);  M1.setCurrentPosition(4000);
  M2.setMaxSpeed(4000);  M2.setAcceleration(500);  M2.setCurrentPosition(0);
  M3.setMaxSpeed(4000);  M3.setAcceleration(500);  M3.setCurrentPosition(0);

  // Parámetros Servo
  servoPinza.attach(52);

  // Configuración Pantalla Táctil
  pinMode(UR,OUTPUT) ; pinMode(LR,OUTPUT) ;
  pinMode(UL,OUTPUT) ; pinMode(LL,OUTPUT) ;
  digitalWrite(LR, LOW); digitalWrite(UL,HIGH );  // Se activan los que permanecen
constantes

  // Se limpia el buffer por precaución
  serialFlush();

  Serial.println("INICIO DEL PROGRAMA");
}

void loop() {

  // Recibe cadena de datos por el puerto serie
```

```
  if (Serial1.available()) {

    // Recibe los datos y comprueba EMERGENCIAS
    recibeDatos(); // Si  EMERGENCIA,  detiene instantáneamente el movimiento

    // Extrae la variable de MANDO
    ESTADO = data[estado];

    switch (ESTADO) {

      // EMERGENCIA
      case 0:
      // No se llama a emergencia puesto que ya se ha hecho
        break;

      // MANUAL
      case 1:
        modoManual();
        break;

      // AUTOMÁTICO
      case 2:
        modoAuto();
        break;

      // HOMING
      case 3:
        homing();
        break;

      default:
          Serial.println("Estado desconocido, seleccione un modo de funcionamiento válido en
el panel de mando");
        break;
    }
  }
} // End loop()

// Función que lee los finales de carrera e implementa un algoritmo antirrebotes
int compruebaFC(char FC) {
  int estado_actual = digitalRead(vFC[FC]);
  if (estado_actual != estado_anterior[FC]) {
    delay(5);
    estado_actual = digitalRead(vFC[FC]);
    if (estado_actual != estado_anterior[FC]) {
      estado_anterior[FC] = estado_actual;
    }
  }
  return estado_anterior[FC];
}

// Función que cierra por completo la pinza
void moverPinza(int porcenObjetivo) {

  // Variables auxiliares utilizadas en los bucles for
  int pos1, pos2;

  // Se convierten los valores de porcentaje a grados reales del servo
```

```
  int posObjetivo = round(map(porcenObjetivo, 0, 100, 82, 0));
  int posPinza = round(map(porcenPinza, 0, 100, 82, 0));

  // Si se debe cerrar la pinza
  if (posPinza < posObjetivo) {
    for (pos1 = posPinza; pos1 <= posObjetivo; pos1++) {
      servoPinza.write(pos1);
      delay(5);
    }
    porcenPinza = round(map(pos1, 82, 0, 0, 100));
    // Si se debe abrir la pinza
  } else if (posPinza > posObjetivo) {
    for (pos2 = posPinza; pos2 >= posObjetivo; pos2--) {
      servoPinza.write(pos2);
      delay(5);
    }
    porcenPinza = round(map(pos2, 82, 0, 0, 100));
    // Si la pinza ya esta en la posición, sale de la función
  } else {
    return;
  }
}

// HOMING MZ
void homingMZ() {
  MZ.enableOutputs();

  // Si no hay datos, mueve el motor hasta detectar el FC
  if (Serial1.available()) {
    emergencia();
    return;
  } else {
    while (compruebaFC(FCZ) == 0) {
      MZ.setSpeed(2500);
      MZ.runSpeed();
      if (Serial1.available()) { emergencia(); break; }
    }
  }

  // Si no hay datos, avanza despacio hasta dejar de detectar el FC
  if (Serial1.available()) {
    emergencia();
    return;
  } else {
    do {
      MZ.setSpeed(-100);
      MZ.runSpeed();
      if (Serial1.available()) { emergencia(); break; }
    } while (compruebaFC(FCZ) == 1);
  }

  // Define la posición actual como 150 mm
  delay(50);
  MZ.setCurrentPosition(grad_mmToPulsos(JZ, 150));  // 60000 pulsos

  // Si no hay datos, mueve el motor a 145 mm
  posJZ = grad_mmToPulsos(JZ, 145);
  MZ.moveTo(posJZ);
```

```
  if (Serial1.available()) {
    emergencia();
    return;
  } else {
    while (MZ.currentPosition() != posJZ) {
      MZ.run();

      // Si recibe datos, sale de bucle
      if (Serial1.available()) {
        emergencia();
        break;
      }
    }
  }
  MZ.disableOutputs();
}

// HOMING M1
void homingM1() {
  M1.enableOutputs();

  // Si no hay datos, mueve el motor hasta detectar el FC
  if (Serial1.available()) {
    emergencia();
    return;
  } else {
    while (compruebaFC(FC1) == 0) {
      M1.setSpeed(-400);
      M1.runSpeed();
      if (Serial1.available()) { emergencia(); break; }
    }
  }

  // Si no hay datos, avanza despacio hasta dejar de detectar el FC
  if (Serial1.available()) {
    emergencia();
    return;
  } else {
    do {
      M1.setSpeed(30);
      M1.runSpeed();
      if (Serial1.available()) { emergencia(); break; }
    } while (compruebaFC(FC1) == 1);
  }

  // Establece el pto de referencia adecuado
  delay(50);
  M1.setCurrentPosition(-P0CI_1);  // El 0 de este motor estará en el 0 de los ejes de CI

  // Si no hay datos, mueve el motor a la posición central
  posJ1 = grad_mmToPulsos(J1, 90.0);
  M1.moveTo(posJ1);

  if (Serial1.available()) {
    emergencia();
    return;
  } else {
```

```
    while (M1.currentPosition() != posJ1) {
      M1.run();

      // Si recibe datos, sale de bucle
      if (Serial1.available()) {
        emergencia();
        break;
      }
    }
  }
  M1.disableOutputs();
}

// HOMING M2
void homingM2() {
  M2.enableOutputs();

  // Si no hay datos, mueve el motor hasta detectar el FC
  if (Serial1.available()) {
    emergencia();
    return;
  } else {
    while (compruebaFC(FC2) == 0) {
      M2.setSpeed(-450);
      M2.runSpeed();
      if (Serial1.available()) { emergencia(); break; }
    }
  }

  // Si no hay datos, avanza despacio hasta dejar de detectar el FC
  if (Serial1.available()) {
    emergencia();
    return;
  } else {
    do {
      M2.setSpeed(20);
      M2.runSpeed();
      if (Serial1.available()) { emergencia(); break; }
    } while (compruebaFC(FC2) == 1);
  }

  // Establece el pto de referencia adecuado
  delay(50);
  M2.setCurrentPosition(-Pcentral_2);


  // Si no hay datos, mueve el motor a la posición central
  posJ2 = 0;
  M2.moveTo(posJ2);

  if (Serial1.available()) {
    emergencia();
    return;
  } else {
    while (M2.currentPosition() != posJ2) {
      M2.run();
      // Si recibe datos, sale de bucle
      if (Serial1.available()) {
```

```
        emergencia();
        break;
      }
    }
  }
  M2.disableOutputs();
}

// HOMING M3
void homingM3() {
  M3.enableOutputs();


  // Si no hay datos, mueve el motor hasta detectar el FC
  if (Serial1.available()) {
    emergencia();
    return;
  } else {
    while (compruebaFC(FC3) == 0) {
      M3.setSpeed(-200);
      M3.runSpeed();
      if (Serial1.available()) { emergencia(); break; }
    }
  }

  // Si no hay datos, avanza despacio hasta dejar de detectar el FC
  if (Serial1.available()) {
    emergencia();
    return;
  } else {
    do {
      M3.setSpeed(10);
      M3.runSpeed();
      if (Serial1.available()) { emergencia(); break; }
    } while (compruebaFC(FC3) == 1);
  }

  // Establece el pto de referencia adecuado
  delay(50);
  M3.setCurrentPosition(-Pcentral_3);

  // Si no hay datos, mueve el motor a la posición central
  posJ3 = 0;
  M3.moveTo(posJ3);

  if (Serial1.available()) {
    emergencia();
    return;
  } else {
    while (M3.currentPosition() != posJ3) {
      M3.run();

      // Si recibe datos, sale de bucle
      if (Serial1.available()) {
        emergencia();
        break;
      }
    }
```

```
  }

  M3.disableOutputs();
}

// HOMING PINZA
void homingPinza() {
  if (Serial1.available()){ emergencia(); return;} else { moverPinza(0);}   // Cerrar
  delay(200);
  if (Serial1.available()){ emergencia(); return;} else { moverPinza(100);} // Abrir
}

// HOMING DE TODAS LAS ARTICULACIONES
void homing() {
  Serial.println("HOMING");
  if (Serial1.available()){ emergencia(); return;} else { homingPinza();}
  if (Serial1.available()){ emergencia(); return;} else { homingMZ();}
  if (Serial1.available()){ emergencia(); return;} else { homingM1();}
  if (Serial1.available()){ emergencia(); return;} else { homingM2();}
  if (Serial1.available()){ emergencia(); return;} else { homingM3();}
  if (Serial1.available()){ emergencia(); return;} else { homingPinza();}
}

// Función que cuenta los pulsos de una articulación de la posición de inicio hasta el final
de carrera
// Esta función solo se ha utilizado para la calibración del Robot
void cuentaPulsosMZ(){
    MZ.enableOutputs();
    while (compruebaFC(FCZ) == 0) {MZ.setSpeed(2500); MZ.runSpeed();}
    do {MZ.setSpeed(-100); MZ.runSpeed();} while (compruebaFC(FCZ) == 1);
    Serial.println(MZ.currentPosition());
    MZ.disableOutputs();
}

// Función que transforma de grados o milímetros a pulsos para la articulación indicada
long grad_mmToPulsos(int articulacion, float distancia){
   return distancia/(vdLoad[articulacion]/vi[articulacion])*thetaStep;
}

// Función
float radianes_a_grados(float radianes) {
  float grados = radianes * 180.0 / PI;
  return grados;
}

// Función que devuelve la CI del robot SCARA para la configuración [0,1] [Codo abajo, Codo
arriba]
void CI_SCARA(float Px, float Py, bool codo){
  // Variables locales
  float D, q1_r, q1_g, q2_r, q2_g;
  long q1_p, q2_p;

  // Cálculos de Cinemática Inversa
  D = (pow(Px, 2) + pow(Py, 2) - pow(L2, 2) - pow(L1, 2)) / (2 * L1 * L2);  // Variable
auxiliar
  if (codo == 0) {q2_r = atan2(sqrt(1 - pow(D, 2)), D);}     // q2 codo abajo [rad]
  else {q2_r = atan2(-sqrt(1 - pow(D, 2)), D);}              // q2 codo arriba [rad]
  q1_r = atan2(Py, Px) - atan2(L2 * sin(q2_r), L1 + L2 * cos(q2_r));  // q1 [rad]
```

```
  // Transforma de radianes a pulsos
  q1_g = radianes_a_grados(q1_r);     // q1 [º]
  q1_p = grad_mmToPulsos(J1,q1_g);    // q1 [pul]
  q2_g = radianes_a_grados(q2_r);     // q1 [º]
  q2_p = grad_mmToPulsos(J2,q2_g);    // q1 [pul]

  // Asigna valor a las variables globales
  q1p_CI = q1_p;
  q2p_CI = q2_p;
}

// Función que obtiene la coordenada X de la bola
float leerPx (){
  digitalWrite(UR,HIGH) ; digitalWrite(LL,LOW) ;
  delay(1);
  float X = map(analogRead(signal), 182, 821, -37429.0, -10189.0) / 100.0 + offsetX;
  return X;
}

// Función que obtiene la coordenada Y de la bola
float leerPy (){
  digitalWrite(UR,LOW) ; digitalWrite(LL,HIGH) ;
  delay(1);
  float Y = map(analogRead(signal), 204, 808, -170.0 * 100.0, 170.0 * 100.0) / 100.0 +
offsetY;
  return Y;
}

void serialFlush() {
  while (Serial.available() > 0) {  // Mientras haya caracteres en el buffer
    Serial.read();                  // coge uno y lo quema
  }
}

// Funcion que ejecuta los movimientos del modo MANUAL
void modoManual(){

  // Muestra info por pantalla
  Serial.println("MANUAL");

  // Si ha cambiado, asigna valor de velocidad
  if (velAnterior != data[velocidad]){
    MZ.setSpeed(data[velocidad]);  M1.setSpeed(data[velocidad]);
    M2.setSpeed(data[velocidad]);  M3.setSpeed(data[velocidad]);
    velAnterior = data[velocidad];
  }

  // Si ha cambiado, asigna valor de aceleración
  if (accAnterior != data[aceleracion]){
    MZ.setAcceleration(data[aceleracion]);  M1.setAcceleration(data[aceleracion]);
    M2.setAcceleration(data[aceleracion]);  M3.setAcceleration(data[aceleracion]);
    accAnterior = data[aceleracion];
  }

  // Calcula posiciones absolutas
  posJZ = grad_mmToPulsos(JZ, data[alturaJZ] + 0.0);  posJ1 = grad_mmToPulsos(J1,
data[anguloJ1] + 0.0);
```

```
  posJ2 = grad_mmToPulsos(J2, data[anguloJ2] + 0.0);  posJ3 = grad_mmToPulsos(J3,
data[anguloJ3] + 0.0);

  // Asigna valores de posición a los motores
  MZ.moveTo(posJZ);  M1.moveTo(posJ1);
  M2.moveTo(posJ2);  M3.moveTo(posJ3);

  // Ejecuta el movimiento de las 4 Articulaciones
  MZ.enableOutputs();
  while (MZ.currentPosition() != posJZ || M1.currentPosition() != posJ1 ||
M2.currentPosition() != posJ2 || M3.currentPosition() != posJ3) {
    MZ.run();    M1.run();
    M2.run();    M3.run();

    // Si recibe datos, comprueba emergencias
    if (Serial1.available()) {
      emergencia();
      break;
    }
  }

  if (Serial1.available()) {return;}

  MZ.disableOutputs();

  // Mueve la pinza si es necesario
  moverPinza(data[aperturaPinza]);
}

// Función que ejecuta los pasos necesarios para alcanzar el pto 1 del modo AUTOMATICO
void autoP1(){

  // Abre la pinza si no estaba abierta
  moverPinza(100);

  // Calcula CI
  CI_SCARA(Pxb, Pyb, 0);

  // Calcula posiciones
  posJZ = grad_mmToPulsos(JZ, 50.0);
  posJ1 = q1p_CI;
  posJ2 = q2p_CI;
  posJ3 = grad_mmToPulsos(J3, 45.0);

  // Asigna valores de posición a los motores
  MZ.moveTo(posJZ);  M1.moveTo(posJ1);
  M2.moveTo(posJ2);  M3.moveTo(posJ3);

  // Si llegan datos antes de comenzar el movimiento, sale de la función
  if (Serial1.available()){
    emergencia();
    return;
  } else {

    // Ejecuta el movimiento de las 4 Articulaciones
    MZ.enableOutputs();
    while (MZ.currentPosition() != posJZ || M1.currentPosition() != posJ1 ||
M2.currentPosition() != posJ2 || M3.currentPosition() != posJ3) {
```

```
      MZ.run();    M1.run();
      M2.run();    M3.run();

      // Si recibe datos, comprueba emergencias
      if (Serial1.available()) {
        emergencia();
        break;
      }
    }
    if (Serial1.available()){return;}
    MZ.disableOutputs();
  }
}

// Función que ejecuta los pasos necesarios para alcanzar el pto 2 del modo AUTOMATICO
void autoP2() {

  // Calcula posiciones y asigna valor
  posJZ = grad_mmToPulsos(JZ, 5.0);
  MZ.moveTo(posJZ);

  // Si llegan datos antes de comenzar el movimiento, sale de la función
  if (Serial1.available()) {
    emergencia();
    return;
  } else {

    // Ejecuta el movimiento
    MZ.enableOutputs();
    while (MZ.currentPosition() != posJZ) {
      MZ.run();

      // Si recibe datos, sale del bucle
      if (Serial1.available()) {
        emergencia();
        break;
      }
    }
    if (Serial1.available()) { return; }
    MZ.disableOutputs();
  }
}

// Función que ejecuta los pasos necesarios para alcanzar el pto 3 del modo AUTOMATICO
void autoP3(){

  // Cierra la pinza y coge el objeto
  moverPinza(0);
}

// Función que ejecuta los pasos necesarios para alcanzar el pto 3_2 del modo AUTOMATICO
void autoP3_2() {

  // Calcula posiciones y asigna valor
  posJZ = grad_mmToPulsos(JZ, 50.0);
  MZ.moveTo(posJZ);

  // Si llegan datos antes de comenzar el movimiento, sale de la función
```

```
  if (Serial1.available()) {
    emergencia();
    return;
  } else {

    // Ejecuta el movimiento de las 4 Articulaciones
    MZ.enableOutputs();
    while (MZ.currentPosition() != posJZ) {
      MZ.run();

      // Si recibe datos, comprueba emergencias
      if (Serial1.available()) {
        emergencia();
        break;
      }
    }
    if (Serial1.available()) { return; }
    MZ.disableOutputs();
  }
}

// Función que ejecuta los pasos necesarios para alcanzar el pto 4 del modo AUTOMATICO
void autoP4() {

  // Calcula posiciones
  posJZ = grad_mmToPulsos(JZ, 100.0);
  posJ1 = grad_mmToPulsos(J1, 90.0);
  posJ2 = grad_mmToPulsos(J2, 0.0);
  posJ3 = grad_mmToPulsos(J3, 0.0);

  // Asigna valores de posición a los motores
  MZ.moveTo(posJZ);  M1.moveTo(posJ1);
  M2.moveTo(posJ2);  M3.moveTo(posJ3);

  // Si llegan datos antes de comenzar el movimiento, sale de la función
  if (Serial1.available()) {
    emergencia();
    return;
  } else {

    // Ejecuta el movimiento de las 4 Articulaciones
    MZ.enableOutputs();
    while (MZ.currentPosition() != posJZ || M1.currentPosition() != posJ1 ||
M2.currentPosition() != posJ2 || M3.currentPosition() != posJ3) {
      MZ.run();  M1.run();
      M2.run();  M3.run();

      // Si recibe datos, comprueba emergencias
      if (Serial1.available()) {
        emergencia();
        break;
      }
    }
    if (Serial1.available()) { return; }
    MZ.disableOutputs();
  }
}
```

```
// Función que ejecuta los pasos necesarios para alcanzar el pto 5 del modo AUTOMATICO
void autoP5() {

  // Calcula CI
  CI_SCARA(xObjetivo, yObjetivo, 1);

  // Calcula posiciones
  posJZ = grad_mmToPulsos(JZ, 70.0);
  posJ1 = q1p_CI;
  posJ2 = q2p_CI;
  posJ3 = grad_mmToPulsos(J3, -45.0);

  // Asigna valores de posición a los motores
  MZ.moveTo(posJZ);  M1.moveTo(posJ1);
  M2.moveTo(posJ2);  M3.moveTo(posJ3);

  // Si llegan datos antes de comenzar el movimiento, sale de la función
  if (Serial1.available()) {
    emergencia();
    return;
  } else {

    // Ejecuta el movimiento de las 4 Articulaciones
    MZ.enableOutputs();
    while (MZ.currentPosition() != posJZ || M1.currentPosition() != posJ1 ||
M2.currentPosition() != posJ2 || M3.currentPosition() != posJ3) {
      MZ.run();      M1.run();
      M2.run();      M3.run();

      // Si recibe datos, comprueba emergencias
      if (Serial1.available()) {
        emergencia();
        break;
      }
    }
    if (Serial1.available()) { return; }
    MZ.disableOutputs();
  }
}

// Función que ejecuta los pasos necesarios para alcanzar el pto 6 del modo AUTOMATICO
void autoP6() {

  // Calcula posiciones y asigna valores
  posJZ = grad_mmToPulsos(JZ, 35.0);
  MZ.moveTo(posJZ);

  // Si llegan datos antes de comenzar el movimiento, sale de la función
  if (Serial1.available()) {
    emergencia();
    return;
  } else {

    // Ejecuta el movimiento de las 4 Articulaciones
    MZ.enableOutputs();
    while (MZ.currentPosition() != posJZ) {
      MZ.run();
```

```
      // Si recibe datos, comprueba emergencias
      if (Serial1.available()) {
        emergencia();
        break;
      }
    }
    if (Serial1.available()) { return; }
    MZ.disableOutputs();
  }
}

// Función que ejecuta los pasos necesarios para alcanzar el pto 7 del modo AUTOMATICO
void autoP7(){

  // Abrir la pinza y coge el objeto
  moverPinza(100);
}

// Función que ejecuta los pasos necesarios para alcanzar el pto 8 del modo AUTOMATICO
void autoP8(){

  // Calcula posiciones
  posJZ = grad_mmToPulsos(JZ, 145.0);
  posJ1 = grad_mmToPulsos(J1, 90.0);
  posJ2 = grad_mmToPulsos(J2, 0.0);
  posJ3 = grad_mmToPulsos(J3, 0.0);

  // Asigna valores de posición a los motores
  MZ.moveTo(posJZ);  M1.moveTo(posJ1);
  M2.moveTo(posJ2);  M3.moveTo(posJ3);

  // Si llegan datos antes de comenzar el movimiento, sale de la función
  if (Serial1.available()) {
    emergencia();
    return;
  } else {

    // Ejecuta el movimiento de las 4 Articulaciones
    MZ.enableOutputs();
    while (MZ.currentPosition() != posJZ || M1.currentPosition() != posJ1 ||
M2.currentPosition() != posJ2 || M3.currentPosition() != posJ3) {
      MZ.run();
      M1.run();
      M2.run();
      M3.run();

      // Si recibe datos, comprueba emergencias
      if (Serial1.available()) {
        emergencia();
        break;
      }
    }
    if (Serial1.available()) { return; }
    MZ.disableOutputs();
  }
}

// Funcion que ejecuta los movimientos del modo AUTOMÁTICO
```

```
void modoAuto() {

  // Muestra info por pantalla
  Serial.println("AUTOMATICO");

  // Si ha cambiado, asigna valor de velocidad
  if (velAnterior != data[velocidad]){
    MZ.setSpeed(data[velocidad]);  M1.setSpeed(data[velocidad]);
    M2.setSpeed(data[velocidad]);  M3.setSpeed(data[velocidad]);
    velAnterior = data[velocidad];
  }

  // Si ha cambiado, asigna valor de aceleración
  if (accAnterior != data[aceleracion]){
    MZ.setAcceleration(data[aceleracion]);  M1.setAcceleration(data[aceleracion]);
    M2.setAcceleration(data[aceleracion]);  M3.setAcceleration(data[aceleracion]);
    accAnterior = data[aceleracion];
  }

  // Lectura de coordenadas con la pantalla Tactil
  delay(500);
  Pxb = leerPx();
  Pyb = leerPy();
  delay(500);

  // Secuencia de movimientos del modo AUTOMÁTICO
  if (Serial1.available()){emergencia(); return;} else {autoP1();}  //ENCIMA BOLA
  if (Serial1.available()){emergencia(); return;} else {autoP2();}  //BAJA BOLA
  if (Serial1.available()){emergencia(); return;} else {autoP3();}  //COGE BOLA
  if (Serial1.available()){emergencia(); return;} else {autoP3_2();}//SUBE CON PIEZA
  if (Serial1.available()){emergencia(); return;} else {autoP4();}  //CENTRAL
  if (Serial1.available()){emergencia(); return;} else {autoP5();}  //ENCIMA OBJETIVO
  if (Serial1.available()){emergencia(); return;} else {autoP6();}  //BAJA OBJETIVO
  if (Serial1.available()){emergencia(); return;} else {autoP7();}  //SUELTA BOLA
  if (Serial1.available()){emergencia(); return;} else {autoP8();}  //CENTRAL
}

// Función que se ejecuta al recibir una EMERGENCIA. Detiene rápidamente los motores
void emergencia() {

  // Se detienen rápidamente todos los motores
  MZ.enableOutputs();
  // Se asignan aceleraciones máximas para detener el robot rápidamente
  MZ.setAcceleration(2000); M1.setAcceleration(2000); M2.setAcceleration(2000);
M3.setAcceleration(2000);

  // Calcula las nuevas posiciones de parada y detiene motores
  MZ.stop();   M1.stop();
  M2.stop();   M3.stop();

  while (MZ.isRunning() || M1.isRunning() || M2.isRunning() || M3.isRunning()) {
    MZ.run();    M1.run();
    M2.run();    M3.run();
  }

  MZ.disableOutputs();

  // Se asignan los valores de aceleración previos
```

```
  MZ.setAcceleration(data[aceleracion]); M1.setAcceleration(data[aceleracion]);
  M2.setAcceleration(data[aceleracion]); M3.setAcceleration(data[aceleracion]);

  // Muestra INFO por pantalla
  Serial.println("EMERGENCIA: Motores detenidos");
}

// Función que recibe los datos y los muestra por palntalla.
void recibeDatos() {
  content = Serial1.readString();                         // Lee los datos y los almacena en
el estrig content
  for (int i = 0; i < 8; i++) {
    int index = content.indexOf(",");                     // Busca el primer caracter  ","
    data[i] = atol(content.substring(0, index).c_str());  //Extrae el  primer número y lo
convierte a entero
    content = content.substring(index + 1);               //Elimina el primer número y la
primera coma para repetir el proceso
  }

  // Mostrar el vector data por pantalla
  Serial.print("data: ");
  for (int i = 0; i < 8; i++) {
    Serial.print(data[i]);
    Serial.print(" ");
  }
  Serial.println();

  /*La información llega de la siguente manera:
      data[0] - ESTADO: EMERGENCIA [0], MANUAL [1], AUTOMATICO [2], HOMING [3]
      data[1] - Altura JZ [mm]
      data[2] - Ángulo J1 [º]
      data[3] - Ángulo J2 [º]
      data[4] - Ángulo J3 [º]
      data[5] - Apertura Pinza [%]
      data[6] - Velocidad
      data[7] - Aceleración
    */
}
```

2 Código del Panel de Mando en Processing

```
/*
 Programa: Panel de Mando
 Desarrollado por: Roberto Lorente Romanos
 Fecha:  5/5/2023
*/
// LIBRERÍAS
import processing.serial.*; // Librería para comunicarse con dispositivos serie (como Arduino)
import controlP5.*; // Librería que permite crear controles de usuario interactivos
import static processing.core.PApplet.*; // Librería que proporciona acceso a las funciones básicas de Processing

// OBJETOS
Serial myPort;                  // Crea un objeto de la clase Serial para la comunicación
ControlP5 cp5;                  // Crea un objeto de la clase ControlP5 para crear los controles

// CONSTANTES: Posiciones iniciales de las Articulaciones
final int JZ_ini = 145;
final int J1_ini = 90;
final int J2_ini = 0;
final int J3_ini = 0;
final int gripper_ini = 0;
final int speed_ini = 500;
final int acc_ini = 500;


// VARIABLES GLOBALES
// Se crean e inicializan variables para guardar los valores de cada elemento interactivo
int zSlider = JZ_ini;
int j1Slider = J1_ini;
int j2Slider = J2_ini;
int j3Slider = J3_ini;
int j1JogValue = 0;
int j2JogValue = 0;
int j3JogValue = 0;
int zJogValue = 0;
int speedSlider = speed_ini;
int accelerationSlider = acc_ini;
int gripperValue = gripper_ini;

// Se crean variables para poder comprobar si hay cambios en alguno de los valores y actualizarlos
int sliderzPrevious = JZ_ini;
int slider1Previous = J1_ini;
int slider2Previous = J2_ini;
int slider3Previous = J3_ini;
int speedSliderPrevious = speed_ini;
int accelerationSliderPrevious = acc_ini;
int gripperValuePrevious = gripper_ini;
```

```
// Otras variables
String data; // string para enviar todos los datos a Arduino

int marchaStatus = 0;
int pausaStatus = 0;
int stopStatus = 0;
int emergencia = 1;
int estado = 0;

// Variable para controlar los modos de marcha
int modo = 4; // modo = 1, 2, 3, 4 --> MAN, AUTO, HOMING, NINGUNO
int manualStatus = 0;
int autoStatus = 0;
int homingStatus = 0;

// Otras variables
boolean activeIK = false; // Variable que determina el tipo de cinemática que se va a utilizar
boolean codo = true; // [false, true] [Codo abajo, Codo arriba]

boolean inicio = true; // Variable utilizada para no enviar datos la 1ª vez que se pulsa MARCHA

// Variables geométricas del Robot
float xP = 0;
float yP = 364.5;
float zP = 145;
float L1 = 228;
float L2 = 136.5;

float theta1, theta2, phi, z; // Variables que representan los ángulos de las articulaciones del SCARA

// Variables de modificación de las coordenadas de visualización
int dy_FK = 237;
int dy_IK = 237;

void setup() {

 size(960, 960); // Tamaño de la ventana de 960x960 píxeles

 myPort = new Serial(this, "COM3", 115200); // Establece conexión entre la aplicación de Processing
y Arduino

 cp5 = new ControlP5(this); // Inicial la clase

 //PFont pfont = createFont("Arial", 25, true); // Carga la fuente Arial
 //ControlFont font = new ControlFont(pfont, 22); // Tamaño 22
 //ControlFont font2 = new ControlFont(pfont, 25); // Tamaño 25

 //JZ controls
 cp5.addSlider("zSlider") .setPosition(110, 565+dy_FK) .setSize(270, 30) .setRange(4, 148)
```

```
 .setValue(145) .setColorLabel(#3269c2) .setCaptionLabel("");

 cp5.addButton("zJogMinus") .setPosition(110, 618+dy_FK) .setSize(90, 40)
.setCaptionLabel("JOG-");

 cp5.addButton("zJogPlus") .setPosition(290, 618+dy_FK) .setSize(90, 40) .setCaptionLabel("JOG+");

 cp5.addNumberbox("zJogValue") .setPosition(220, 618+dy_FK) .setSize(50, 30) .setRange(0, 90)
.setMultiplier(0.1)
                              .setValue(1)    .setDirection(Controller.HORIZONTAL)     .setCaptionLabel("");

 //J1 controls
 cp5.addSlider("j1Slider") .setPosition(110, 190+dy_FK) .setSize(270, 30) .setRange(-70, 250)
.setValue(90) .setColorLabel(#3269c2) .setCaptionLabel("");

 cp5.addButton("j1JogMinus") .setPosition(110, 238+dy_FK) .setSize(90, 40)
.setCaptionLabel("JOG-");

 cp5.addButton("j1JogPlus") .setPosition(290, 238+dy_FK) .setSize(90, 40) .setCaptionLabel("JOG+");

 cp5.addNumberbox("j1JogValue") .setPosition(220, 243+dy_FK) .setSize(50, 30) .setRange(0, 90)
.setMultiplier(0.1)
                             .setValue(1)            .setDirection(Controller.HORIZONTAL)
        .setCaptionLabel("");

 //J2 controls
 cp5.addSlider("j2Slider") .setPosition(110, 315+dy_FK) .setSize(270, 30) .setRange(-149, 149)
.setValue(0) .setColorLabel(#3269c2) .setCaptionLabel("");

 cp5.addButton("j2JogMinus") .setPosition(110, 363+dy_FK) .setSize(90, 40)
.setCaptionLabel("JOG-");

 cp5.addButton("j2JogPlus") .setPosition(290, 363+dy_FK) .setSize(90, 40) .setCaptionLabel("JOG+");

 cp5.addNumberbox("j2JogValue") .setPosition(220, 368+dy_FK) .setSize(50, 30) .setRange(0, 90)
.setMultiplier(0.1)
                              .setValue(1)    .setDirection(Controller.HORIZONTAL)    .setCaptionLabel("");

 //J3 controls
 cp5.addSlider("j3Slider") .setPosition(110, 440+dy_FK) .setSize(270, 30) .setRange(-150, 150)
.setValue(0) .setColorLabel(#3269c2) .setCaptionLabel("");

 cp5.addButton("j3JogMinus") .setPosition(110, 493+dy_FK) .setSize(90, 40)
.setCaptionLabel("JOG-");

 cp5.addButton("j3JogPlus") .setPosition(290, 493+dy_FK) .setSize(90, 40) .setCaptionLabel("JOG+");

 cp5.addNumberbox("j3JogValue") .setPosition(220, 493+dy_FK) .setSize(50, 30) .setRange(0, 90)
.setMultiplier(0.1)
```

```
                    .setValue(1)    .setDirection(Controller.HORIZONTAL)     .setCaptionLabel("");

 // GRIPPER
  cp5.addSlider("gripperValue") .setPosition(605, 445+dy_IK) .setSize(190, 30) .setRange(0, 100)
.setValue(0) .setColorLabel(#3269c2) .setCaptionLabel("");

 // Campos de texto XYZ
 cp5.addTextfield("xTextfield") .setPosition(530, 205+dy_IK) .setSize(70, 40) .setColor(255)
.setCaptionLabel("");

 cp5.addTextfield("yTextfield") .setPosition(680, 205+dy_IK) .setSize(70, 40) .setColor(255)
.setCaptionLabel("");

 cp5.addTextfield("zTextfield") .setPosition(830, 205+dy_IK) .setSize(70, 40) .setColor(255)
.setCaptionLabel("");

 // Pulsadores de Mando
 cp5.addButton("move") .setPosition(590, 315+dy_IK) .setSize(240, 45) .setCaptionLabel("MOVE TO
POSITION");

 cp5.addButton("codo") .setPosition(500, 315+dy_IK) .setSize(70, 45) .setCaptionLabel("C.ARRIBA");

 cp5.addButton("marcha") .setPosition(40, 120) .setSize(150, 40) .setCaptionLabel("MARCHA");

 cp5.addButton("pausa") .setPosition(40, 180) .setSize(150, 40) .setCaptionLabel("PAUSA");

 cp5.addButton("sstop") .setPosition(40, 240) .setSize(150, 40) .setCaptionLabel("STOP");

 cp5.addButton("manual") .setPosition(230, 120) .setSize(150, 40) .setCaptionLabel("MANUAL");

 cp5.addButton("automatico") .setPosition(230, 180) .setSize(150, 40)
.setCaptionLabel("AUTOMATICO");

 cp5.addButton("homing") .setPosition(230, 240) .setSize(150, 40) .setCaptionLabel("HOMING");

 // Velocidad y Aceleración
 cp5.addSlider("speedSlider") .setPosition(490, 565+dy_FK+15) .setSize(180, 30) .setRange(500,
4000) .setColorLabel(#3269c2) .setCaptionLabel("");

 cp5.addSlider("accelerationSlider") .setPosition(720, 565+dy_FK+15) .setSize(180, 30)
.setRange(500, 4000) .setColorLabel(#3269c2) .setCaptionLabel("");

}

// Programa principal que se ejecuta en bucle. Dibuja y actualiza la interfaz gráfica
void draw() {

 background(#F2F2F2); // Fondo gris claro
 textSize(35); // Tamaño de fuente para el texto que se dibuja a continuación
```

```
  fill(33); // Establece el color de relleno del texto en negro

  // Escribe los títulos de cada sección
  text("Cinemática Directa", 120, 135+dy_FK);
  text("Cinemática Inversa", 590, 135+dy_IK);
  textSize(40);
  text("Panel de Mando", 350, 60);
  textSize(45);
  text("J1", 35, 250+dy_FK);
  text("J2", 35, 375+dy_FK);
  text("J3", 35, 500+dy_FK);
  text("Z", 35, 625+dy_FK);
  textSize(22);
  text("Velocidad", 545, 565+dy_FK);
  text("Aceleración", 745, 565+dy_FK);

  // Actualiza y rellena los valores de los sliders y los numberBox
  fill(speedSlider);
  fill(accelerationSlider);
  fill(j1Slider);
  fill(j2Slider);
  fill(j3Slider);
  fill(zSlider);
  fill(j1JogValue);
  fill(j2JogValue);
  fill(j3JogValue);
  fill(zJogValue);
  fill(gripperValue);

  // Actualiza el vector data con los valores actuales de cada variable
  updateData();

  // COMPRUEBA EL SLIDER JZ
  if (sliderzPrevious != zSlider) {
    if (activeIK == false) { // Si la CI no está activada
      zP = cp5.getController("zSlider").getValue(); // Actualiza el valor de Z
      updateData();
    }
  }
  sliderzPrevious = zSlider;

  // COMPRUEBA EL SLIDER J1
  // Si el Slider ha sido movido, calcula la nueva posición X, Y, Z con la Cinemática Directa siempre y
 cuando la CI mo esté activada
  if (slider1Previous != j1Slider) {
    if (activeIK == false) {
      theta1 = round(cp5.getController("j1Slider").getValue()); // Obtiene los ángulos de J1 y J2
      theta2 = round(cp5.getController("j2Slider").getValue());
      forwardKinematics();                                    // Calcula la CD
```

```
    updateData();
   }
  }
  slider1Previous = j1Slider;                    // Actualiza el valor del slider

  // COMPRUEBA EL SLIDER J2
  if (slider2Previous != j2Slider) {
   if (activeIK == false) {
    theta1 = round(cp5.getController("j1Slider").getValue());
    theta2 = round(cp5.getController("j2Slider").getValue());
    forwardKinematics();
    updateData();
   }
  }
  slider2Previous = j2Slider;

 // COMPRUEBA EL SLIDER J3
  if (slider3Previous != j3Slider) {
   if (activeIK == false) {
    theta1 = round(cp5.getController("j1Slider").getValue());
    theta2 = round(cp5.getController("j2Slider").getValue());
    forwardKinematics();
    updateData();
   }
  }
  slider3Previous = j3Slider;


 // COMPRUEBA EL SLIDER DEL GRIPPER
  if (gripperValuePrevious != gripperValue) {
   if (activeIK == false) {
    gripperValue = round(cp5.getController("gripperValue").getValue());
    updateData();
   }
  }
  gripperValuePrevious = gripperValue;

  activeIK = false; // Desactiva la Cinemática Inversa para la próxima iteración

  // Escribe en negro la siguiente información
  fill(33);
  textSize(32);
  text("X: ", 500, 290+dy_IK);
  text(nf(xP, 0, 1), 533, 290+dy_IK);
  text("Y: ", 650, 290+dy_IK);
  text(nf(yP, 0, 1), 685, 290+dy_IK);
  text("Z: ", 800, 290+dy_IK);
  text(nf(zP, 0, 1), 835, 290+dy_IK);
  textSize(26);
```

```
  text("Pinza", 665, 420+dy_IK);
  text("Cerrada", 510, 470+dy_IK);
  text("Abierta", 810, 470+dy_IK);
  textSize(18);


  // Visualización del botón multipropósito de ejecutar
  switch (modo) {
    case 1:
      cp5.getController("move").setCaptionLabel("MOVER A POSICION");
      break;

    case 2:
      cp5.getController("move").setCaptionLabel("BOLA SITUADA");
      break;

    case 3:
      cp5.getController("move").setCaptionLabel("INICIAR HOMING");
      break;

    default:
      cp5.getController("move").setCaptionLabel("SELECCIONE MODO");
      break;
  }

}  // FINAL DEL PROGRAMA
//-----------------------------------------------------------------------------
// FUNCIONES

// CINEMÁTICA DIRECTA
void forwardKinematics() {
  // Se crean 2 variables para convertir theta1 y 2 de grados a radianes
  float theta1F = theta1 * PI / 180;
  float theta2F = theta2 * PI / 180;
  // Se calculan las coordenadas x, y
  xP = L1 * cos(theta1F) + L2 * cos(theta1F + theta2F);
  yP = L1 * sin(theta1F) + L2 * sin(theta1F + theta2F);
}

// CINEMÁTICA INVERSA
void inverseKinematics(float x, float y, boolean codo) { //[0,1] [Codo abajo, Codo arriba]

  float D = (pow(x, 2) + pow(y, 2) - pow(L2, 2) - pow(L1, 2)) / (2 * L1 * L2); // Variable auxiliar

  if (! codo) {theta2 = atan2(sqrt(1 - pow(D, 2)), D);} // theta2 codo abajo [rad]
  else {theta2 = atan2(-sqrt(1 - pow(D, 2)), D);} // theta2 codo arriba [rad]

  theta1 = atan2(y, x) - atan2(L2 * sin(theta2), L1 + L2 * cos(theta2)); // theta1 [rad]
```

```
 theta2 = theta2 * 180 / PI; // theta2 [grados]
 theta1 = theta1 * 180 / PI; // theta1 [grados]

 // Ajuste de ángulos
 if (x < 0 & y < 0) { // 4o Cuadrante
   theta1 = theta1 + 360;
 }

 // Redondea valores
 theta1=round(theta1);
 theta2=round(theta2);

 // Asigna valores a sliders
 cp5.getController("j1Slider").setValue(theta1);
 cp5.getController("j2Slider").setValue(theta2);
 cp5.getController("zSlider").setValue(round(zP));
}

// Función que se activa al introducir un valor en X
public void xTextfield(String theText) {
 activeIK = true; // Activa el modo CI
 //xP=Integer.parseInt(theText); // Convierte el número de string a integer
 xP = Float.parseFloat(theText); // Convierte el número de string a float
 inverseKinematics(xP, yP, codo);        // Calcula posiciones J1(theta1), J2(theta2), y J3(phi)
}

// Función que se activa al introducir un valor en Y
public void yTextfield(String theText) {
 //yP=Integer.parseInt(theText);
 yP = Float.parseFloat(theText); // Convierte el número de string a float
 activeIK = true;
 inverseKinematics(xP, yP, codo);
}

// Función que se activa al introducir un valor en Z
public void zTextfield(String theText) {
 //zP=Integer.parseInt(theText);
 zP = Float.parseFloat(theText); // Convierte el número de string a float
 activeIK = true;
 inverseKinematics(xP, yP, codo);
}

// Función que se activa al apretar el botón JOGZ-
public void zJogMinus() {
 int a  =  round(cp5.getController("zSlider").getValue());
 a = a - zJogValue;
 cp5.getController("zSlider").setValue(a);
}
```

```
// Función que se activa al apretar el botón JOG1+
public void zJogPlus() {
 int a = round(cp5.getController("zSlider").getValue());
 a=a+zJogValue;
 cp5.getController("zSlider").setValue(a);
}

// Función que se activa al apretar el botón JOG1-
public void j1JogMinus() {
 int a = round(cp5.getController("j1Slider").getValue()); // Obtiene el valor del slider
 a=a-j1JogValue;                                          // Le resta el valor del JOG
 cp5.getController("j1Slider").setValue(a); // Asigna el nuevo valor al slider
}

// Función que se activa al apretar el botón JOG1+
public void j1JogPlus() {
 int a = round(cp5.getController("j1Slider").getValue());
 a=a+j1JogValue;
 cp5.getController("j1Slider").setValue(a);
}

// Función que se activa al apretar el botón JOG2-
public void j2JogMinus() {
 int a = round(cp5.getController("j2Slider").getValue());
 a=a-j2JogValue;
 cp5.getController("j2Slider").setValue(a);
}

// Función que se activa al apretar el botón JOG1+
public void j2JogPlus() {
 int a = round(cp5.getController("j2Slider").getValue());
 a=a+j2JogValue;
 cp5.getController("j2Slider").setValue(a);
}

// Función que se activa al apretar el botón JOG3-
public void j3JogMinus() {
 int a = round(cp5.getController("j3Slider").getValue());
 a=a-j3JogValue;
 cp5.getController("j3Slider").setValue(a);
}

// Función que se activa al apretar el botón JOG1+
public void j3JogPlus() {
 int a = round(cp5.getController("j3Slider").getValue());
 a=a+j3JogValue;
 cp5.getController("j3Slider").setValue(a);
}
```

```
// Función que se activa al presionar MARCHA/PARO
public void marcha() {

 // VIENE DE PARO Y VA A MARCHA
 if (marchaStatus == 0) {
  // Manejo de variables
  marchaStatus = 1;
  emergencia = 0;    // No hay EMERGENCIA
  // Se envían los datos siempre a excepción de la primera vez que se pulsa MARCHA
  if (!inicio){ sendData();}
  if (inicio) {inicio = false;}


  // Visualización
  cp5.getController("marcha").setCaptionLabel("PARO"); // Cambia nombre
  cp5.getController("marcha").setColorBackground(color(46, 204, 113)); // Color verde

 // VIENE DE MARCHA Y VA A PARO
 } else if (marchaStatus == 1) {
  // Manejo de variables y envío de datos
  marchaStatus = 0;
  emergencia = 1;      // Hay EMERGENCIA
  pausaStatus = 0;     stopStatus = 0;    modo = 0;
  manualStatus = 0;    autoStatus = 0;    homingStatus = 0;
  sendData();

  cp5.getController("marcha").setCaptionLabel("MARCHA"); // Restablece nombre
  cp5.getController("marcha").setColorBackground(color(0,45,90)); // Marcha en Azul

  cp5.getController("pausa").setCaptionLabel("PAUSA"); // Restablece nombre
  cp5.getController("pausa").setColorBackground(color(0,45,90)); // Azul

  cp5.getController("sstop").setCaptionLabel("STOP"); // Restablece nombre
  cp5.getController("sstop").setColorBackground(color(0,45,90)); // Azul

  cp5.getController("manual").setColorBackground(color(0,45,90)); // Azul Oscuro
  cp5.getController("automatico").setColorBackground(color(0,45,90)); // Azul Oscuro
  cp5.getController("homing").setColorBackground(color(0,45,90)); // Azul Oscuro

 }
}

// Función que se activa al presionar PAUSA/CONTINÚA
public void pausa() {

 // Si está en AUTOMÁTICO, al pulsar PAUSA no sucede nada
 if (modo == 2){
```

```
    return;
  } else {
    // VIENE DE INICIO O CONTINÚA Y VA A PAUSA
    if (pausaStatus == 0) {

      // Manejo de variables
      pausaStatus = 1;
      emergencia = 1;      // Hay EMERGENCIA
      sendData();

      // Visualización
      cp5.getController("pausa").setCaptionLabel("CONTINUA"); // Cambia la etiqueta
      cp5.getController("pausa").setColorBackground(color(255,0,0)); // Rojo


    // VIENE DE PAUSA Y VA A CONTINÚA
    } else if (pausaStatus == 1) {

      // Manejo de variables
      pausaStatus = 0;
      emergencia = 0;    // No hay EMERGENCIA
      sendData();

      // Visualización
      cp5.getController("pausa").setCaptionLabel("PAUSA"); // Restablece nombre
      cp5.getController("pausa").setColorBackground(color(0,45,90)); // Azul
    }
  }
}

// Función que se activa al presionar STOP/REARME
public void sstop() {

  // Si está en HOMING, al pulsar STOP no sucede nada
  if (modo == 3){
    return;
  } else {
    // VIENE DE INICIO O REARME Y VA A STOP
    if (stopStatus == 0) {

      // Manejo de variables
      stopStatus = 1;
      emergencia = 1;      // Hay EMERGENCIA
      pausaStatus = 0;     modo = 0;
      manualStatus = 0;    autoStatus = 0;    homingStatus = 0;
      sendData();

      // Visualización
      cp5.getController("sstop").setCaptionLabel("REARME"); // Cambia la etiqueta
```

```
    cp5.getController("sstop").setColorBackground(color(255,0,0)); // Rojo

    cp5.getController("pausa").setCaptionLabel("PAUSA"); // Restablece nombre
    cp5.getController("pausa").setColorBackground(color(0,45,90)); // Azul

    cp5.getController("manual").setColorBackground(color(0,45,90)); // Azul Oscuro
    cp5.getController("automatico").setColorBackground(color(0,45,90)); // Azul Oscuro
    cp5.getController("homing").setColorBackground(color(0,45,90)); // Azul

  // VIENE DE STOP Y VA A REARME
  } else if (stopStatus == 1) {

    // Manejo de variables
    stopStatus = 0;
    emergencia = 0;    // No hay EMERGENCIA

    // Al rearmar, asigna los valores iniciales a los sliders
    cp5.getController("zSlider").setValue(JZ_ini);
    cp5.getController("j1Slider").setValue(J1_ini);
    cp5.getController("j2Slider").setValue(J2_ini);
    cp5.getController("j3Slider").setValue(J3_ini);
    cp5.getController("gripperValue").setValue(gripper_ini);

    // Genera el vector data de forma "manual"
    data = str(1) // Envía 1 para que entre en modo MANUAL en Arduino
      +","+str(round(cp5.getController("zSlider").getValue()))
      +","+str(round(cp5.getController("j1Slider").getValue()))
      +","+str(round(cp5.getController("j2Slider").getValue()))
      +","+str(round(cp5.getController("j3Slider").getValue()))
      +","+str(gripperValue)
      +","+str(speedSlider)
      +","+str(accelerationSlider);

    // Envía y muestra datos
    myPort.write(data);
    println("Datos: "+data);

    // Visualización
    cp5.getController("sstop").setCaptionLabel("STOP"); // Restablece nombre
    cp5.getController("sstop").setColorBackground(color(0,45,90)); // Azul
   }
  }
 }

 // Función que se activa al presionar MANUAL
 public void manual() {

  if (manualStatus == 0) {
    manualStatus = 1;                                    // Activa la variable asignada al botón
```

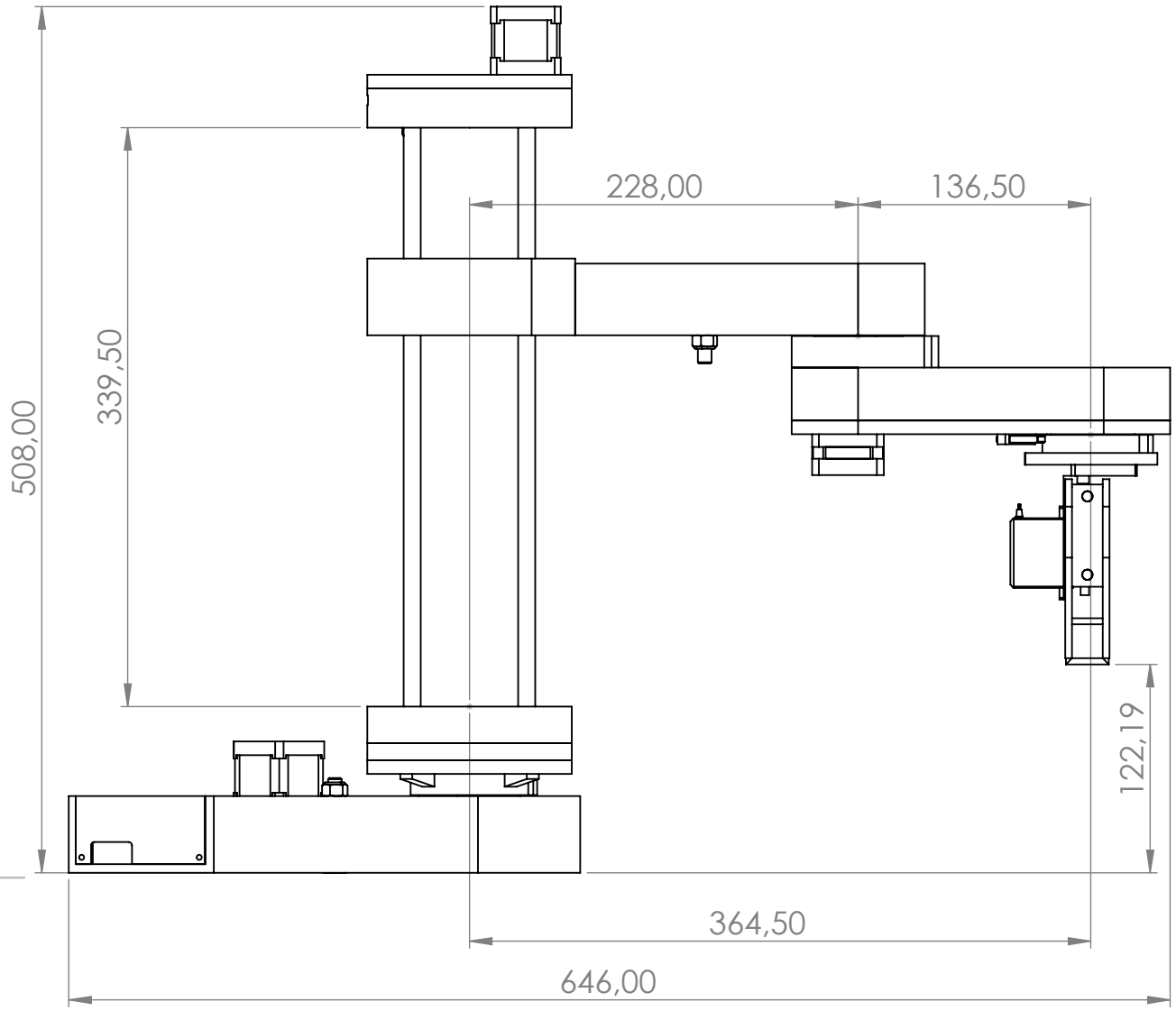
```
    modo = 1;                                                  // Establece valor al indicador de modo
    autoStatus = 0;                                            // Desactiva el modo auto
    homingStatus = 0;                                          // Desactiva el modo homing
    cp5.getController("manual").setColorBackground(color(46, 204, 113)); // Marcha en Verde
    cp5.getController("automatico").setColorBackground(color(0,45,90)); // Manual en Azul Oscuro
    cp5.getController("homing").setColorBackground(color(0,45,90)); // Homing en Azul Oscuro
    cp5.getController("sstop").setCaptionLabel("STOP"); // "Activa" el botón de STOP
    cp5.getController("pausa").setCaptionLabel("PAUSA"); // "Activa" el botón PAUSA


  } else if (manualStatus == 1) { // En esta parte entra al volver a pulsar
    manualStatus = 0;                                          // Desactiva la variable asignada
    modo = 4;                                                  // Establece valor al indicador de modo
    cp5.getController("manual").setColorBackground(color(0,45,90)); // Azul Oscuro
  }
}

// Función que se activa al presionar AUTOMÁTICO
public void automatico() {
  if (autoStatus == 0) {
    autoStatus = 1;                                            // Activa la variable asignada al botón
    modo = 2;                                                  // Establece valor al indicador de modo
    manualStatus = 0;                                          // Desactiva el modo manual
    homingStatus = 0;                                          // Desactiva el modo homing
    cp5.getController("automatico").setColorBackground(color(46, 204, 113)); // Auto en Verde
    cp5.getController("manual").setColorBackground(color(0,45,90)); // Manual en Azul Oscuro
    cp5.getController("homing").setColorBackground(color(0,45,90)); // Homing en Azul Oscuro
    cp5.getController("pausa").setCaptionLabel(""); // "Desactiva" el botón PAUSA
    cp5.getController("sstop").setCaptionLabel("STOP"); // "Activa" el botón de STOP


  } else if (autoStatus == 1) { // En esta parte entra al volver a pulsar
    autoStatus = 0;                                            // Desactiva la variable asignada
    modo = 4;                                                  // Establece valor al indicador de modo
    cp5.getController("automatico").setColorBackground(color(0,45,90)); // Auto en Azul Oscuro
    cp5.getController("pausa").setCaptionLabel("PAUSA"); // "Reactiva" el botón PAUSA
  }
}

// Función que se activa al presionar HOMING
public void homing() {
  if (homingStatus == 0) {
    homingStatus = 1;                                          // Activa la variable asignada al botón
    modo = 3;                                                  // Establece valor al indicador de modo
    manualStatus = 0;                                          // Desactiva el modo manual
    autoStatus = 0;                                            // Desactiva el modo auto

    cp5.getController("homing").setColorBackground(color(46, 204, 113)); // Homing en Verde
    cp5.getController("manual").setColorBackground(color(0,45,90)); // Manual en Azul Oscuro
```

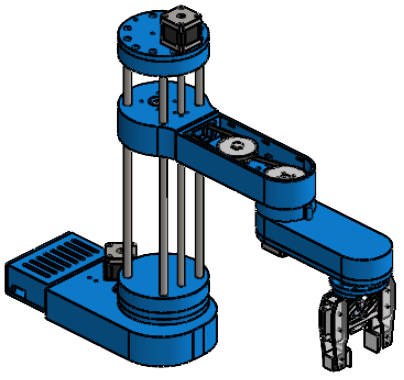
```
    cp5.getController("automatico").setColorBackground(color(0,45,90)); // Auto en Azul Oscuro

    cp5.getController("sstop").setCaptionLabel(""); // "Desactiva" el botón de STOP
    cp5.getController("pausa").setCaptionLabel("PAUSA"); // Visualiza el botón PAUSA

  } else if (homingStatus == 1) { // En esta parte entra al volver a pulsar
    homingStatus = 0;                                    // Desactiva la variable asignada
    modo = 4;                                           // Establece valor al indicador de modo
    cp5.getController("homing").setColorBackground(color(0,45,90)); // Homing en Azul Oscuro

    cp5.getController("sstop").setCaptionLabel("STOP"); // "Activa" el botón de STOP
  }
}

// Función que se activa al presionar el botón de CODO
// [false, true] [Codo abajo, Codo arriba]
public void codo() {

  if (codo) { // Si codo era true, se cambia a false
    cp5.getController("codo").setCaptionLabel("C.ABAJO"); // Cambia la etiqueta
    codo = false; // Cambia la variable asignada al botón
  } else { // Si codo era true, se cambia a false
    cp5.getController("codo").setCaptionLabel("C.ARRIBA");// Cambia la etiqueta
    codo = true; // Cambia la variable asignada al botón
  }
  activeIK = true; // Activa bandera de CI
  inverseKinematics(xP, yP, codo);                    // Recalcula CI
}

// Función que se activa al pulsar el botón MOVE TO POSITION
public void move() {
  // Llama a la función de actualizar y enviar datos
  sendData();
}

// Función que actualiza el vector data con los valores actuales de cada variable
public void updateData() {
  data = str(estado)
    +","+str(round(cp5.getController("zSlider").getValue()))
    +","+str(round(cp5.getController("j1Slider").getValue()))
    +","+str(round(cp5.getController("j2Slider").getValue()))
    +","+str(round(cp5.getController("j3Slider").getValue()))
    +","+str(gripperValue)
    +","+str(speedSlider)
    +","+str(accelerationSlider);
}

// Función que actualiza y envía los datos
void sendData(){
```

```
  // Asigna valor a "estado"
  if (emergencia == 1){
  estado = 0;} else if (modo !=
  0) {estado = modo;}

  // Si hay emergencia o si no hay emergencia y hay un estado seleccionado, actualiza y
  envía datos
  if ((estado == 0) || ((estado != 0) && (modo != 4))){
    updateData();
    myPort.write(d
    ata);
    println("Datos:
    "+data);
  }
 }
```

ANEXO II: Planos

ANEXO III: Datasheet





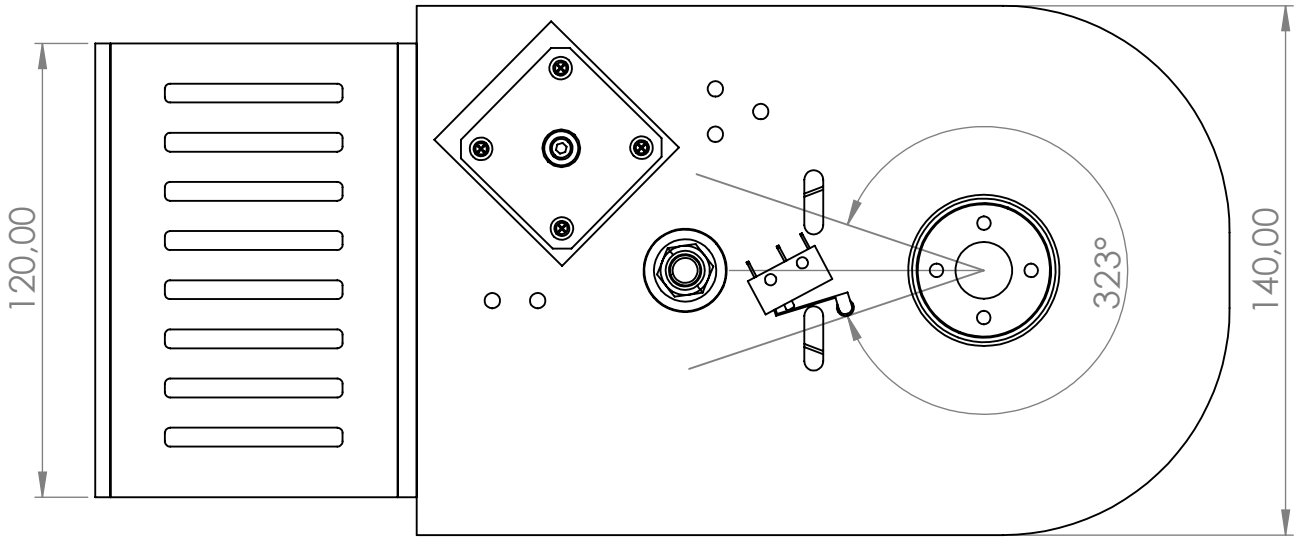
SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO:	REBARBAR Y ROMPER ARISTAS VIVAS	NO CAMBIE LA ESCALA / REVISIÓN

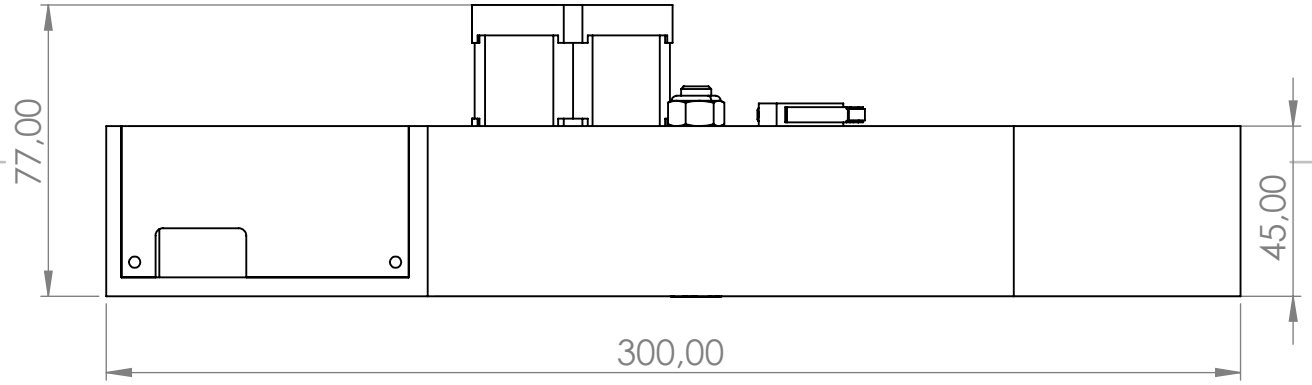
	NOMBRE	FIRMA	FECHA
DIBUJ.			
VERIF.			
APROB.			
FABR.			
CALID.			

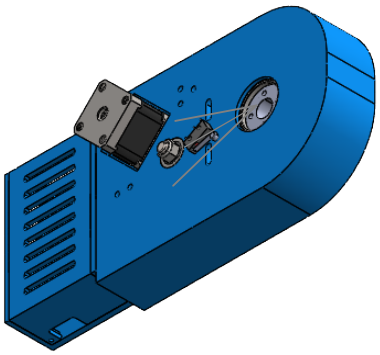
TÍTULO:

ENS_SCARA

MATERIAL:	N.º DE DIBUJO	A4
PESO:	ESCALA:1:10	HOJA 1 DE 1







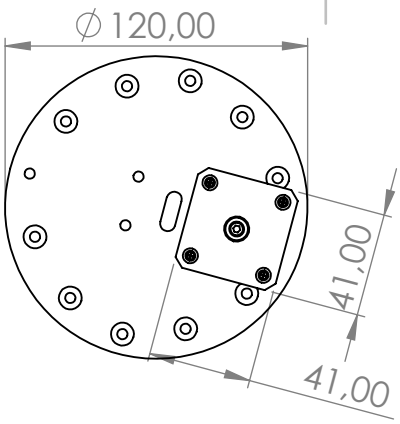
SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO:	REBARBAR Y ROMPER ARISTAS VIVAS	NO CAMBIE LA ESCALA / REVISIÓN

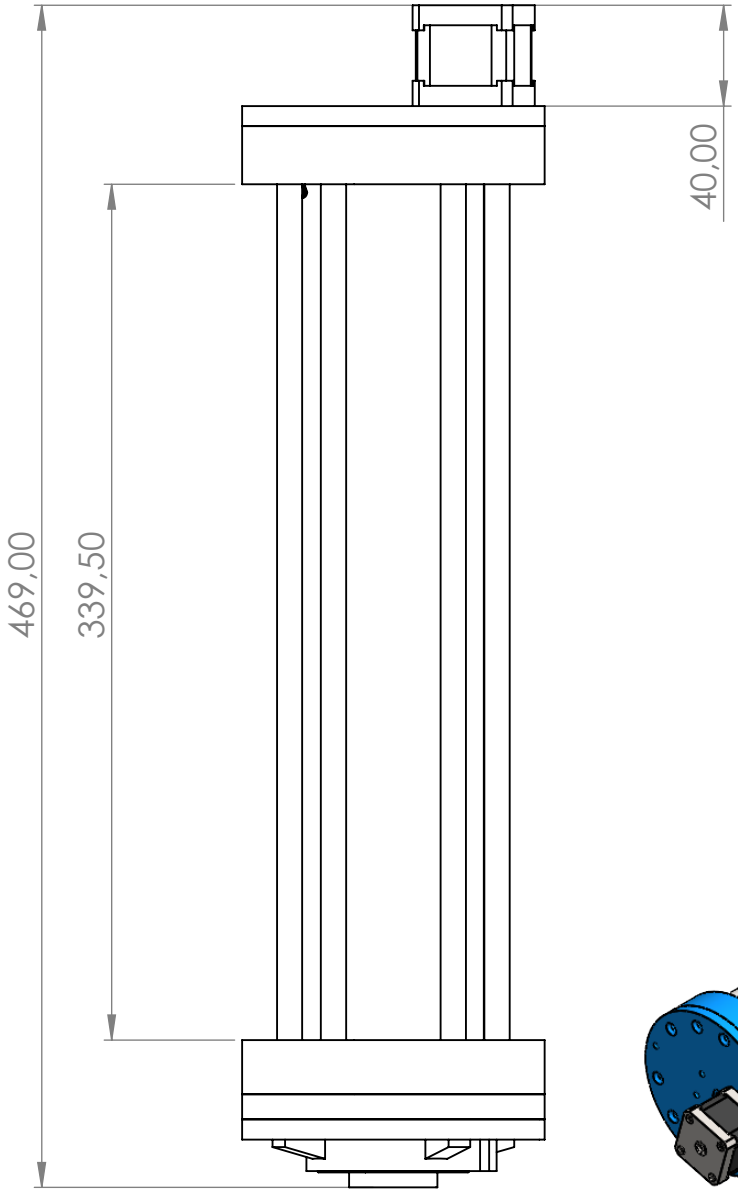
	NOMBRE	FIRMA	FECHA
DIBUJ.			
VERIF.			
APROB.			
FABR.			
CALID.			

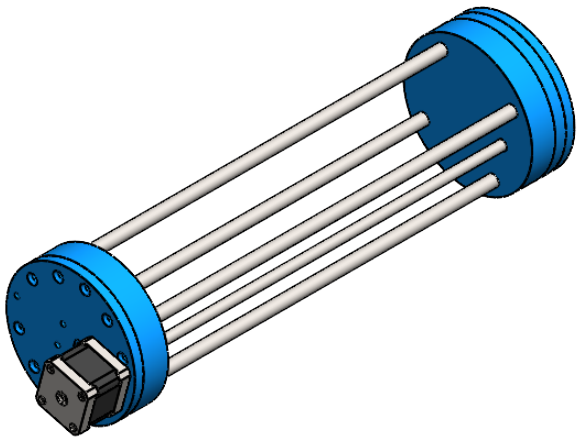
TÍTULO:

ENS_ BASE

MATERIAL:	N.º DE DIBUJO	A4
PESO:	ESCALA:1:5	HOJA 1 DE 1







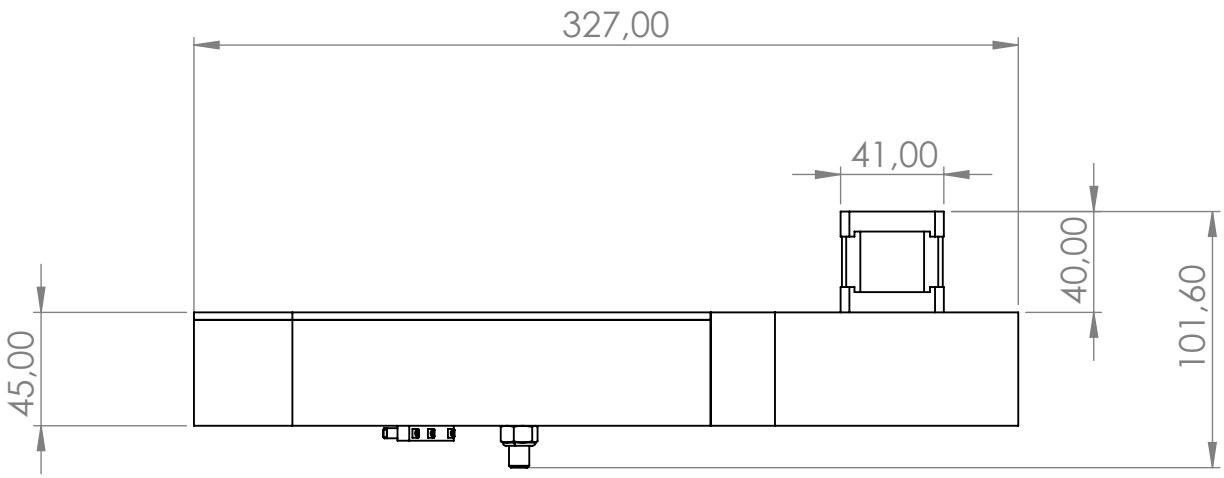
SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO:	REBARBAR Y ROMPER ARISTAS VIVAS	NO CAMBIE LA ESCALA / REVISIÓN

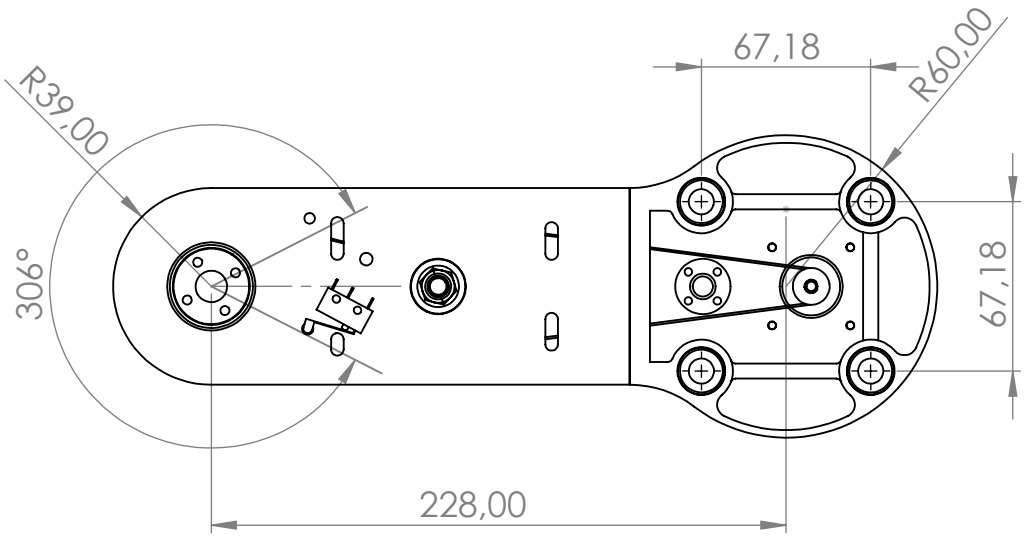
	NOMBRE	FIRMA	FECHA
DIBUJ.			
VERIF.			
APROB.			
FABR.			
CALID.			

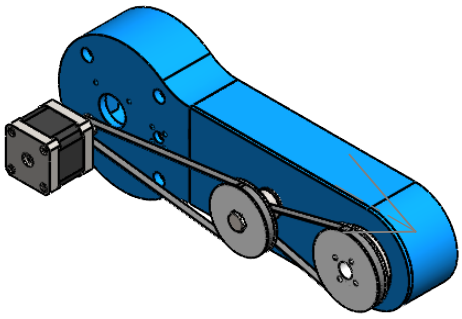
TÍTULO:

ENS_TORRE_ELEVACIÓN

MATERIAL:	N.º DE DIBUJO	A4
PESO:	ESCALA:1:10	HOJA 1 DE 1







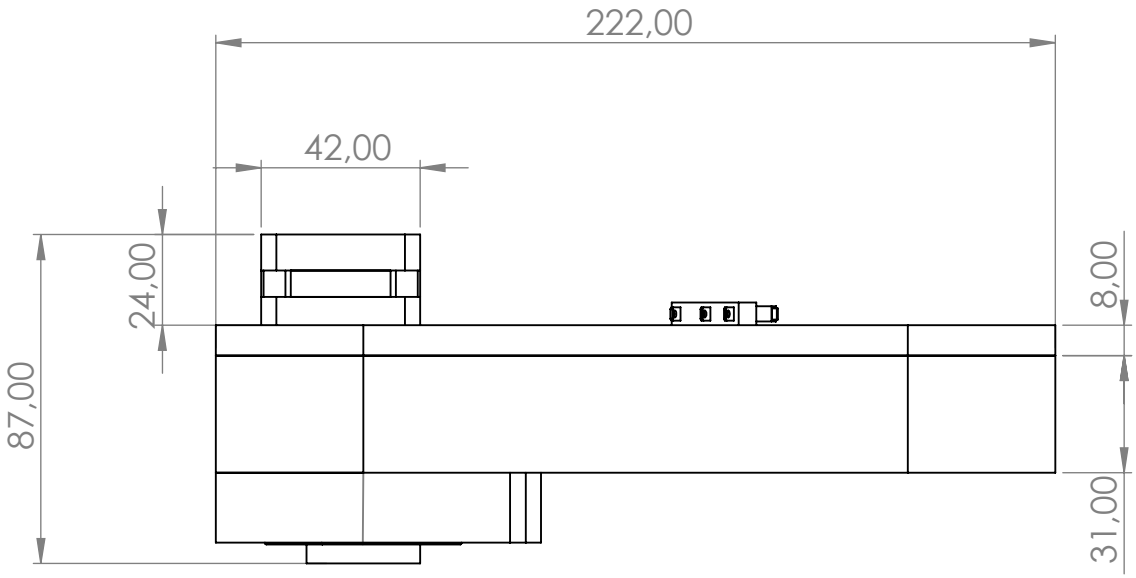
SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO:	REBARBAR Y ROMPER ARISTAS VIVAS	NO CAMBIE LA ESCALA / REVISIÓN

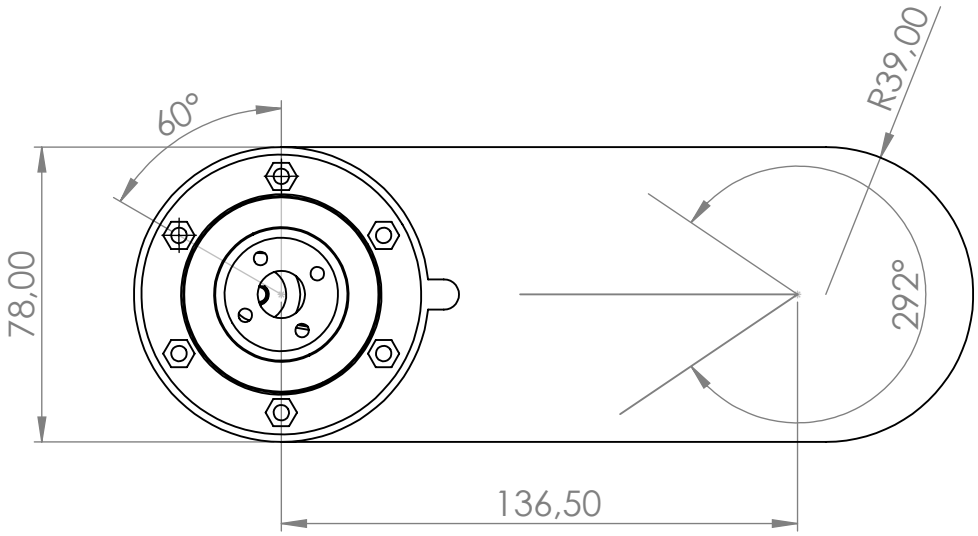
	NOMBRE	FIRMA	FECHA
DIBUJ.			
VERIF.			
APROB.			
FABR.			
CALID.			

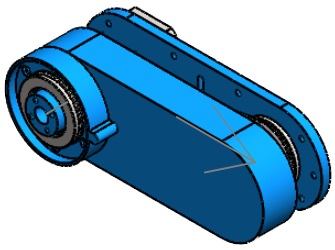
TÍTULO:

ENS_BRAZO_1

MATERIAL:	N.º DE DIBUJO	A4
PESO:	ESCALA:1:5	HOJA 1 DE 1







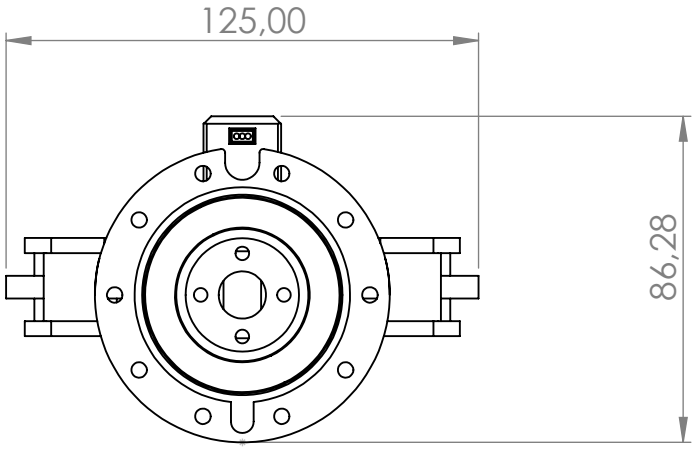
SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO:	REBARBAR Y ROMPER ARISTAS VIVAS	NO CAMBIE LA ESCALA / REVISIÓN

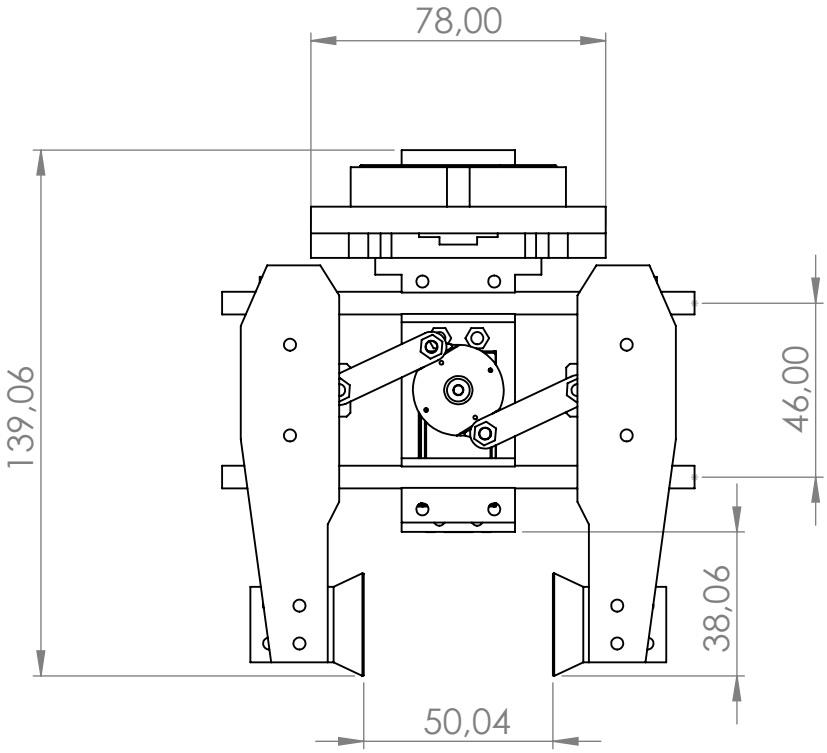
	NOMBRE	FIRMA	FECHA
DIBUJ.			
VERIF.			
APROB.			
FABR.			
CALID.			

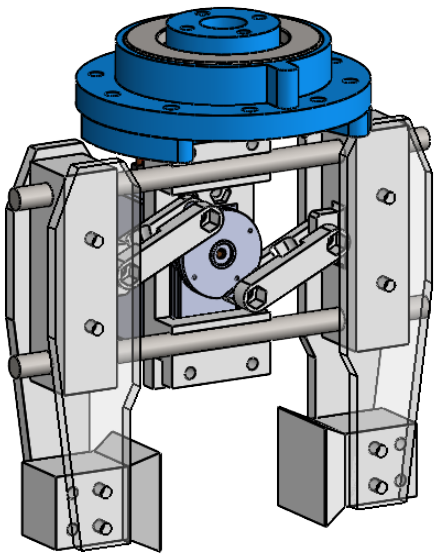
TÍTULO:

ENS_BRAZO_2

MATERIAL:	N.º DE DIBUJO	A4
PESO:	ESCALA:1:5	HOJA 1 DE 1







SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO:	REBARBAR Y ROMPER ARISTAS VIVAS	NO CAMBIE LA ESCALA / REVISIÓN

	NOMBRE	FIRMA	FECHA
DIBUJ.			
VERIF.			
APROB.			
FABR.			
CALID.			

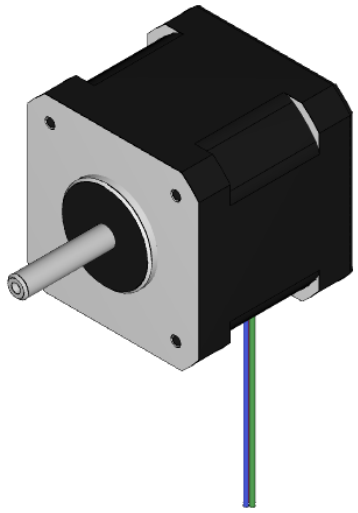
TÍTULO:

ENS_PINZA

MATERIAL:	N.º DE DIBUJO	A4
PESO:	ESCALA:1:2	HOJA 1 DE 1

LIN ENGINEERING
MOTOR SPECIALISTS IN MOTION
A MOONS' COMPANY

16245 Vineyard Blvd, Morgan Hill, CA 95037
Phone: (408) 919-0200 Fax: (408) 919-0201
Email: sales@linengineering.com
Email: techsupport@linengineering.com
www.linengineering.com



4118 SERIES

4118M-06P

HYBRID STEPPER MOTOR

CONFIGURATION

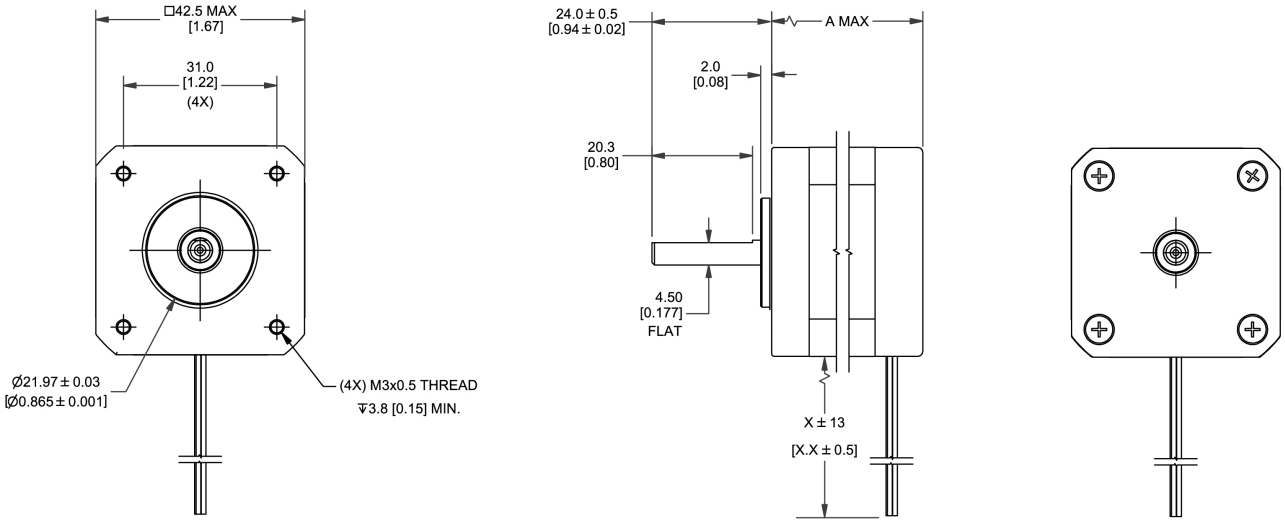
Gearbox : No Gearbox
Shaft : Single Shaft
Shaft Option (front shaft) : Flat Shaft
Encoder : No Encoder
Connector : Flying Leads
Leadwire Length : 24 inch (61.00 cm)

MOTOR SPECIFICATIONS

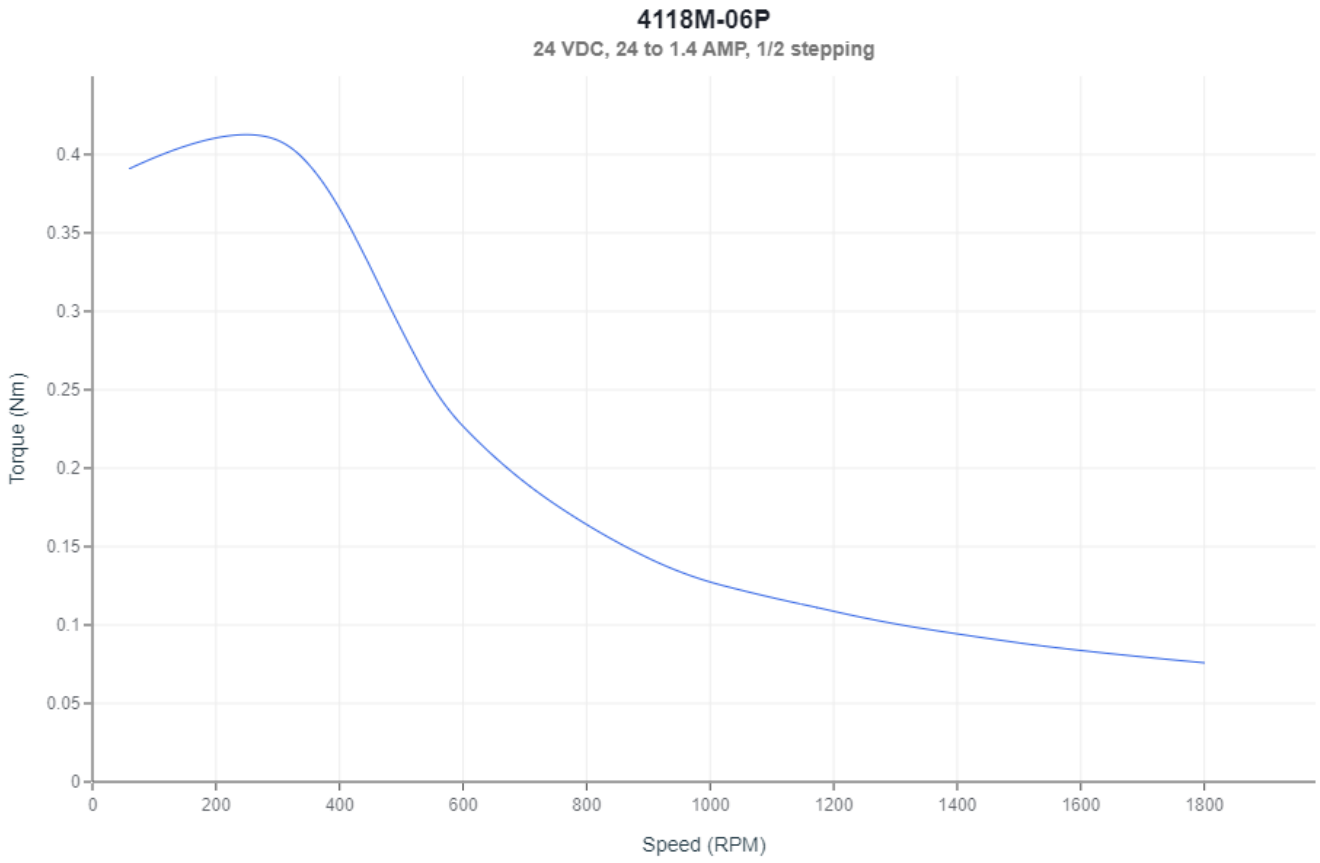
Base Motor Part Number	4118M-06P
Configured Motor Part Number	WO-4118M-06P
Step Angle	1.8°
Frame Size	42.4 mm
NEMA Size	NEMA 17
Body Length	40.1 mm
Current (AMP)	1.4 AMP
Holding Torque	0.44 Nm
Resistance	2.7
Rotor Inertia	51.22 g-cm²
Number of leads	4
Connection	Parallel
Weight	0.273 kg

DIMENSIONS

Model	WO-4118M-06P
DIM. A (length) mm	40.1 mm



PERFORMANCE CURVE



OPERATING SPECIFICATIONS

Radial Play	0.03 mm @ 0.454 kg
End Play	0.08 mm @ 1.361 kg
Shaft Run Out	0.05 TIR
Concentricity of Mounting Pilot to Shaft	0.08 TIR
Perpendicularity of Mounting Pilot to Face	0.08 TIR
Max Radial Load at Dimension "K" from mounting face	2.722 kg
Dimension "K"	15.75 mm
Max Axial Load	2.722 kg
Maximum Case Temperature	80.00 °C maximum
Ambient Temperature	-20.00 ° to 50.00 °C
Storage Temperature	-20.00 ° to 100.00 °C
Humidity Range (%)	85% or less, non-condensing
Magnet Wire Insulation	Class B 130 deg C
Insulation Resistance	100M Ohm at 500 VDC
Dielectric Strength	500 VDC for 1 min

FEATURES

https://www.youtube.com/embed/4n2C_7a6E54

Unbeatable Value

The 4118 Series stepper motor is our best-selling stepper motor for numerous reasons: it delivers unbeatable balance between high performance and low price; it offers a high range of customizations; and its wide performance range makes it ideal for many applications.

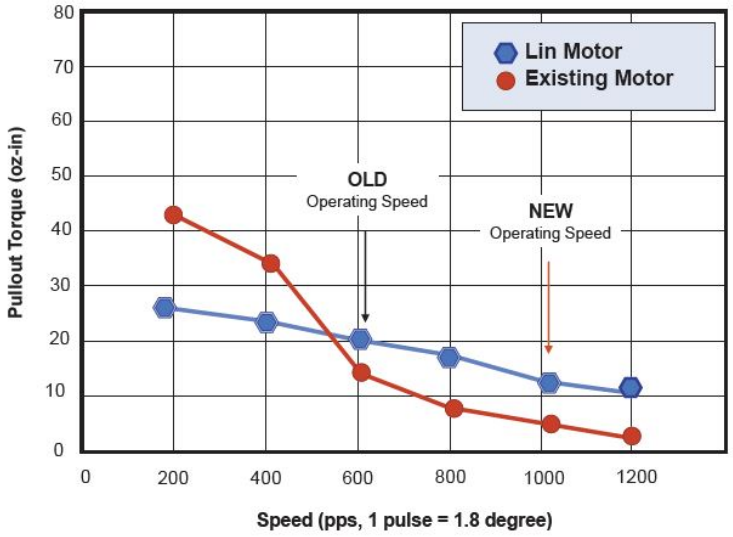
Versatile performance

The 4118 Series stepper motor is a good fit for wide range of applications. The series is available in many stator lengths, from 1.34" (34mm) to 2.34" (59.4mm). Holding torque ranges from 44 oz-in (0.3 Nm), up to 115 oz-in (0.88 Nm). High range of dynamic torque with speed up to 1200 RPM. And a large selection of windings to meet your specific requirements.

Lin Engineering Quality

Every component and every motor that leaves our facility must meet our mean value control. Additionally, every motor is tested to meet the required electrical specifications (resistance, inductance, leakage), torque specifications (holding and detent torque), mechanical specification (front shaft extension dimension and overall body length), and any other special feature specification. We want to ensure that your motor delivers the precise specifications you require. This gives you confidence that your motors will perform consistently and reliably within your application.

Take advantage of our Custom Winding Services



Motion Control, Solved.

Motor Engineering and Manufacturing

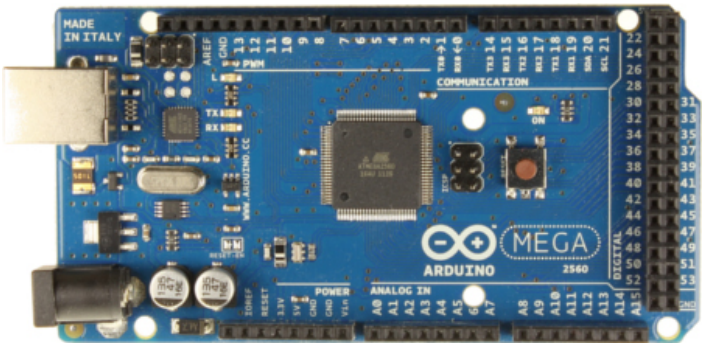
Optimized For Your Applications

Quick Prototype Turnaround

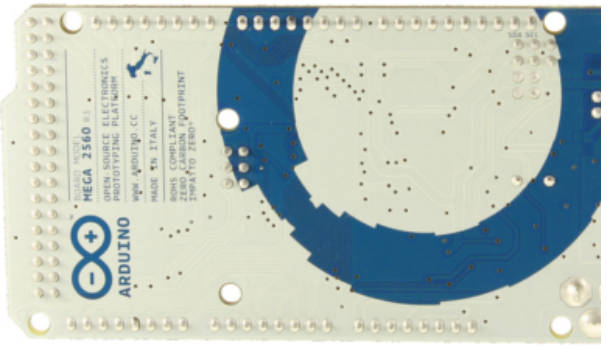
Small Batch to OEM Volume Production

US Based Support & Manufacturing

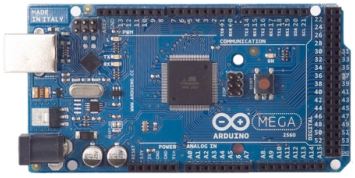
Arduino Mega 2560



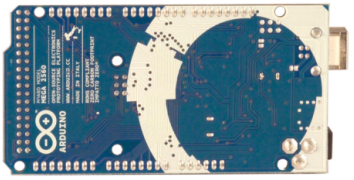
Arduino Mega 2560 R3 Front



Arduino Mega2560 R3 Back



Arduino Mega 2560 Front



Arduino Mega 2560 Back

Buy From Arduino Store Buy From Distributors

Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

The Mega 2560 is an update to the Arduino Mega, which it replaces.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the ATmega16U2 (ATmega8U2 in the revision 1 and revision 2 boards) programmed as a USB-to-serial converter.
Revision 2 of the Mega2560 board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode.
Revision 3 of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with

3.3V. The second one is a not connected pin, that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

Schematic, Reference Design & Pin Mapping

EAGLE files: arduino-mega2560_R3-reference-design.zip

Schematic: arduino-mega2560_R3-schematic.pdf

Pin Mapping: PinMap2560 page

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Power

The Arduino Mega can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

- **5V.** This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.

- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and

digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.

- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.

- **PWM: 2 to 13 and 44 to 46.** Provide 8-bit PWM output with the analogWrite() function.

- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.

- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

- **TWI: 20 (SDA) and 21 (SCL).** Support TWI communication using the Wire library. Note that these pins are not in the same location as the TWI pins on the Duemilanove or Diecimila.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().

- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega16U2 (ATmega 8U2 on the revision 1 and revision 2 boards) on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2/ATmega16U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Mega2560's digital pins.

The ATmega2560 also supports TWI and SPI communication. The Arduino software includes a Wire library to simplify use of the TWI bus; see the documentation for details. For SPI communication, use the SPI library.

Programming

The Arduino Mega can be programmed with the Arduino software (download). For details, see the reference and tutorials.

The ATmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available in the Arduino repository. The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode. You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega2560 contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

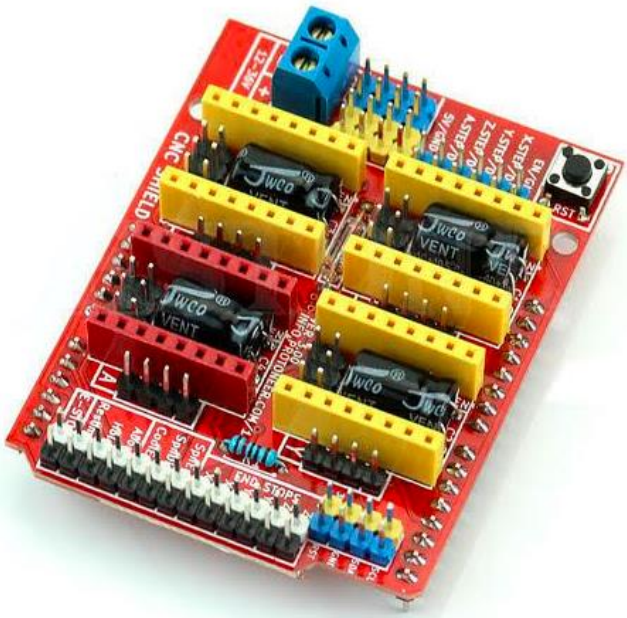
The Arduino Mega2560 has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega2560 PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega2560 is designed to be compatible with most shields designed for the Uno, Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega2560 and Duemilanove / Diecimila. *Please note that I^2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).*

CNC Shield For A4988



The CNC Shield V3.0 allows to build a engraving machine, 3D printer, mini CNC and other similar devices using your Arduino. It is designed as a shield and can plug on top of an Arduino requiring no external connections and wiring. There are 4 slots on the board for plugging in stepper motor drive module which can drive 1 stepper motor each. Controlling each step stepper motor requires only two IO pins on the Arduino. Just insert this Arduino CNC Shield V3.0 onto an Arduino UNO, and install the GRBL firmware, so it can quickly build a DIY CNC engraving machine.

FEATURES:

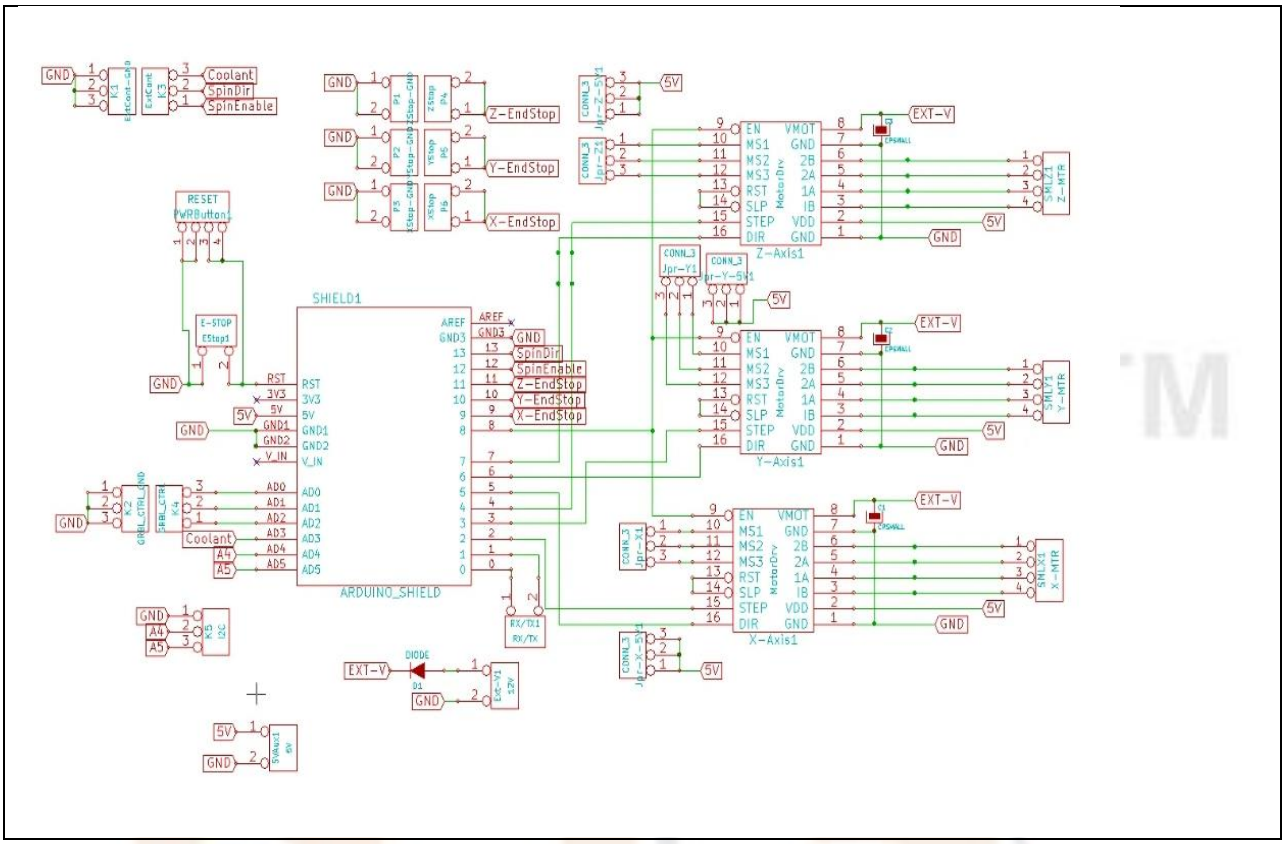
- The extension board can be used for Small CNC routers, Carving Machine, 3D Printers, DIY Laser Cutters, and almost any project where you need to control a stepper motors with high precision
- This shield allows you to control upto 4 stepper motors
- Controller each stepper motor requires 2 IO Pins only, which saves a lot of IO Pins for other purposes

- Arduino compatible
- Latest Arduino CNC Shield Version 3.10
- GRBL 0.9 compatible. (Open source firmware that runs on an Arduino UNO that turns G-code commands into stepper signals)
- Supports PWM Spindle and direction pins
- 4-Axis support (X, Y, Z , A-Can duplicate X,Y,Z or do a full 4th axis with custom firmware using pins D12 and D13)
- Supports Coolant enable
- Supports removable A4988 compatible stepper drivers.
- Jumpers to set the Micro-Stepping for the stepper drivers. (Some drivers like the DRV8825 can do up to 1/32 micro-stepping)
- Compact design.
- Stepper Motors can be connected with 4 pin molex connectors or soldered in place.
- Runs on 12-36V DC. (At the moment only the DRV8825 drivers can handle up to 36V so please consider the operation voltage when powering the board.)
- Uses removable A4988 or DRV8825 compatible stepping driver.

SPECIFICATIONS:

- Motor Voltage: 8 V to 35 V
- Logic Circuits Voltage: 3 V to 5.5 V
- Current: 2 A (MAX)
- Five step resolutions: full, 1/2, 1/4, 1/8 and 1/16
- Protection: under-voltage, over-current and over-temperature
- External resources

SCHEMATIC DIAGRAM:

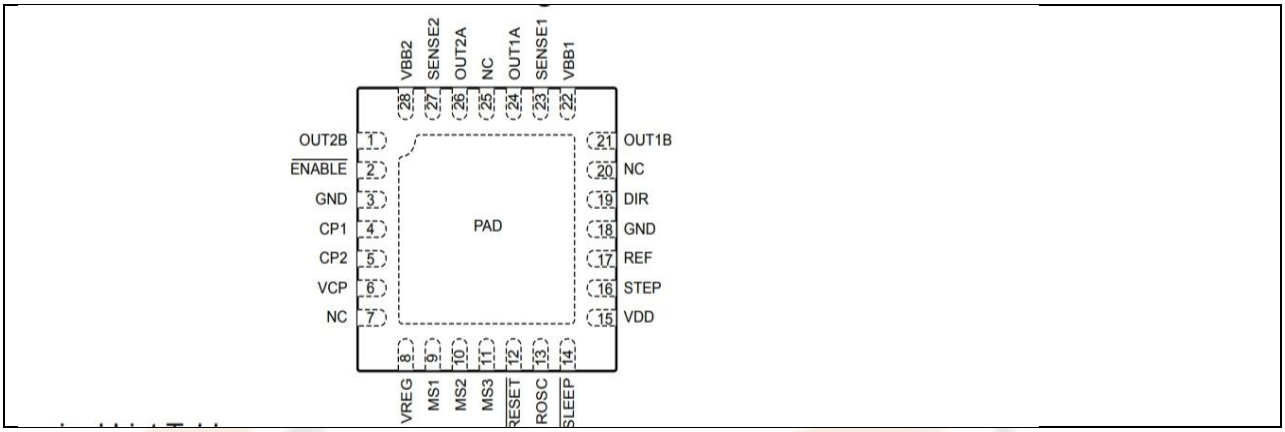


The schematic diagram of CNC Shield for A4988 is as shown above

- The A4988 is a complete microstepping motor driver with a built-in translator for easy operation with minimal control lines. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth, and sixteenth-step modes.
- The currents in each of the two output full-bridges and all of the N-channel DMOS FETs are regulated with fixed off-time PWM (pulse width modulated) control circuitry.
- At each step, the current for each full-bridge is set by the value of its external current-sense resistor (RS1 and RS2), a reference voltage (VREF), and the output voltage of its DAC (which in turn is controlled by the output of the translator).
- At power-on or reset, the translator sets the DACs and the phase current polarity to the initial Home state , and the current regulator to Mixed Decay Mode for both phases. When a step command signal occurs on the STEP input, the translator automatically sequences the DACs to the next level and current polarity.

- The microstep resolution is set by the combined effect of the MSx inputs. When stepping, if the new output levels of the DACs are lower than their previous output levels, then the decay mode for the active full-bridge is set to Mixed. If the new output levels of the DACs are higher than or equal to their previous levels, then the decay mode for the active full-bridge is set to slow.
- This automatic current decay selection improves microstepping perfor-mance by reducing the distortion of the current waveform that results from the back EMF of the motor.
- This stepper motor driver lets you control one bipolar stepper motor at up to 2 A output current per coil . Here are some of the driver's key features:
- Simple step and direction control interface
- Five different step resolutions: full-step, half-step, quarter-step, eighth-step, and sixteenth-step
- Adjustable current control lets you set the maximum current output with a potentiometer, which lets you use voltages above your stepper motor's rated voltage to achieve higher step rates
- Intelligent chopping control that automatically selects the correct current decay mode (fast decay or slow decay)
- Over-temperature thermal shutdown, under-voltage lockout, and crossover-current protection
- Short-to-ground and shorted-load protection

PIN FUNCTION:



Name	Number	Description
CP1	4	Charge pump capacitor terminal
CP2	5	Charge pump capacitor terminal
VCP	6	Reservoir capacitor terminal
VREG	8	Regulator decoupling terminal
MS1	9	Logic Input
MS2	10	Logic Input
MS3	11	Logic Input
Reset	12	Logic Input
ROSC	13	Timing Set
Sleep	14	Logic Input
VDD	15	Logic Supply
STEP	16	Logic input
REF	17	Gm reference voltage input
GND	3,18	Ground
DIR	19	Logic Input
OUT1B	21	DMOS Full Bridge 1 Output B
VBB1	22	Load supply
SENSE1	23	Sense resistor terminal for Bridge 1
OUT1A	24	DMOS Full Bridge 1 Output A
OUT2A	26	DMOS Full Bridge 2 Output A
SENSE2	27	Sense resistor terminal for Bridge 2
VBB2	28	Load Supply
OUT2B	1	DMOS Full Bridge 2 Output B
ENABLE	2	Logic Input
NC	7,20,25	No connection
PAD	-	Exposed pad for enhanced thermal dissipation*

- **Step Input (STEP):** A low-to-high transition on the STEP input sequences the translator and advances the motor one incre-ment. The translator controls the input to the DACs and the direction of current flow in each winding. The size of the increment is determined by the combined state of the MSx inputs.
- **Direction Input (DIR):** This determines the direction of rotation of the motor. Changes to this input do not take effect until the next STEP rising edge.
- **Internal PWM Current Control**: Each full-bridge is controlled by a fixed off-time PWM current control circuit that limits the load current to a desired value, ITRIP . Initially, a diagonal pair of source and sink FET outputs are enabled and current flows through the motor winding and the current sense resistor, RSx. When the voltage across RSx equals the DAC output voltage, the current sense comparator resets the PWM latch. The latch then turns off the appropriate source driver and initiates a fixed off time decay mode.
- **Fixed Off-Time:** The internal PWM current control circuitry uses a one-shot circuit to control the duration of time that the DMOS FETs remain off. The off-time, tOFF, is determined by the ROSC terminal.
- **Blanking:** This function blanks the output of the current sense comparators when the outputs are switched by the internal current control circuitry. The comparator outputs are blanked to prevent false overcurrent detection due to reverse recovery currents of the clamp diodes, and switching transients related to the capacitance of the load. The blank time, tBLANK (μs), is approximately

  tBLANK ≈ 1 μs
- **Shorted-Load and Short-to-Ground Protection:** If the motor leads are shorted together, or if one of the leads is shorted to ground, the driver will protect itself by sensing the overcurrent event and disabling the driver that is shorted, protect-ing the device from damage. In the case of a short-to-ground, the device will remain disabled (latched) until the SLEEP input goes high or VDD power is removed.
- **VREG (VREG):** This internally-generated voltage is used to operate the sink-side FET outputs. The nominal output voltage of the VREG terminal is 7 V. The VREG pin must be decoupled with a 0.22 μF ceramic capacitor to ground. VREG is internally monitored. In the case of a fault condition, the FET outputs of the A4988 are disabled. Capacitor

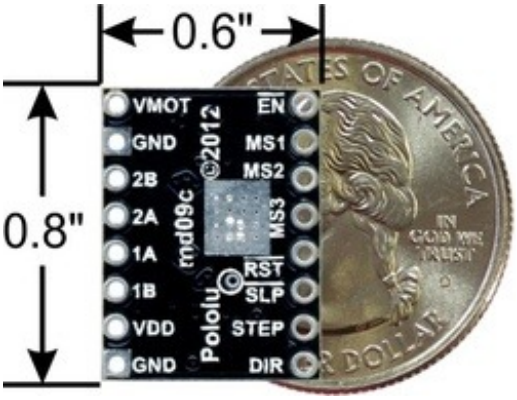
values should be Class 2 dielectric ±15% maximum, or tolerance R, according to EIA (Electronic Industries Alliance) specifications.

- **Enable Input:** This input turns on or off all of the FET outputs. When set to a logic high, the outputs are disabled. When set to a logic low, the internal control enables the outputs as required. The translator inputs STEP, DIR, and MSx, as well as the internal sequencing logic, all remain active, independent of the enable input state.

PACKAGE INCLUDE :

1 x CNC shield V3 engraving

A4988 Stepper Motor Driver Carrier, Black Edition



A4988 stepper motor driver carrier, Black Edition, bottom view with dimensions.

Overview

This product is a carrier board or breakout board for Allegro's A4988 DMOS Microstepping Driver with Translator and Overcurrent Protection; we therefore recommend careful reading of the A4988 datasheet (380k pdf) before using this product. This stepper motor driver lets you control one bipolar stepper motor at up to 2 A output current per coil (see the Power Dissipation Considerations section below for more information). Here are some of the driver's key features:

- **Simple step and direction control interface**
- **Five different step resolutions: full-step, half-step, quarter-step, eighth-step, and sixteenth-step**
- **Adjustable current control lets you set the maximum current output with a potentiometer, which lets you use voltages above your stepper motor's rated voltage to achieve higher step rates**
- **Intelligent chopping control that automatically selects the correct current decay mode (fast decay or slow decay)**
- **Over-temperature thermal shutdown, under-voltage lockout, and crossover-current protection**
- **Short-to-ground and shorted-load protection**
- **4-layer, 2 oz copper PCB for improved heat dissipation**
- **Exposed solderable ground pad below the driver IC on the bottom of the PCB**

This product ships with all surface-mount components—including the A4988 driver IC—installed as shown in the product picture.

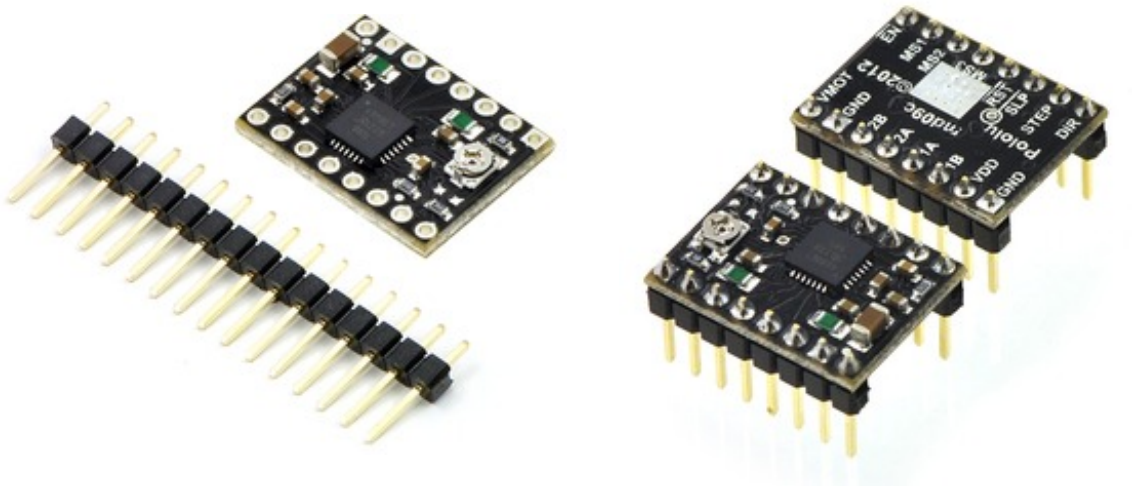
The Black Edition has the same component layout and pinout as our A4988 stepper motor driver carrier, so it can be used as a higher-performance drop-in replacement in applications designed for our original drivers. The Black Edition achieves its higher performance through its four-layer printed circuit board (PCB), which better draws heat out of the A4988 driver—while our original carrier can deliver up to approximately 1 A per phase in full-step mode without a heat sink or air flow, the Black Edition can deliver up to approximately 1.2 A under the same conditions.

For an even higher-performance alternative, please consider our DRV8825 stepper motor driver carrier, which can deliver more current over a wider voltage range. For lower-voltage applications, consider our DRV8834 carrier, which works with motor supply voltages as low as 2.5 V. Either of

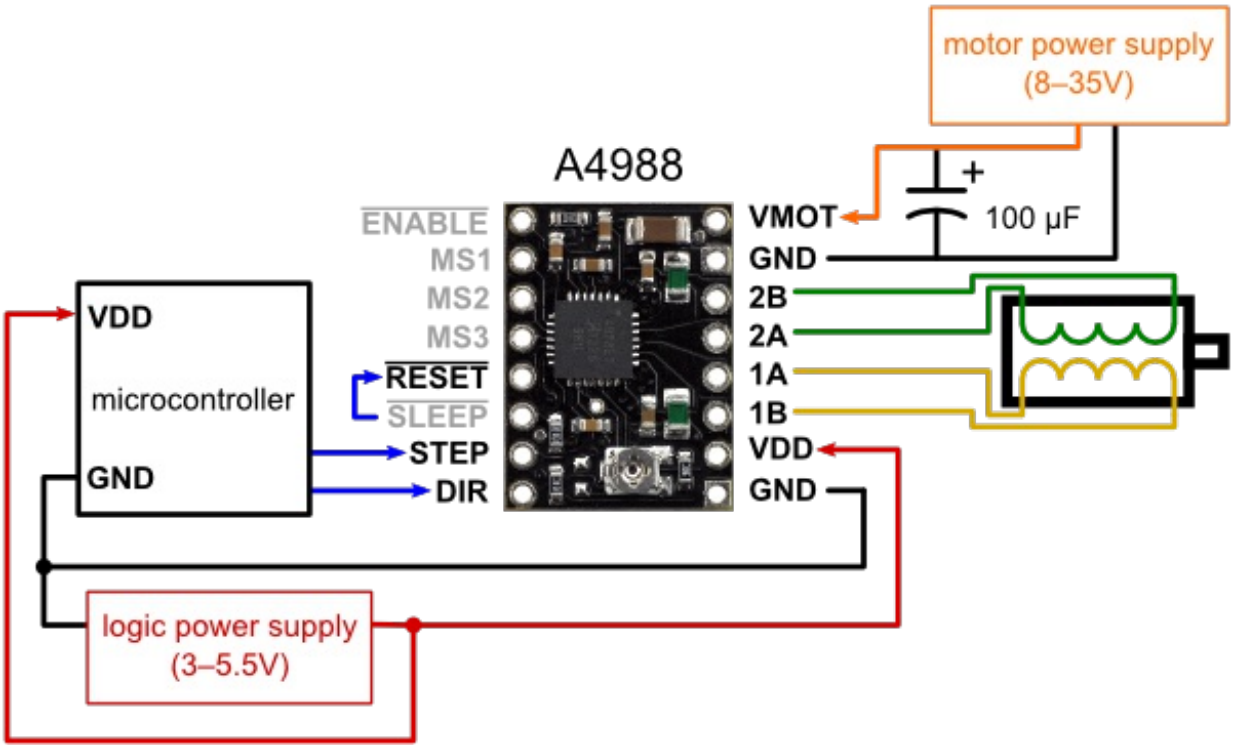
these boards can be used as a drop-in replacement for the Black Edition in many applications.

> Some unipolar stepper motors (e.g. those with six or eight leads) can be controlled by this driver as bipolar stepper motors. For more information, please see the frequently asked questions. Unipolar motors with five leads cannot be used with this driver.

Included hardware
The A4988 stepper motor driver carrier comes with one 1×16-pin breakaway 0.1" male header. The headers can be soldered in for use with solderless breadboards or 0.1" female connectors. You can also solder your motor leads and other connections directly to the board.



Using the driver



Minimal wiring diagram for connecting a microcontroller to an A4988 stepper motor driver carrier (full-step mode).

Power connections
The driver requires a logic supply voltage (3 – 5.5 V) to be connected across the VDD and GND pins and a motor supply voltage (8 – 35 V) to be connected across VMOT and GND. These supplies should have appropriate decoupling capacitors close to the board, and they should be

capable of delivering the expected currents (peaks up to 4 A for the motor supply).

> Warning: This carrier board uses low-ESR ceramic capacitors, which makes it susceptible to destructive LC voltage spikes, especially when using power leads longer than a few inches. Under the right conditions, these spikes can exceed the 35 V maximum voltage rating for the A4988 and permanently damage the board, even when the motor supply voltage is as low as 12 V. One way to protect the driver from such spikes is to put a large (at least 47 µF) electrolytic capacitor across motor power (VMOT) and ground somewhere close to the board.

Motor connections
Four, six, and eight-wire stepper motors can be driven by the A4988 if they are properly connected; a FAQ answer explains the proper wirings in detail.

> Warning: Connecting or disconnecting a stepper motor while the driver is powered can destroy the driver. (More generally, rewiring anything while it is powered is asking for trouble.)

Step (and microstep) size
Stepper motors typically have a step size specification (e.g. 1.8° or 200 steps per revolution), which applies to full steps. A microstepping driver such as the A4988 allows higher resolutions by allowing intermediate step locations, which are achieved by energizing the coils with intermediate current levels. For instance, driving a motor in quarter-step mode will give the 200-step-per-revolution motor 800 microsteps per revolution by using four different current levels.

The resolution (step size) selector inputs (MS1, MS2, and MS3) enable selection from the five step resolutions according to the table below. MS1 and MS3 have internal 100kΩ pull-down resistors and MS2 has an internal 50kΩ pull-down resistor, so leaving these three microstep selection pins disconnected results in full-step mode. For the microstep modes to function correctly, the current limit must be set low enough (see below) so that current limiting gets engaged. Otherwise, the intermediate current levels will not be correctly maintained, and the motor will skip microsteps.

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

Control inputs
Each pulse to the STEP input corresponds to one microstep of the stepper motor in the direction selected by the DIR pin. Note that the STEP and DIR pins are not pulled to any particular voltage internally, so you should not leave either of these pins floating in your application. If you just want rotation in a single direction, you can tie DIR directly to VCC or GND. The chip has three different inputs for controlling its many power states: **RST**, **SLP**, and **EN**. For details about these power states, see the datasheet. Please note that the **RST** pin is floating; if you are not using the pin, you can connect it to the adjacent **SLP** pin on the PCB to bring it high and enable the board.

Current limiting

To achieve high step rates, the motor supply is typically much higher than would be permissible without active current limiting. For instance, a typical stepper motor might have a maximum current rating of 1 A with a 5Ω coil resistance, which would indicate a maximum motor supply of 5 V. Using such a motor with 12 V would allow higher step rates, but the current must actively be limited to under 1 A to prevent damage to the motor.

The A4988 supports such active current limiting, and the trimmer potentiometer on the board can be used to set the current limit. One way to set the current limit is to put the driver into full-step mode and to measure the current running through a single motor coil without clocking the STEP input. The measured current will be 0.7 times the current limit (since both coils are always on and limited to 70% of the current limit setting in full-step mode). Please note that changing the logic voltage, Vdd, to a different value will change the current limit setting since the voltage on the “ref” pin is a function of Vdd.

Another way to set the current limit is to measure the voltage on the “ref” pin and to calculate the resulting current limit (the current sense resistors are 0.05Ω). The ref pin voltage is accessible on a via that is circled on the bottom silkscreen of the circuit board. The current limit relates to the reference voltage as follows:

Current Limit = VREF × 2.5

So, for example, if the reference voltage is 0.3 V, the current limit is 0.75 A. As mentioned above, in full step mode, the current through the coils is limited to 70% of the current limit, so to get a full-step coil current of 1.2 A, the current limit should be 1.2 A/0.7=1.7 A, which corresponds to a VREF of 1.7 A/2.5=0.68 V. See the A4988 datasheet for more information.

Note: The coil current can be very different from the power supply current, so you should not use the current measured at the power supply to set the current limit. The appropriate place to put your current meter is in series with one of your stepper motor coils.

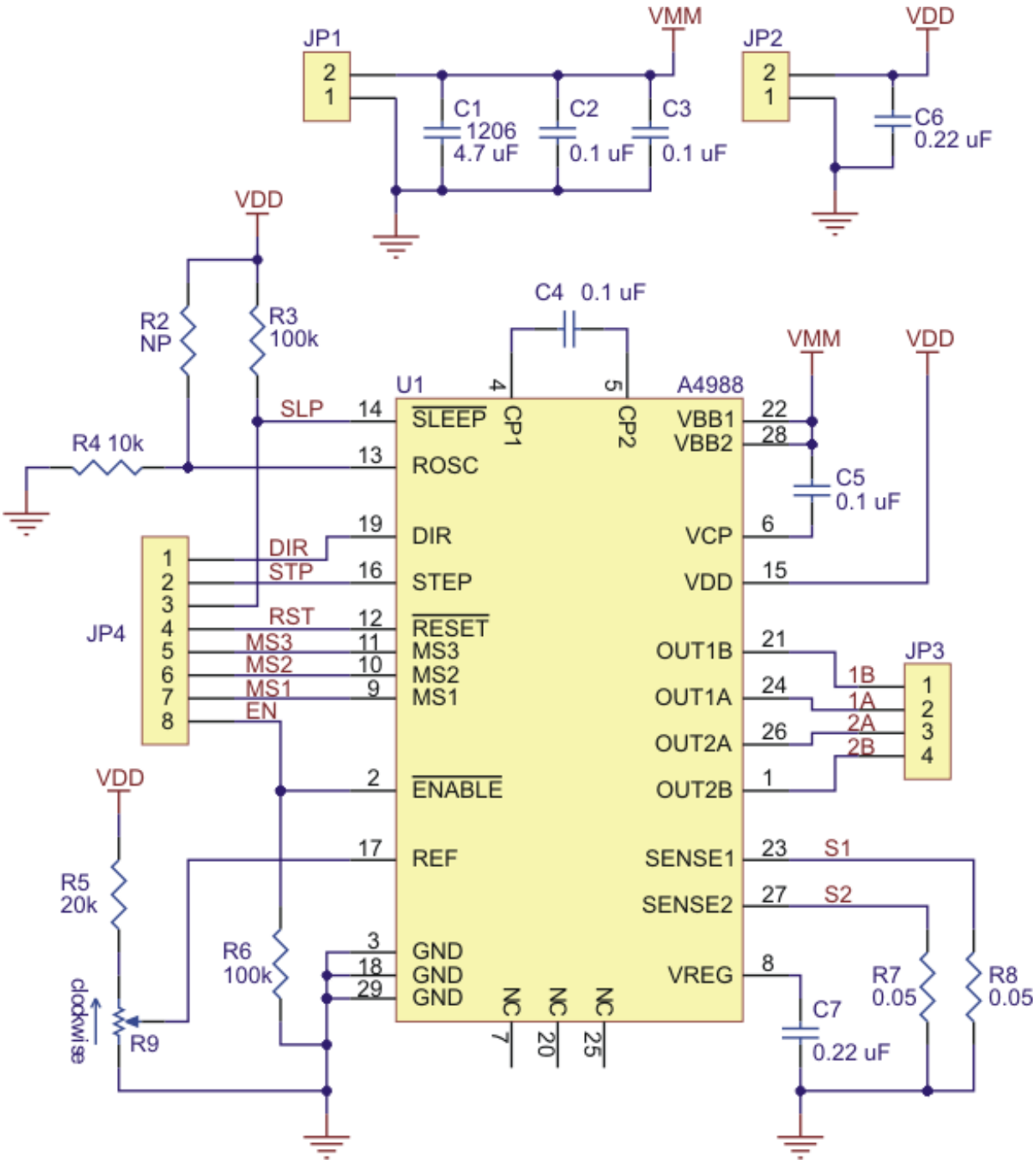
Power dissipation considerations

The A4988 driver IC has a maximum current rating of 2 A per coil, but the actual current you can deliver depends on how well you can keep the IC cool. The carrier’s printed circuit board is designed to draw heat out of the IC, but to supply more than approximately 1.2 A per coil, a heat sink or other cooling method is required (in our tests, we were able to deliver approximately 1.4 A per coil with air flow from a PC fan and no heat sink).

This product can get **hot** enough to burn you long before the chip overheats. Take care when handling this product and other components connected to it.

Please note that measuring the current draw at the power supply will generally not provide an accurate measure of the coil current. Since the input voltage to the driver can be significantly higher than the coil voltage, the measured current on the power supply can be quite a bit lower than the coil current (the driver and coil basically act like a switching step-down power supply). Also, if the supply voltage is very high compared to what the motor needs to achieve the set current, the duty cycle will be very low, which also leads to significant differences between average and RMS currents.

Schematic diagram



Schematic diagram of the md09b A4988 stepper motor driver carrier.

Note: This board is a drop-in replacement for our original A4988 stepper motor driver carrier.

Documentation on producer website.

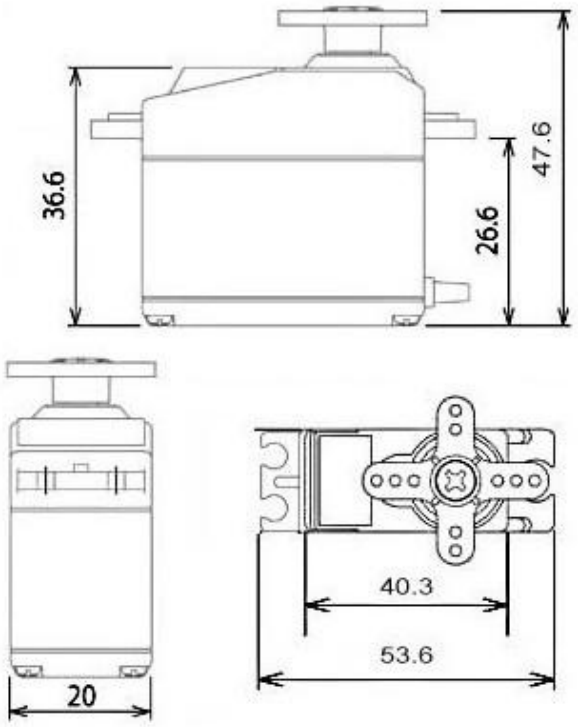
Datasheet Fuente Alimentación 12V 6A

Artículo No: Transformador LED
Voltaje de entrada: 1 o 2, 2 o 2
Voltaje de salida: DC12V
Corriente: 6A
Material: plástico
Temperatura de trabajo:-45 a 49 grados
Garantía: 1 año
Certificación: RoHS CE
Paquete: transformador LED de 1 pieza
Característica:

Esta fuente de alimentación convierte CA a CC, ampliamente utilizada en adaptadores de corriente, tiras de luces LED, pantallas LED, vallas publicitarias, equipos industriales, etc.
Cambie la fuente de alimentación, convierta CA 100-240V a CC 12V.
Amplio rango de voltaje de entrada, voltaje de salida constante y preciso.
Protección contra escasez, protección contra sobrecarga (105% ~ 200%), protección contra sobretensión (115% ~ 135%).
Ampliamente utilizado como adaptador de corriente para tiras Led, pantallas LED, vallas publicitarias, equipos industriales, etc. ¡SOLO PARA USO EN PUERTA!

Si necesita el cable de alimentación, haga clic en las siguientes imágenes para realizar el pedido.

MG996R High Torque
Metal Gear Dual Ball Bearing Servo



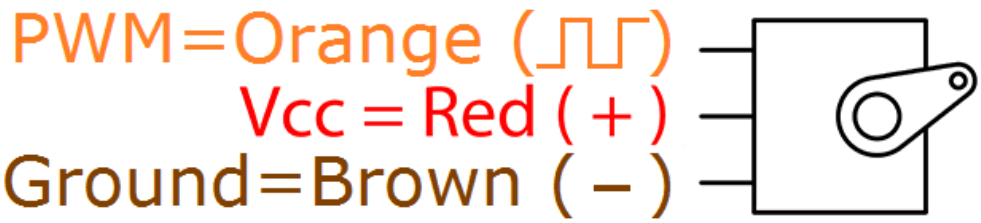
This High-Torque MG996R Digital Servo features metal gearing resulting in extra high 10kg stalling torque in a tiny package. The MG996R is essentially an upgraded version of the famous MG995 servo, and features upgraded shock-proofing and a redesigned PCB and IC control system that make it much more accurate than its predecessor. The gearing and motor have also been upgraded to improve dead bandwith and centering. The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

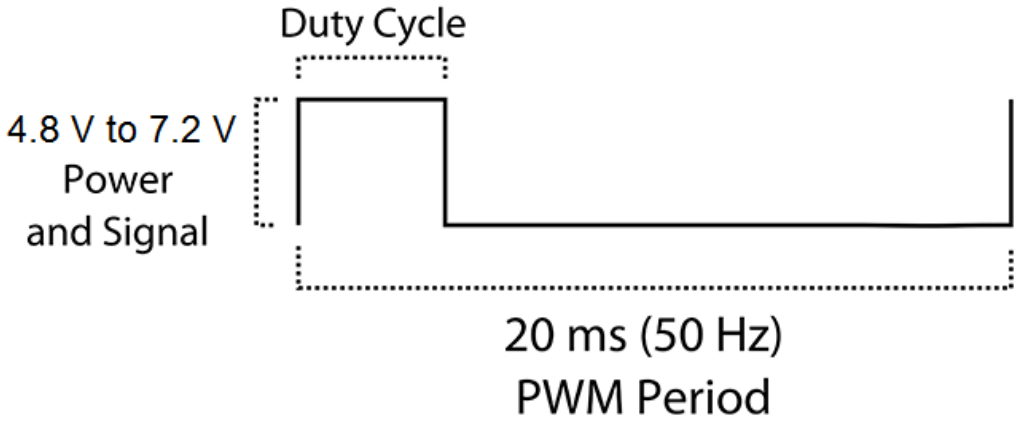
This high-torque standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG996R Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast!

Specifications

- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 9.4 kgf·cm (4.8 V), 11 kgf·cm (6 V)
- Operating speed: 0.17 s/60º (4.8 V), 0.14 s/60º (6 V)

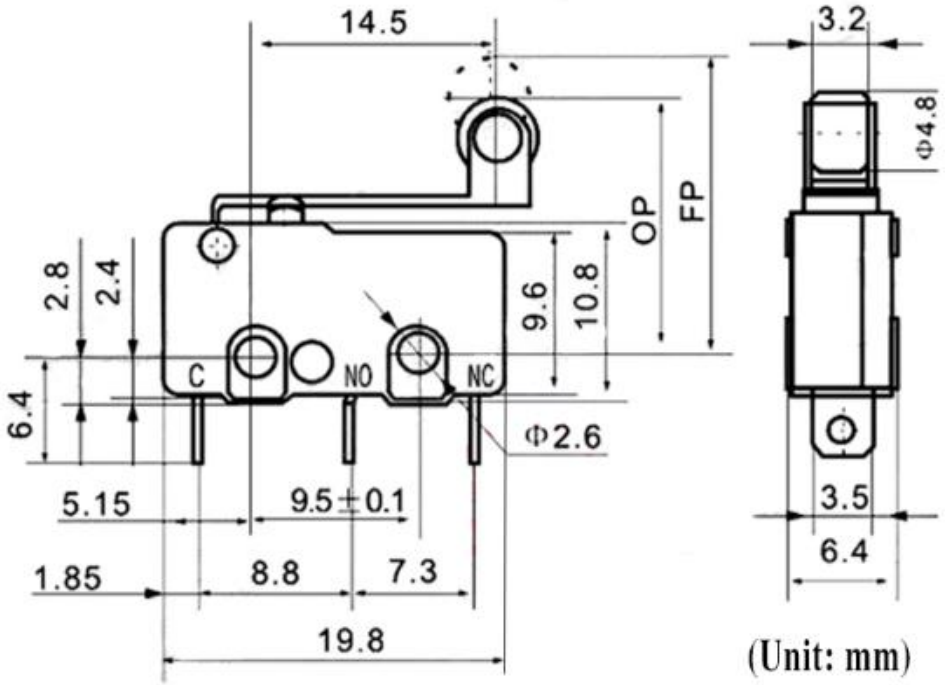
- Operating voltage: 4.8 V a 7.2 V
- Running Current 500 mA – 900 mA (6V)
- Stall Current 2.5 A (6V)
- Dead band width: 5 µs
- Stable and shock proof double ball bearing design
- Temperature range: 0 ºC – 55 ºC





Datasheet Final de carrera SS-5GL2

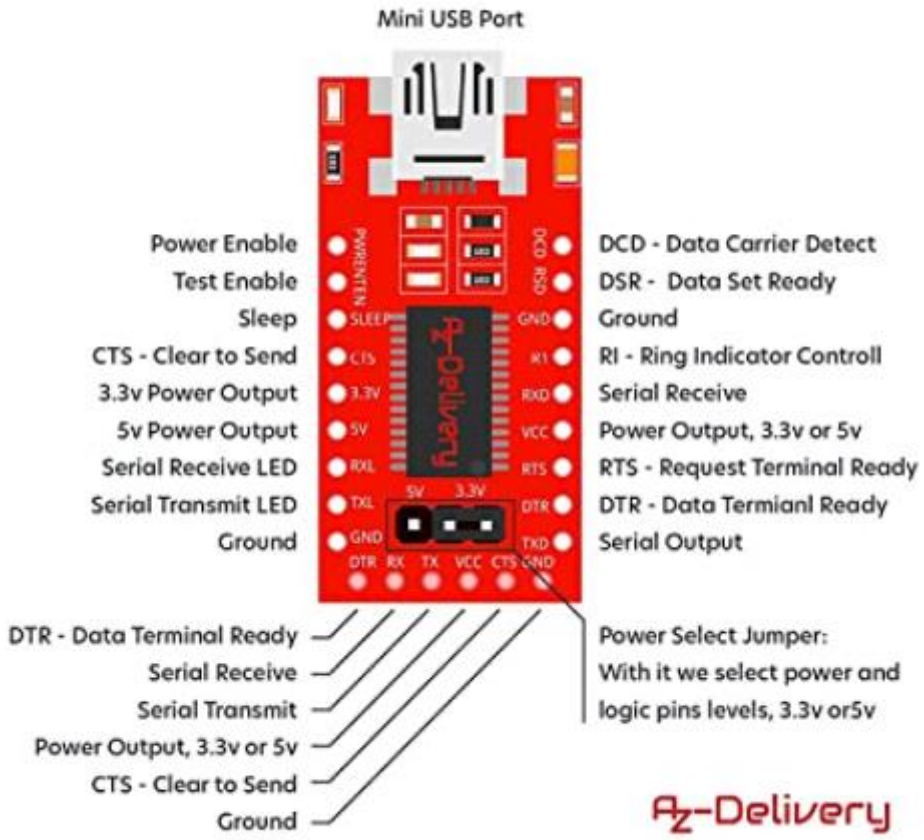
- Micropuerta de microondas/horno/interruptor de bloqueo N/O / N/C del refrigerador
- Con Límite de palanca de rodillo
- 5A CA 125V-250V
- Diamension: 20x10x5mm (L X W X H)
- Cantidad: 20 piezas



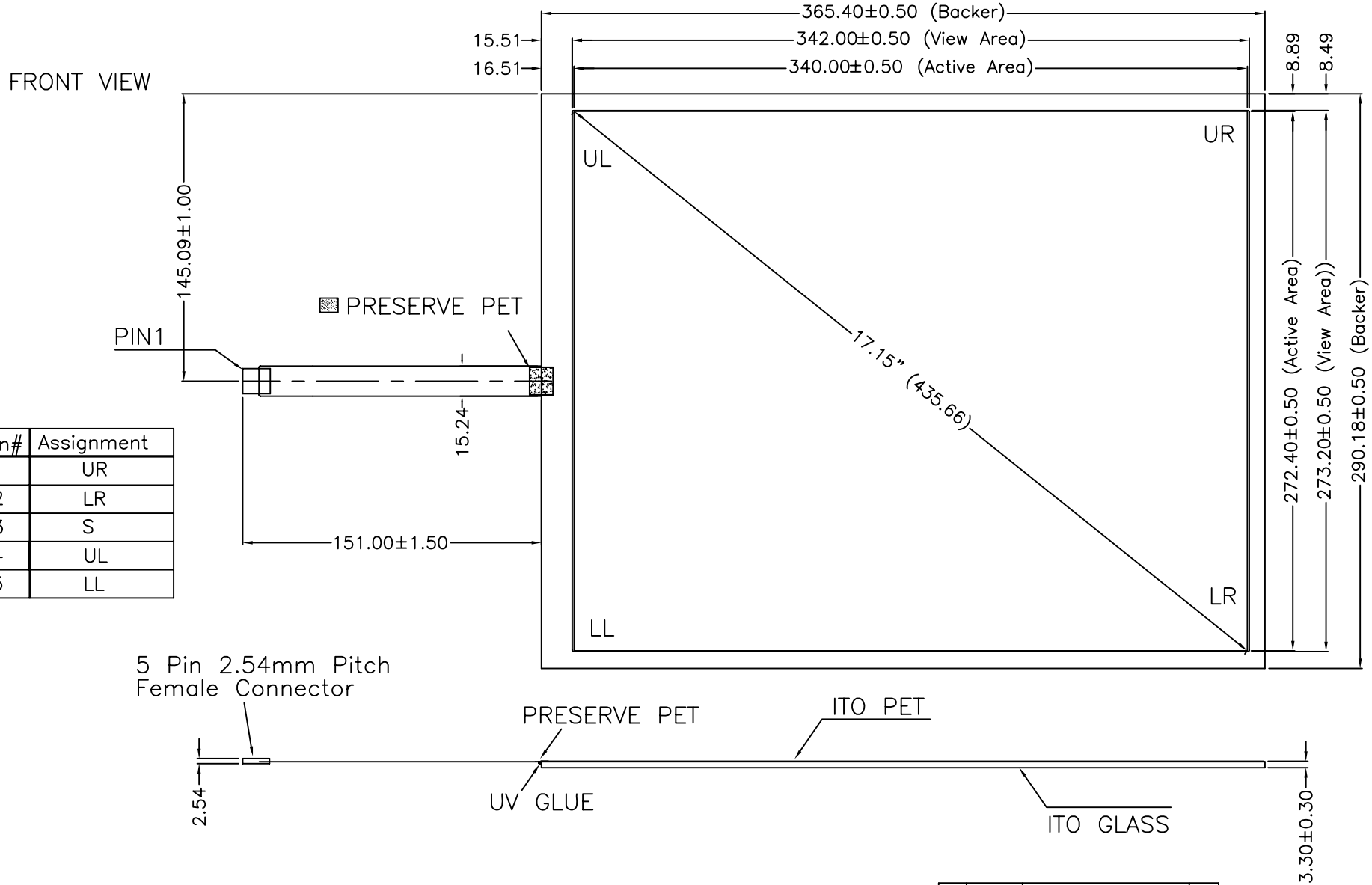
Weight:2.2g

Datasheet Módulo Comunicaciones FT232RL

Pinout



MODEL NAME : 2512	REV. A



Pin#	Assignment
1	UR
2	LR
3	S
4	UL
5	LL

NOTES:

1. ITO GLASS THICKNESS : 2.80mm
2. OVERALL THICKNESS : 3.30±0.30mm
3. CONNECTOR AND PINOUT AS INDICATED
4. FRONT SURFACE : ANTI-GLARE HARDCOAT
5. LAYER TO LAYER ASSEMBLY TOLERANCE: ±0.50mm
6. TAIL TYPE : IMMERSION GOLD PLATED FPC WITH AMP-C TIN PLATED CONTACTS
7. OTHER SPEC : SEE APPROVAL SHEET

RoHS COMPLIANT

NO.	DATE	REVISION	CHK
A	MAR.06,07	Pattern modified	Alex

CHIEF OF DESIGN		APPROVED	Alex
ENGINEER		PROJECT MANEGER	
DRAWN BY	WILLIAM	DATE	MAR. 06,2007
SHT 1 OF 1		REV.	

Apex Material Technology Corp.

TOLERANCES UNLESS SPECIFIED	
.X	± 0.50
.XX	± 0.50
.XXX	± 0.30
ANGULAR	.

MODEL NAME : 2512
DWG NO : 2512-1-01
SCALE: 1:1 UNIT: mm