UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO, IMPLEMENTACIÓN Y PROGRAMACIÓN DE UN SISTEMA ROBOTIZADO PARA TAREAS DE TELEOPERACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA

INDUSTRIAL Y AUTOMÁTICA

Presentado por:

GUILLERMO MÁRQUEZ RUIZ

Dirigido por:

LUIS IGNACIO GRACIA CALANDÍN

Valencia, 11 de junio de 2019

RESUMEN

En este proyecto se aborda la construcción y diseño de un prototipo de brazo articulado antropomórfico de tres ejes de rotación con una pinza como herramienta para realizar tareas básicas de *pick and place*.

Se utiliza un microcontrolador de la familia STM32F407 para el control de los motores paso a paso de las articulaciones junto con un módulo de comunicación Bluetooth HC-05 para enviar y recibir mensajes desde una aplicación compatible con cualquier teléfono móvil.

Todas las piezas necesarias para la construcción del prototipo se imprimen en 3D, por lo que se trata de un proyecto replicable y escalable fácilmente, además de ser económico.

Por último, se incluyen dos modos de funcionamiento: cinemática directa y cinemática inversa. Para este último se realizan los cálculos necesarios utilizando la trigonometría.

Palabras clave: robot, articulado, STM32F4, motor paso a paso, bluetooth impresión 3D

ABSTRACT

This Project will deal the construction and design of an anthropomorphic arm articulated of three rotation axis with a clamp as a tool to perform basic pick and place tasks.

It is used a microcontroller of the family STM32F407 for the stepper motors control of the articulations with the Bluetooth communication module HC-05 to send and receive messages from an app compatible with any smartphone.

All of the pieces needed for the prototype construction are 3D printed, so the project can be repeated and moved up easily, as well as being economic.

Lastly, there are two operation modes included: direct cinematic and inverse cinematic, in which the needed calculations will be resolved using trigonometry.

Key words: robot, articulated, STM32F4, stepper motor, Bluetooth, 3D printed

DOCUMENTOS DEL TFG

Contenido

Índice de figuras

Índice de tablas

Índice de ecuaciones

Glosario

- SLA: *Sterelithography Appratus*.
- SLS: *Selective Laser Sintering*
- FDM: *Fused Deposition Modeling*.
- PID: Controlador proporcional, integral y derivativo.
- PLA: Ácido Poliláctico
- ABS: Acrilonitrilo Butadieno Estireno
- CC: Corriente Continua
- USART: *Universal Synchronous Asynchronous Receiver-Transmitter*
- PWM: *Pulse Width Modulation*

DOCUMENTO 1:

MEMORIA DESCRIPTIVA

DISEÑO, IMPLEMENTACIÓN Y PROGRAMACIÓN DE UN SISTEMA ROBOTIZADO PARA TAREAS DE TELEOPERACIÓN

Presentado por:

GUILLERMO MÁRQUEZ RUIZ

Dirigido por:

LUIS IGNACIO GRACIA CALANDÍN

Valencia, 11 de junio de 2019

1. Introducción

1.1. Objetivo del trabajo

El objetivo de este trabajo es el diseño y la implementación de un prototipo de robot articulado antropomórfico con 3 grados de libertad controlado telemáticamente mediante una aplicación para el móvil. Se trata de un prototipo que pretende imitar a los robots industriales articulados e incorpora las funcionalidades propias de este tipo de robots para la realización de tareas sencillas como, por ejemplo, *pick and place*.

Por lo tanto, este proyecto se podría dividir en diversos objetivos:

- Diseño 3D de las partes mecánicas del brazo y posterior impresión en impresora 3D.
- Desarrollo del control de motores.
- Comunicación del robot con el dispositivo móvil mediante bluetooth.
- Desarrollo de una aplicación móvil como interfaz.
- Implementación de funcionalidades propias de un brazo articulado

1.2. Motivación

La motivación principal que ha llevado a la realización de este proyecto es la de demostrar los conocimientos adquiridos durante los años de carrera. Por lo tanto, era necesario un tipo de trabajo que agrupara la mayoría de los conocimientos de las diferentes disciplinas dentro de la electrónica y automática.

Gracias a la asignatura de Sistemas Robotizados impartida en el último curso, se ha podido comprobar que lo interesante que puede llegar a ser el control de un brazo articulado de estas características. Sin embargo, se han realizado proyectos meramente teóricos en entornos de simulación, con lo que el aprendizaje no queda completo a menos que se vea en un entorno práctico. Es por eso que, el desarrollo de un prototipo de brazo antropomórfico articulado asienta las bases de lo que puede llegar a ser un brazo de este calibre a tamaño real.

De esta forma, se pueden implementar los conocimientos adquiridos sobre control de robots en un prototipo real y aprender mediante el método de prueba – error, puesto que la mayoría de las veces lo que se aprende en la teoría dista bastante de lo que se ejecuta en la realidad.

1.3. Marco del proyecto y estado del arte

Dentro de este punto se comentan distintas áreas de la tecnología y su evolución a lo largo de la historia que presentan gran relación con el trabajo.

I. Microprocesadores

Los procesadores o microprocesadores son circuitos electrónicos capaces de realizar los cálculos aritméticos y ejecutar las lógicas necesarias para que sistema ejecute las operaciones que se han programado.

Estos surgieron a mediados del siglo XX, y con la invención del transistor y el avance de la computación, se comenzaron a realizar los primeros circuitos integrados.

Intel fue la primera compañía en fabricar un procesador que aunara los transistores con el circuito integrado fabricando el Intel 4004 en 1971. Este procesador era capaz de realizar 60.000 operaciones por segundo y se comenzó a usar en máquinas registradoras, calculadoras y semáforos.

Ilustración 1 - Procesador Intel 4004

Fue entonces cuando Gordon Moore propuso su ley, donde explicaba que cada dos años los avances en tecnología permitían que la cantidad de transistores colocados en un microprocesador se duplicase. Esto permitía que los microprocesadores fueran cada vez más potentes.

Actualmente, los microprocesadores se encuentran presentes en todos los aparatos que usamos día a día y pueden llegar a contener millones de transistores cada vez más pequeños de incluso 7 nanómetros [1], presentes hoy en día en los smartphones.

II. Impresión 3D y movimiento Maker

La impresión 3D es una técnica de inyección de plástico utilizada ampliamente en la industria para la realización de prototipos. Dentro de la impresión 3D hay infinidad de materiales disponibles y diferentes técnicas como SLA (*Sterelithography Appratus*), SLS (*Selective Laser Sintering*) y FDM (*Fused Deposition Modeling*).

De las técnicas anteriores, la más popularizada es la de FDM, en la que el material es depositado por capas, creando estructuras y formas para crear un objeto en 3D.

Está técnica ganó la popularidad hacia el 2004 cuando surgió el proyecto de código abierto RepRap, el cual pretendía proporcionar a todo el mundo la oportunidad de montar su propia impresora 3D. Se crearía una red de *Makers* para construir impresoras 3D a partir de piezas creadas por otras impresoras.

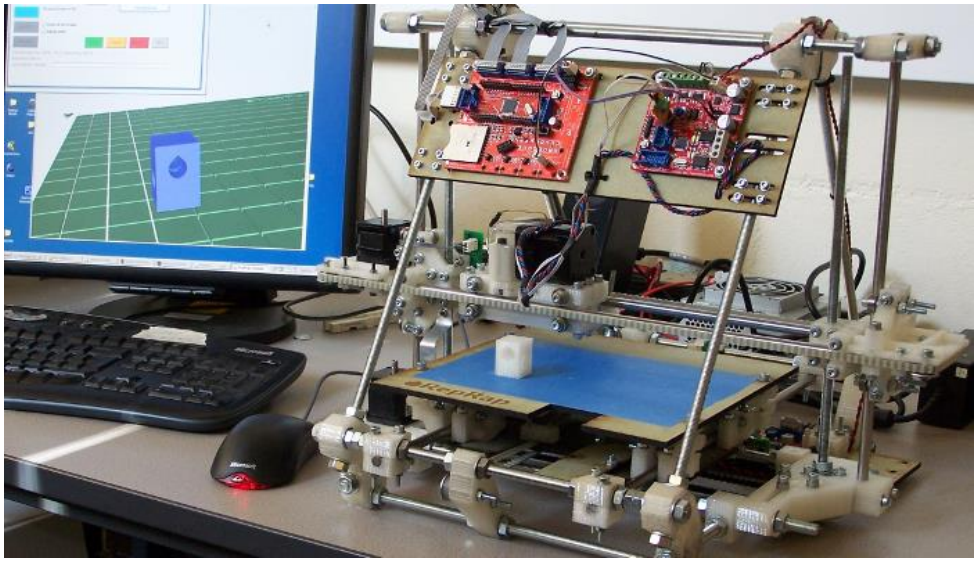


Ilustración 2 - Impresora 3D RepRap

A partir de este momento, se inician diferentes proyectos de Kickstarter que recaudan millones de dólares para crear impresoras de código abierto como Micro o Makerbot.

Esta última compañía es también la creadora de la biblioteca de archivos online *Thingiverse*, en la que cualquier persona puede compartir sus diseños entre toda la comunidad *Maker*.

Actualmente, se puede adquirir una impresora 3D por menos de 100€ por lo que es un gran aliciente para cualquier persona interesada en esta tecnología.

Los precios bajos y la gran popularidad de estas impresoras han creado una comunidad de personas que diseñan sus propios objetos y los hacen realidad gracias a la impresión 3D. Esto ha sido gracias al movimiento de "*Do it yourself*" y que crea una gran independencia de los productos del mercado con las amplias posibilidades que brinda esta tecnología.

III. Robots industriales: [2]

Desde antes que existiera la tecnología para construir este tipo de robots, los grandes visionarios como Nikola Tesla divagaban sobre máquinas que pudieran reemplazar al ser humano en diferentes tareas.

Pero no fue hasta 1954 cuando George Divol creó el primer robot industrial como tal que pudiera realizar distintas tareas. Este robot fue implantado en la cadena de montaje de la empresa General Motors y se convirtió en la primera cadena de producción automatizada de la historia.

A continuación, esta tecnología se fue expandiendo a diversas áreas como la industrial, militar, espacial o nuclear. Este auge es debido a que este tipo de robots sustituyen al ser humano en las tareas más peligrosas, además de ser más fiables, reducir costes y por la repetibilidad de estos procesos sin fallo alguno.

Actualmente, la mayor parte de la producción industrial está respaldada por este tipo de tecnología gracias a los grandes avances en cuanto a microprocesadores para el rápido cálculo de sistemas de control automático. Además, son robots versátiles que se pueden reprogramar para realizar distintas tareas dependiendo de la carga de trabajo.

Lo más potente en el mercado actual es el Fanuc M-2000iA [3], robot capaz de levantar cargas de hasta 2700 kg sin perder precisión en el movimiento. La ventaja que supone es la evitar costes de roturas de grúas puente, incrementar la velocidad de manipulación de las cargas sin perder repetibilidad y eliminar la necesidad de trabajos manuales peligrosos.

Por otra parte, se encuentran los denominados cobots (robot colaborativo), estos robots son capaces de trabajar en entornos con personas sin necesidad de jaulas de protección debido a que poseen una serie de sensores capaces de medir si existe contacto involuntario con operarios. A parte, pueden ser manipulados manualmente sin necesidad de ser programados anteriormente debido a que son capaces de aprender movimientos y luego ejecutarlos repetitivamente con gran precisión.

1.4. Metodología de trabajo

La metodología de trabajo es la forma en que se va a enfocar la realización de este trabajo en cuanto a la planificación y división de las tareas. Para empezar, se deben definir los objetivos como se han enumerado en el apartado 1.1 (Objetivo del trabajo).

Después de definir los objetivos se determinará la forma en que se desarrolla la solución mediante la siguiente estrategia:

Al ser un proyecto multidisciplinar que engloba aspectos de diversas áreas que finalmente tendrán que crear sinergias, la metodología que se utilizará será la **ingeniería concurrente**.

Esta metodología es popularmente usada en gran cantidad de empresas y su objetivo principal es la optimización de los recursos durante la realización de las distintas tareas de forma simultánea. Esto se alcanza coordinando equipos de trabajo de las distintas disciplinas para que entre todos se consiga un producto final adecuado y en el menor tiempo posible.

En el caso de ser una única persona para la realización del proyecto se enfocará de la siguiente manera: las tareas de las distintas áreas de trabajo se irán entrelazando a medida que avanza el proyecto, es decir, se realizará por ejemplo una tarea de diseño de una pieza al mismo tiempo que se programa el control del componente para que los fallos en el diseño sean menores y poder ahorrar en tiempos.

En la siguiente ilustración (Ilustración 3) se puede observar un esquemático de la metodología que se va a seguir durante la realización de este trabajo:

Ilustración 3 - Esquemático ingeniería concurrente

2. Factores a considerar: Necesidades y Limitaciones

2.1. Necesidades

Teniendo en cuenta los distintos objetivos del trabajo podemos traducirlo a necesidades a desarrollar:

- El brazo robótico deberá estar controlado mediante una interfaz para que su manejo sea sencillo y cualquier persona pueda utilizarlo.
- Lo principal de este proyecto es realizar un prototipo funcional y que se pueda observar cómo se mueve. Por lo tanto, los motores deben estar bien dimensionados para que al menos se pueda mover por sí solo y que las diferentes articulaciones tengan la fuerza suficiente para soportar como mínimo su propio peso.
- Los movimientos del robot deben ser controlados y que éste tenga cierta precisión para asemejarse a los robots industriales del mercado. Por lo tanto, el control debe ser lo más sencillo posible sin perder en precisión y repetibilidad.
- Diseño simple de las piezas 3D para poder ser modificadas más adelante.
- No es necesario un cálculo detallado de esfuerzos puesto que no es competencia de la carrera. Sin embargo, como se ha comentado antes, debe tener la fuerza suficiente para moverse por sí solo, por lo que las piezas mecánicas también deberán cumplir un mínimo de esfuerzo.

2.2. Limitaciones

- La principal limitación es económica, puesto que todos los componentes van a ser adquiridos por el alumno para su posterior uso y disfrute del prototipo. El presupuesto total de los materiales no debe superar los 300 euros.
- El área de impresión de la impresora 3D es de 220 mm x 220 mm x 250 mm, por lo que las piezas deben tener un tamaño menor que estas medidas.
- La segunda limitación es temporal, puesto que el proyecto debe estar acabado a principios del mes de junio de 2019 para poder ser entregado y defendido ante el tribunal.

3. Planteamiento de soluciones alternativas y justificación de la solución adoptada

3.1. Motores y control de estos

1) Soluciones alternativas y criterios de selección:

Para la elección de los motores que controlarán el movimiento de las articulaciones del brazo robótico se presentan los distintos tipos existentes en el mercado con sus respectivas características:

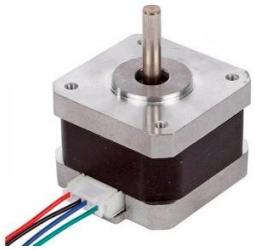
	Motores paso a paso	**Motores de C.C.**	**Motores sin escobillas**
Características			
Tipo de control	Sencillo (Contar pasos)	Control complejo (PIDs para cad motor)	Control por conmutación de transistores complejo
Necesidad de sensor	No	Sí	Sí
Precio	Barato	Alto	Muy alto
Precisión	No muy alta (1.8 grados)	Buena precisión	Mejor precisión
Rendimiento	Bajo por el alto consumo de corriente de mantenimiento	Mejor que el de motores de paso a paso	Mejor que el de motores de C.C.
Par	Alto	Medio	Alto
Velocidad	Baja	Alta	Muy alta

Tabla 1 - Comparativa de elección de motores

2) Justificación de la solución adoptada:

En la tabla, se puede observar que el motor que más de ajusta a nuestro proyecto es el **motor paso a paso** puesto que cumple con dos de nuestras necesidades/limitaciones principales: control sencillo y precio económico. Además, este tipo de motores poseen un buen par a velocidades bajas y el brazo robótico no necesita ser rápido, sino fiable y tener suficiente fuerza.

Los otros dos tipos de motores son descartados por las mismas razones de precio y complejidad de control.

En cuanto a la precisión en los motores de paso a paso, se puede mejorar mediante el uso de drivers de control llegando a precisiones de $1.8 \cdot \frac{1}{32} = 0.05625$ grados.

3) Elección de drivers de control para los motores paso a paso:

Este tipo de motores es muy utilizado en las máquinas de impresión 3D puesto que, combinados en los 3 ejes se puede conseguir una gran precisión y crear figuras de gran detalle. Por este motivo, utilizaremos un driver muy popular en esta industria el cual es el **DRV8825 de la marca Pololu**.

Características principales:

	DRV8825 Pololu
Características	
Resolución	Seis diferentes resoluciones: full-step, half-step, 1/4-step, 1/8-step, 1/16-step, 1/32-step.
Corriente máxima adjustable	2,2 A
Voltaje máximo de alimentación	45 V
Arquitectura interna	Puede interactuar con sistemas de 3,3V y 5V.
Control	Tren de pulsos

Tabla 2 - Características driver DRV8825

3.2. Microprocesador de control del robot

1) Soluciones alternativas y criterios de selección:

Para el procesamiento de las funcionalidades del robot y el control del mismo utilizaremos un microcontrolador que será programado según las necesidades de las especificaciones.

Durante el grado se han visto varias clases de microcontroladores que serán resumidas en dos grandes familias por lo económico de estas placas y la potencia que tienen. En la siguiente tabla (Tabla 3) se enumeran los aspectos de estas dos placas y se elige la más acorde al proyecto. Datos extraídos de los datasheets de estos productos.

	Arduino UNO Rev3	**STM32F4 Discovery**	**Raspberry**
Características			
Microcontrolador	ATmega328	ARM Cortex M4	Quad-Core Cortex A7 a 900MHZ
Voltaje operativo	5 V	3 V - 5 V	1.8 V - 3.3 V
Pines de Entradas/Salidas Digitales	14 (6 salidas PWM)	Up to 140 I/O ports with interrupt capability: – Up to 136 fast I/Os up to 84 MHz – Up to 138 5 V-tolerant I/Os	40 GPIO pin
Pines de Entradas/Salidas Analógicas	6	Cualquier puerto es configurable como analógico	Cualquier puerto es configurable como analógico
Velocidad de reloj	16 MHz	168 MHz	900 MHz
Memoria flash	32 Kb	1 Mb	RAM: 1GB DDR2
Software de desarrollo	Arduino	Keil, Mbed, Eclipse…	NOOBS, Raspbian…
Precio	20 €	18 €	37 €
Comunidad y soporte técnico	Amplia y con muchos tutoriales online	Comunidad menos amplia pero con muchos documentos oficiales disponibles	Amplia comunidad con muchos ejemplos de proyectos
Lenguaje de programación	Arduino basado en C	C/C++	Python
Complejidad de programación	Baja	Media	Alta

Tabla 3 - Comparativa de microcontroladores

Viendo todas las características expuestas y comparadas se puede afirmar que la placa Raspberry tiene mucha más potencia de reloj, pero la Discovery tiene muchos más pines I/O. Sin embargo, no se necesita mucha velocidad de reloj para un proyecto de este calibre por lo que cualquiera de las placas sería apta para el proyecto. Por tanto, la decisión recae en la habilidad del programador de manejarse con estos microcontroladores.

2) Justificación de la solución adoptada:

Finalmente, **se elige la placa STM32 Discovery** debido a varios motivos importantes a recalcar:

- **Cantidad de pines**: la Discovery dispone de 140 pines que pueden ser configurados de cualquier manera por lo que es improbable que falten pines, cosa que con Arduino podría darse el caso.
- **Velocidad del reloj**: la capacidad de cálculo de esta placa es 10 veces mayor a la de Arduino. Por eso se elige la Discovery, por la necesidad de cálculos de matrices y cinemáticas inversas.
- **Interrupciones**: son funciones que son capaces de parar el programa y ejecutar una parte específica, por lo que son muy útiles si se utilizan correctamente.
- **Habilidad de programación**: durante la carrera se ha programado con esta placa en diversas ocasiones y la habilidad es mayor pese a ser más complicada que la de Arduino.

3.3. Software de diseño 3D

1) Soluciones alternativas y criterios de selección:

De las licencias disponibles por la UPV se dispone de dos programas principales para el diseño de piezas en 3D. La elección se realizará en cuanto a complejidad y cantidad de herramientas para el diseño 3D mostradas en la siguiente tabla (Tabla 4).

	Fusion 360	**AutoCAD Inventor**
Características		
Complejidad	Baja	Media
Cantidad de herramientas	Alta	Media
Habilidad del diseñador	La mayoría de los diseños anteriores han sido realizados con este software	Pocos diseños realizados con este programa

Tabla 4 - Comparativa software diseño 3D

2) Justificación de la solución adoptada:

Según la comparativa, lo mejor será utilizar el software Fusion 360 puesto que cumple con todos los parámetros que se necesitan: baja complejidad, gran cantidad de herramientas de diseño y ya existe cierta habilidad con el manejo de este software.

3.4. Material piezas de impresión 3D

Para la impresión 3D, existe una gran cantidad de materiales con los que imprimir, proporcionando diferentes características mecánicas y de acabado. En este caso, se hará una comparativa en la siguiente tabla (Tabla 5) de los materiales en los que puede imprimir la impresora de que se dispone.

1) Soluciones alternativas y criterios de selección:

	PLA	**ABS**	**TPE, TPU, TPC**	**Nailon**
Características				
Dureza	Alta	Alta	Media	Alta
Flexibilidad	Baja	Media	Muy alta	Alta
Durabilidad	Media	Alta	Muy alta	Alta
Temperatura de impresión	180 ºC – 230 ºC	210 ºC – 250 ºC	210 ºC – 230 ºC	240 ºC – 260 ºC
Temperatura cama de impresión	20 ºC – 60 ºC (aunque no necesaria)	80 ºC – 110 ºC	30 ºC – 60 ºC	70 ºC – 100 ºC
Contracción/ deformación	Mínima	Considerable	Mínima	Considerable
Tipo de material	Termoplástico biodegradable.	Acrilonitrilo butadieno estireno	Elastómeros termoplásticos	Polímero sintético
Emisión gases durante impresión	No emite gases	Emite fuertes humos, necesidad de ventilación	No emite gases	No emite gases
Dificultad de uso	Baja	Media	Media	Media
Precio de 1 kg	19 €	19 €	27 €	29 €

Tabla 5 - Comparativa materiales impresión 3D

2) Justificación de la solución adoptada:

Primeramente, de los cuatro materiales comparados, se descarta el ABS debido a que emite gases nocivos durante la impresión y no se dispone de una zona ventilada donde situar la impresora.

La familia TPE, TPU, TPC también se descarta debido a que es un material flexible y en el brazo robótico necesitamos robustez en las articulaciones, por lo que un material que se deforme al aplicarle esfuerzos es contraproducente.

Por último, entre el nailon y el PLA elegimos el PLA por los siguientes motivos:

- **Temperatura de impresión**: el PLA necesita menos temperatura durante la impresión, lo que se traduce en un menor consumo energético de la impresora.
- **Precio**: el precio del PLA es el más económico de todos.
- **Tipo de material**: se trata del material más ecológico de los seleccionados puesto que está derivado de recursos renovables como el almidón de maíz o la caña de azúcar, además de ser biodegradable.

3.5. Software para la creación de la app

Puesto que no se dispone de conocimientos de programación en Android y no se debe dedicar gran esfuerzo en la búsqueda de un portal de creación de apps. Ante la gran cantidad de opciones, se usará el software **MIT App Inventor** por las ventajas que ofrece:

- Este software usa el lenguaje de programación Scratch, en el cual mediante el uso de bloques predefinidos se puede crear la aplicación acorde a las necesidades.
- No necesita instalar ningún software en el ordenador, todo se realiza online.
- Se puede probar la aplicación en tiempo real mientras se programa o descargar un archivo *.apk*.

3.6. Resumen solución adoptada

En la siguiente tabla (Tabla 6), se muestra un resumen de los componentes escogidos en los anteriores apartados:

Aspecto	Elección
Movimiento articulaciones	Motores paso a paso y *driver* DRV8825
Microprocesador de control del robot	STM32F4 Discovery
Software de diseño 3D	Autodesk Fusion 360
Material piezas de impresión 3D	PLA
Software creación app	MIT App Inventor

Tabla 6 - Resumen soluciones adopatadas

4. Desarrollo del proyecto

Como ya se ha visto anteriormente en el apartado 1.2, el proyecto se divide en cuatro áreas principales que a lo largo de este punto se irán desarrollando por separado. Sin embargo, en la realidad muchas de estas tareas han sido realizadas simultáneamente utilizando la metodología de ingeniería concurrente. Estas tareas simultáneas se explican para un mejor seguimiento del documento.

4.1. Articulaciones robot

En este primer apartado se explicará cómo se ha desarrollado el movimiento del robot mediante los motores elegidos en el apartado 3.1. Los pasos que más adelante se detallan para el desarrollo de esta parte son los siguientes: realización de los cálculos necesarios para el dimensionamiento de los motores, selección de componentes necesarios, configuración de los drivers y conexiones, y programación del control de giro.

4.1.1. Tipo de robot

El tipo de robot seleccionado para este proyecto es un robot articulado antropomórfico, que imita el movimiento de un brazo humano por la forma de sus articulaciones. Este tipo de robots se componen de elementos unidos por articulaciones de revolución, en concreto para este proyecto con 3 articulaciones como el que se muestra en la figura siguiente:

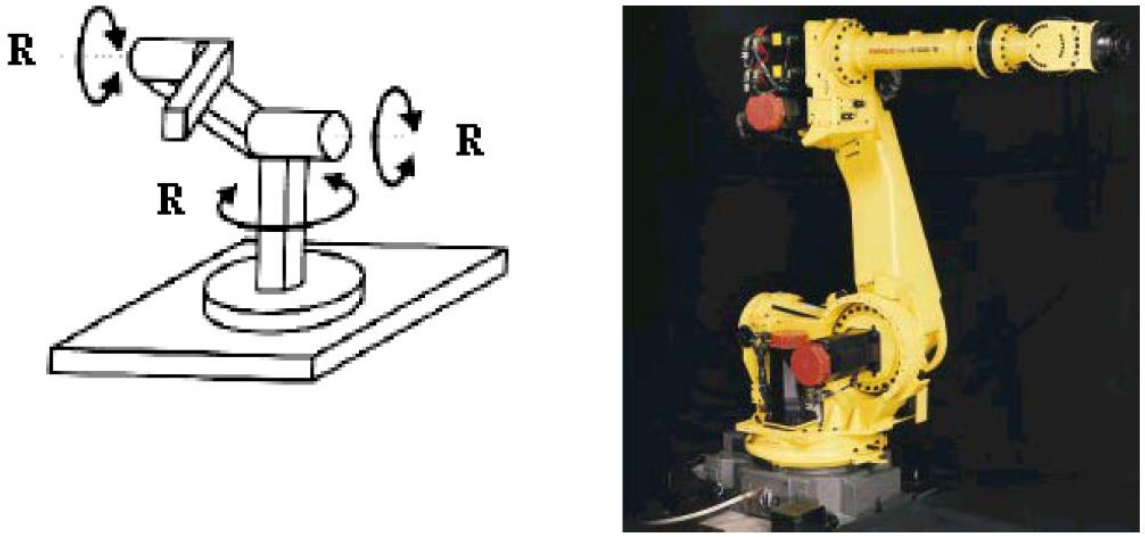


Ilustración 4 - Robot antropomórfico

4.1.2. Selección motores

Como ya se ha visto en el apartado 2, la necesidad de este proyecto es crear un prototipo funcional que pueda moverse sin problema, por lo que una buena selección de los motores es

clave para poder soportar el peso del robot. Por este mismo motivo, la **característica clave** a la hora de seleccionar los motores paso a paso es el **par máximo** que pueden soportar.

Para proyectos de este calibre se dispone de diferentes modelos según el par que pueden soportar, por lo que para cada articulación se selecciona un tipo de motor.

4.1.2.1. Articulación 1

Esta articulación hace de base y es sobre la que se asienta todo el peso del robot, aunque no es necesario tener un gran par debido a que sólo debe vencer la fuerza de rozamiento. Para minimizar esta fuerza de rozamiento, el robot se apoya sobre un rodamiento de una rueda para muebles que se reutiliza de un mueble antiguo. En la siguiente imagen se puede observar una rueda original y cómo se ha modificado para conseguir una base para el robot que no ofrezca apenas rozamiento.

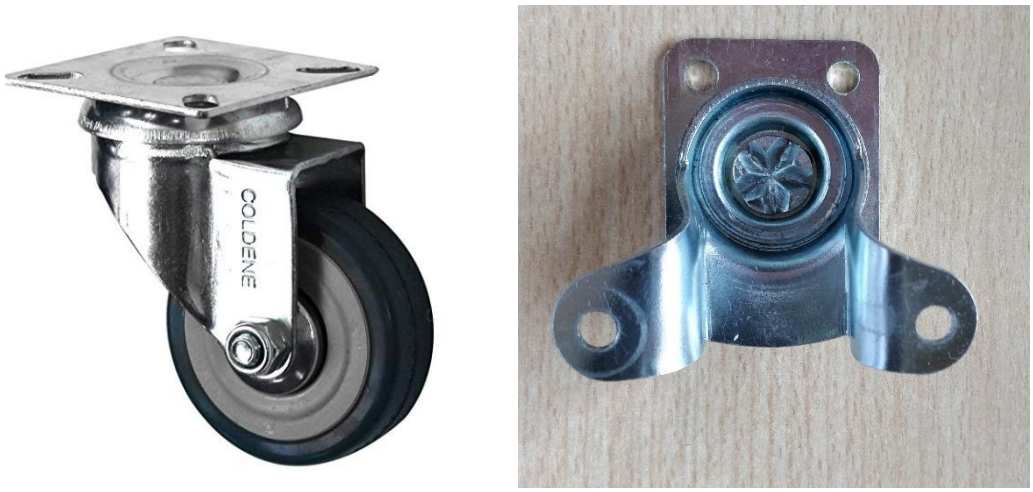


Ilustración 5 - Rodamiento base articulación 1

Las modificaciones realizadas a la rueda se observan en la imagen de la derecha, en las que se ha retirado la rueda y se han doblado las patillas que la sujetaban hasta formar un ángulo de 90 º.

Al utilizar esta base, se necesita un engranaje con una correa de distribución para transmitir el giro desde la posición del motor (alejado del eje de rotación) al robot. Este engranaje de 64 dientes ha sido descargado de la plataforma *Thingiverse* y se ha impreso en 3D (en el apartado 4.2 se detallan las piezas impresas).

Esta transmisión se completa con una correa de tipo GT2 de 6 mm que encaje con el engranaje y otro engranaje que se atornilla al eje del motor. En la siguiente imagen (Ilustración 6) se pueden observar la correa y el engranaje.

Ilustración 6 - Correa GT2 y engranaje de 20 dientes

Esta relación de engranajes más grande en el objeto a mover que en el motor, provoca que la velocidad de giro del engranaje de 64 dientes sea menor que en el engranaje de 20 dientes, por lo que la fuerza que tenga que ejercer el motor también será menor.

A continuación, se calcula la relación de transmisión entre los engranajes para más tarde poder controlar el ángulo de giro del robot. Se utiliza la fórmula que relaciona el número de dientes y el número de revoluciones de cada engranaje de la siguiente manera:

Ecuación 1 - Relación engranajes

$$n_1 \cdot Z_1 = n_2 \cdot Z_2$$

Donde:

n_1: número de vueltas del engranaje 1

Z_1: número de dientes del engranaje 1

n_2: número de vueltas del engranaje 2

Z_2: número de dientes del engranaje 2

Por lo que si el engranaje 1 (el motor) realiza una vuelta, se puede calcular el número de vueltas que realiza el segundo engranaje (el robot) para obtener la relación de transmisión que más adelante se usará en el control de giro de la primera articulación. El resultado será:

Ecuación 2 - Numero de vueltas según relación de engranajes

$$n_2 = \frac{Z_1 \cdot n_1}{Z_2} = \frac{20 \cdot 1}{64} = 0.3125 \text{ vueltas}$$

Finalmente, el motor elegido para esta articulación por la necesidad de bajo par es el *Nema 17 Bipolar 59Ncm (84oz.in) 2A* cuyo datasheet se adjunta en los anexos y que podemos observar en la siguiente imagen (Ilustración 7):

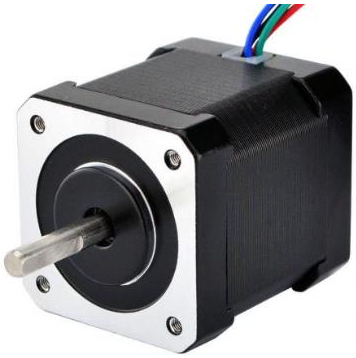


Ilustración 7 - Nema 17 stepper motor

4.1.2.2. Articulación 2

Esta articulación es la que más fuerza del robot debe ejercer puesto que debe soportar todo el peso del brazo extendido y sujetando un objeto en el caso más desfavorable. Por lo tanto, como todavía no se sabe con exactitud el peso que debe soportar este eje, se estima el par máximo que debe soportar y se busca un motor acorde a esta estimación.

Teniendo en cuenta el peso del motor anterior como referencia para la siguiente articulación sumado al peso de la herramienta se obtiene el par que debe soportar mediante la siguiente ecuación:

Ecuación 3 - Fórmula par calculado

$$Par\ (N \cdot m) = Peso_1 \cdot D_1 + Peso_2 \cdot D_2$$

Siendo

D_i: distancia del peso al centro de la articulación

Ecuación 4 - Peso del motor 1

$$Peso_1(N) = Motor_1\ (Kg) \cdot 9.8 = 0.5 \cdot 9.8 = \ 4.9\ N$$

Ecuación 5 - Peso de la herramienta y objeto

$$Peso_2(N) = \big(Herramienta\ (Kg) + Objeto\ (Kg)\big) \cdot 9.8 =$$

$$= (0.2 + 0.2) \cdot 9.8 = \ 3.92\ N$$

Por lo tanto

Ecuación 6 - Par calculado que debe soportar el motor 1

$$Par\ (N \cdot m) = 4.9 \cdot 0.15 + 3.92 \cdot 0.3 = 1.91\ N \cdot m$$

Teniendo el par estimado, se puede seleccionar el motor adecuado a esta articulación, siendo en este caso el *Nema 24 Stepper Motor 4.0Nm(566oz.in)*, el cual tiene mayor par que el calculado y que se puede observar en la siguiente imagen:

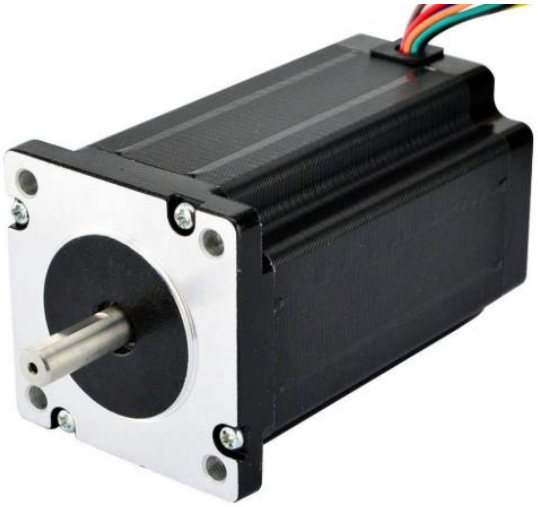


Ilustración 8 - Nema 24 stepper motor

4.1.2.3. Articulación 3

Para la selección de este motor, se usan los cálculos realizados en la anterior articulación pero sólo para el peso 2:

Ecuación 7 - Par calculado para motor 3

$$Par\ (N \cdot m) = Peso_2 \cdot D_2 = 3.92 \cdot 0.12 = \ \ 0.47\ N \cdot m$$

Por lo tanto, el motor seleccionado es el mismo que para la articulación 1, el *Nema 17 Bipolar 59Ncm (84oz.in) 2ª*.

4.1.3. Conexiones y configuración de los drivers

Como ya se ha comentado en el apartado 3.1 (Motores y control de estos), el driver que se va a utilizar es el DRV8825, en concreto el de la marca Pololu. El datasheet correspondiente a este driver se adjunta en el apartado de Anexo VI: Datasheet driver DRV8825. Del mismo datasheet se extrae el siguiente esquemático que determina las conexiones necesarias para controlar un motor paso a paso.

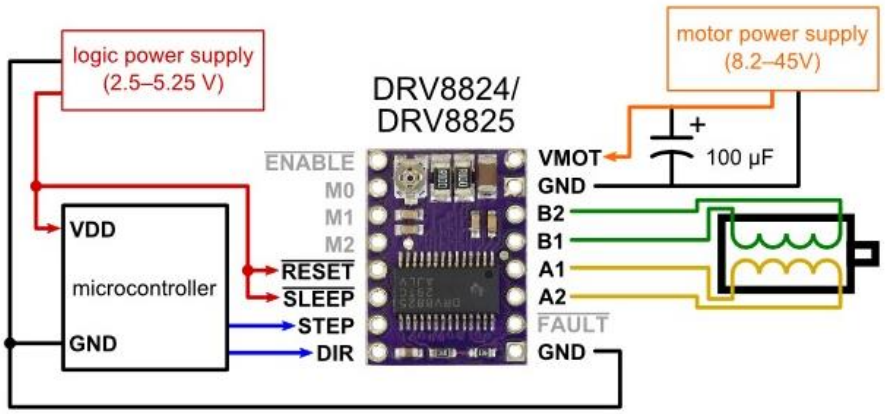


Ilustración 9 - Conexiones DRV8825

- ENABLE: Pin para habilitar el funcionamiento del driver. Si no se conecta nada, siempre estará en funcionamiento.
- M0, M1 y M2: Sirve para determinar mediante una tabla, que se proporciona en el datasheet, la resolución del ángulo del motor. La tabla (Tabla 7) es la siguiente:

MODE0	MODE1	MODE2	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	1/4 step
High	High	Low	1/8 step
Low	Low	High	1/16 step
High	Low	High	1/32 step
Low	High	High	1/32 step
High	High	High	1/32 step

Tabla 7 - Modos de resolución DRV8825

En este caso, se ha elegido una resolución de 1/16 puesto que buscar la mayor resolución en el giro de los motores no es un objetivo a seguir en este proyecto.

- RESET y SLEEP: estos dos pines deben estar a 1 para que el driver funcione correctamente.
- STEP: por este pin se envía el tren de pulsos encargado de provocar el giro del motor paso a paso.
- DIR: pin encargado de determinar la dirección de giro. En las pruebas de funcionamiento se determinará la lógica de este pin.

- VMOT: pin conectado al positivo de la fuente de alimentación cuyos valores deben estar entre 8.2 y 45 V.
- GND: pin conectado a la masa de la fuente de alimentación.
- B2, B1, A1 y A2: pines de conexión al motor que más adelante se detalla para cada articulación.
- FAULT: pin que debe estar desconectado para no provocar un modo de fallo en el driver.
- GND: pin conectado a la masa del microcontrolador.

Sin embargo, antes de realizar todas las conexiones se debe configurar el driver en lo que refiere a la limitación de corriente máxima y a la división de pasos. La configuración varía dependiendo del motor a controlar.

Seleccionar una limitación de corriente adecuada es importante porque una corriente mayor puede provocar el sobrecalentamiento y el consiguiente mayor desgaste del motor.

Para realizar la limitación de corriente, se conectan todos los pines según el esquemático de la Ilustración 9 excepto los referentes al motor paso a paso.

A continuación, se mide con un multímetro el voltaje entre GND y el potenciómetro del driver como se puede observar en la siguiente imagen (Ilustración 10):

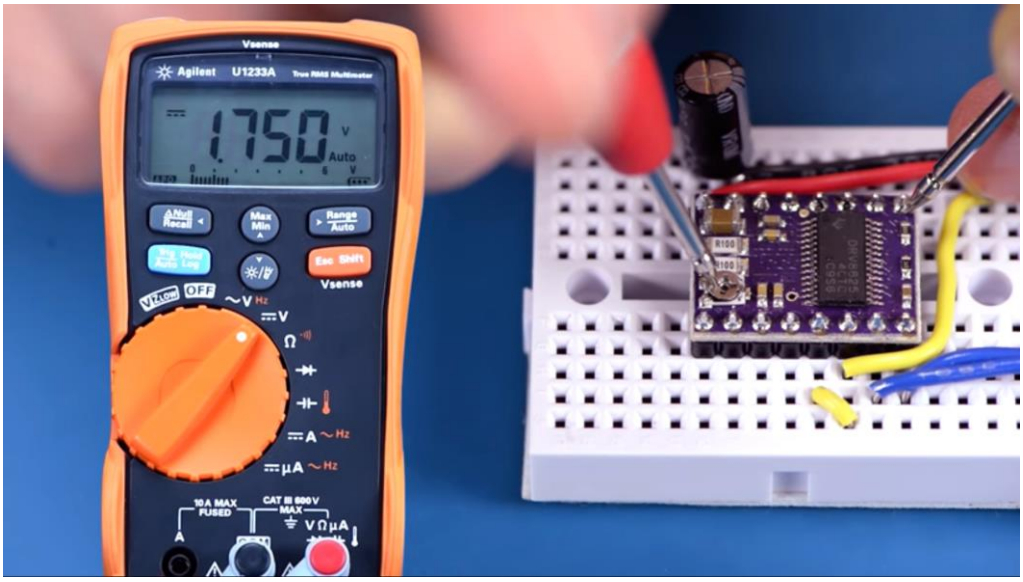


Ilustración 10 - Medida de Vref para la limitación de corriente

Este voltaje, Vref, sirve para calcular la corriente máxima del motor mediante la siguiente fórmula proporcionada por el fabricante:

Ecuación 8 - Fórmula limitación de corriente driver DRV8825

$$Current\ Limit\ =\ VREF\cdot 2$$

Por lo que, para cada motor, se gira el potenciómetro hasta conseguir el valor deseado de Vref. Una vez ajustado este valor, se podrá conectar el motor sin peligro de sobrecalentamiento.

4.1.3.1. Configuración articulaciones 1 y 3 (motor Nema 17)

El motor de estas articulaciones tiene una configuración bipolar, lo que significa que se deben conectar los cuatro conectores al driver. En la siguiente imagen (Ilustración 11) se pueden observar las diferencias de conexión entre un motor unipolar y otro bipolar:

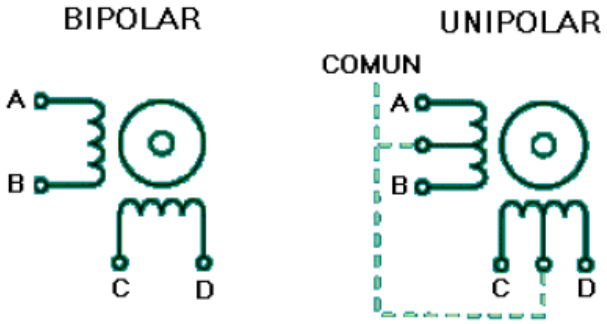


Ilustración 11 - Diferencias entre motor unipolar o bipolar

Al tratarse de motores bipolares las conexiones con el resto de los componentes serán las siguientes:

Articulación 1	
Pines DRV8825 1	**STM32F4 Discovery**
ENABLE	-
M0	GND
M1	GND
M2	+5V
RESET	+5V
SLEEP	+5V
STEP	PD7
DIR	PD5
VMOT	V+ Fuente alimentación
GND (pin arriba derecha)	GND Fuente alimentación
GND (pin abajo derecha)	GND STM32 F4 discovery
Pines DRV8825 1	**Motor 1**
B2	Cable azul
B1	Cable rojo
A1	Cable negro
A2	Cable verde
FAULT	-

Tabla 8 - Conexiones articulación 1

Articulación 3	
Pines DRV8825 3	**STM32F4 Discovery**
ENABLE	-
M0	GND
M1	GND
M2	+5V
RESET	+5V
SLEEP	+5V
STEP	PD4
DIR	PD2
VMOT	V+ Fuente alimentación
GND (pin arriba derecha)	GND Fuente alimentación
GND (pin abajo derecha)	GND STM32 F4 discovery
Pines DRV8825 3	**Motor 3**
B2	Cable azul
B1	Cable rojo
A1	Cable negro
A2	Cable verde
FAULT	-

Tabla 9 - Conexiones articulación 3

En las tablas de conexiones (Tabla 8 y Tabla 9), se puede observar que el modo de resolución 1/16 ha sido seleccionado en los dos drivers según la información de la Tabla 7.

En cuanto a la **limitación de corriente**, el datasheet del motor Nema 17 determina que la corriente máxima por fase es de 2 amperios. Sin embargo, no es recomendable que el motor trabaje en corriente nominal por el problema del sobrecalentamiento, así que la corriente limitada por el driver se fija en 1.4 amperios al ser Vref = 0.7 voltios medidos entre el potenciómetro y GND.

4.1.3.2. Articulación 2 (motor Nema 24)

En el caso del motor Nema 24 el datasheet (Anexo III: Datasheet motor paso a paso Nema 24) sugiere las siguientes combinaciones de conexiones según si se quiere trabajar en unipolar, bipolar en serie o bipolar en paralelo:

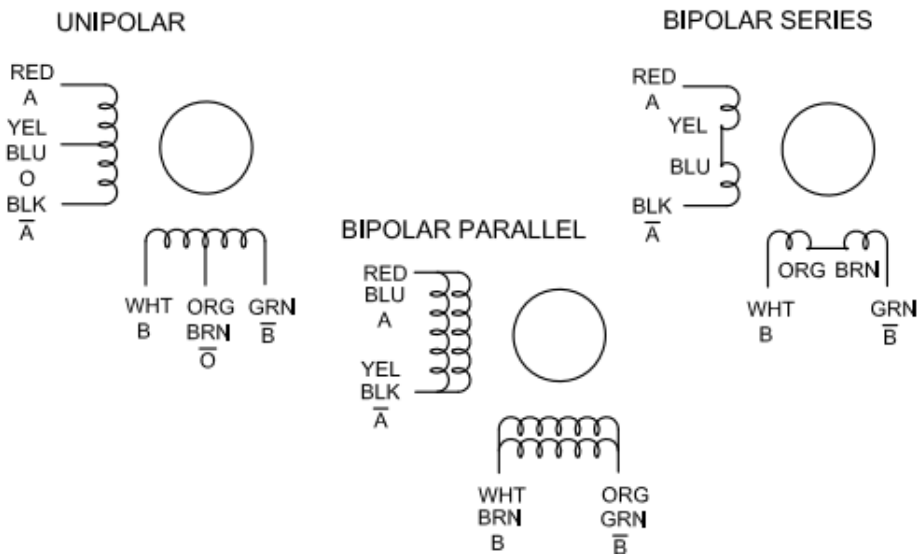


Ilustración 12 - Conexiones Nema 24

También se detalla el consumo de corriente según el tipo de conexión realizada en la siguiente imagen (Ilustración 13):

UNIPOLAR	BIPOLAR (SERIAL)	BIPOLAR (PARALLEL)
3.00	2.12	4.24

Ilustración 13 - Consumo Nema 24

Por lo tanto, se puede observar que el modo bipolar en serie es el que menor consumo realiza. Este es el modo seleccionado y a continuación se detallan las conexiones a realizar.

Para trabajar en modo bipolar en serie, se sueldan los cables amarillo con azul y el naranja con el marrón. Después de soldados, se le añade un plástico termorretráctil para evitar posibles contactos y mantener cierta seguridad como se puede observar en la siguiente imagen (Ilustración 14):

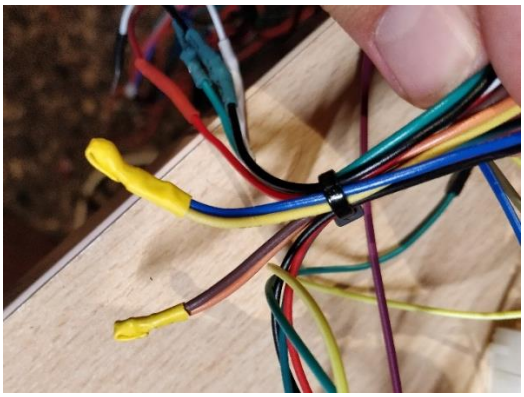


Ilustración 14 - Conexión cables Nema 24

Por tanto, las conexiones del motor con el driver y el microcontrolador serán las siguientes:

Articulación 2	
Pines DRV8825 2	**STM32F4 Discovery**
ENABLE	-
M0	GND
M1	GND
M2	+5V
RESET	+5V
SLEEP	+5V
STEP	PD3
DIR	PD1
VMOT	V+ Fuente alimentación
GND (pin arriba derecha)	GND Fuente alimentación
GND (pin abajo derecha)	GND STM32 F4 discovery
Pines DRV8825 3	**Motor 3**
B2	Cable verde
B1	Cable blanco
A1	Cable rojo
A2	Cable negro
FAULT	-

Tabla 10 - Conexiones articulación 2

Por último y antes de realizar las conexiones del motor con el driver, se debe determinar la **limitación de corriente** para este motor. En este caso, la corriente máxima según el datasheet (Anexo III: Datasheet motor paso a paso Nema 24) es de 2.12 amperios, sin embargo, del mismo modo que con los otros motores se reduce para no provocar sobrecalentamientos. Se ajusta el potenciómetro del driver 2 para medir Vref = 0.8 voltios y por tanto ajustar la corriente máxima a 1.6 amperios.

4.1.4. Alimentación de los motores

Para alimentar los motores se selecciona una fuente de alimentación regulable capaz de proporcionar suficiente corriente para los tres motores en funcionamiento simultáneamente. Cabe recalcar que, aunque los tres motores no se muevan a la vez, sí que están constantemente consumiendo porque necesitan la corriente para los electroimanes y mantener el ángulo de giro.

La fuente seleccionada es la siguiente: *350W 36V 9.7A 115/230V Switching Power Supply* Con 9.7 amperios se satisface la demanda de corriente puesto que se ha determinado el máximo que pueden consumir los motores ajustando el Vref de los drivers.

Ilustración 15 - Fuente de alimentación

4.1.5. Programación control de giro

Como ya se ha visto en apartados anteriores (Selección motores), los motores son controlados por drivers que se encargan de alimentar las bobinas en la secuencia correcta para efectuar el giro. Por lo tanto, para controlar los drivers, se usa el microcontrolador con los pines que se ha determinado en el apartado 4.1.3 (Conexiones y configuración de los drivers) para realizar la rotación y el sentido de la rotación. Esta programación sirve para los dos tipos de motores al ser los dos controlados por el mismo tipo de driver.

4.1.5.1. Cálculos

Para determinar el ángulo de giro se envía un tren de pulsos al driver, que es una señal de HIGH-LOW durante un tiempo determinado y que cada subida y bajada de la señal se contabiliza como un paso. Por tanto, la conversión de ángulo a pasos sería la siguiente:

La resolución del motor es de 1.8 º, que multiplicado por la resolución del driver (1/16) se obtiene una resolución final de 0.1125 º, por lo que cada subida y bajada de la señal controla al motor para que realice un giro de 0.1125 º en el sentido que determine el pin DIR.

A continuación, se debe determinar el período de la señal que se envía al driver. A partir de la gráfica de curva de torque del motor, se escoge un rango de velocidades en las que funcionará el motor para poder cambiarla desde el control. A partir de esta tabla (Ilustración 16), se puede observar que a medida que aumenta el número de pulsos por segundo, el torque disminuye. En este caso, se determina entre [500 – 4000] pps (pulsos por segundo).

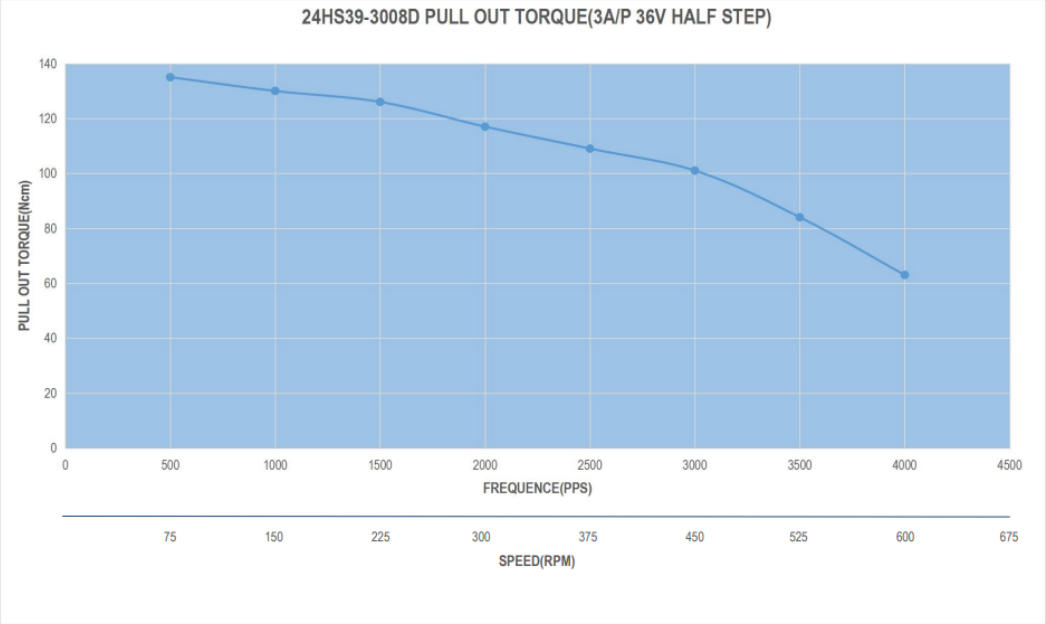


Ilustración 16 - Gráfica curva de torque

Por lo tanto, se va a aproximar a una recta para realizar la función que determina la velocidad. En primer lugar, se calcula el periodo de la señal para 500 pps:

Ecuación 9 - Cálculo del periodo de la señal de pasos

$$T_1 = \frac{1\ s}{500\ pps} = 2000\ \mu s$$

Y para 4000 pps:

Ecuación 10 - Cálculo del periodo de la señal de pasos para 4000 pps

$$T_2 = \frac{1\ s}{4000\ pps} = 250\ \mu s$$

A continuación, en el eje x se ajusta para que la velocidad a introducir sea de 0 a 100, por lo que a velocidad 0, el período debe ser de $2000\ \mu s$; y a velocidad 100, el período será de $250\ \mu s$. Debido a que el ancho de pulso no afecta a la velocidad de rotación, se fija en un 50 % de ancho, por lo que se deben dividir los valores anteriores a la mitad para el cálculo de tiempos de HIGH-LOW.

Se ajusta la recta de la siguiente forma:

Ecuación 11 - Datos para ecuación de la recta de velocidades

$$x = 0 \quad \rightarrow \quad y = 1000\ (\mu s)$$

$$x = 100 \quad \rightarrow \quad y = 125\ (\mu s)$$

Por lo tanto, los valores de pendiente y ordenada en el origen de la recta serán:

Ecuación 12 - Pendiente y ordenada en el origen de la recta

$$n = 1000$$

$$m = \frac{125 - 1000}{100} = -8.75$$

Quedando la recta de la siguiente forma:

Ecuación 13 - Ecuación de la recta de velocidad

$$y\ (tiempo\ en\ \mu s) = -8.75 \cdot x\ (velocidad\ de\ 0\ a\ 100) + 1000$$

4.1.5.2. Implementación en C

Habiendo realizado los cálculos necesarios, se pasa a implementar la funcionalidad en el programa Keil para programar el microcontrolador basándose en el libro de [4] y el manual para la programación que proporciona el fabricante [5].

Se ha creado una función para cada articulación, cuyos parámetros de entrada son los siguientes: ángulo deseado, ángulo anterior y velocidad deseada. Como parámetro de salida, devuelve el valor actual de giro en grados una vez ha realizado el movimiento. En el ejemplo que se detalla a continuación se usa la función para la articulación 2.

Las partes que componen esta función son las siguientes:

- **Inicialización de pines:** primeramente, se realiza la declaración de todos los pines necesarios para el control de los 3 motores en la función *void pin_inicializar(void)*. Se configuran de la siguiente forma:

```
GPIO_InitTypeDef GPIO_Motor;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

    GPIO_Motor.GPIO_Pin = GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15|GPIO_Pin_5|GPIO_Pin_1|GPIO
_Pin_3|GPIO_Pin_4|GPIO_Pin_2|GPIO_Pin_7|GPIO_Pin_6;
    GPIO_Motor.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_Motor.GPIO_OType = GPIO_OType_PP;
    GPIO_Motor.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Motor.GPIO_PuPd = GPIO_PuPd_NOPULL;

    GPIO_Init(GPIOD, &GPIO_Motor);
```

- **Declaración de variables y cálculos previos**: en esta parte se declaran las variables necesarias y se realiza el cálculo del número de pasos y la velocidad de estos.

```
int16_t stepper2(int16_t angulo, int16_t angulo_ant, int8_t vel_0_100){
    //Funcion para la joint 2: stepper Nema 24 y driver drv8825
    fc2=1;
    // ángulo máximo permitido
```

```
if(angulo>240){angulo=240;}
//cáculo del número de pasos
double steps=__fabs(angulo-angulo_ant)/STEP_ANGLE2;
int16_t i=steps;

int16_t steps_vuelta=0;
//cálculo del periodo según la velocidad
uint16_t cycle=vel_0_100*(-8.75) +1000;
int16_t ang_devuelto=0;
```

- **Determinar el sentido de giro:** el sentido es determinado durante las pruebas de funcionamiento y varía según la articulación.

```
//Sentido
if((angulo-angulo_ant)<0){GPIO_SetBits(GPIOD,GPIO_Pin_1);} //Giro hacia atrás
    else {GPIO_ResetBits(GPIOD,GPIO_Pin_1);} //Giro en sentido normal
```

- **Envío de la señal de movimiento del motor:** en esta parte se genera la señal mediante un bucle cuya condición es el número de pasos y que el final de carrera de la articulación no se encuentre pulsado. Una vez el motor se acerca al final del giro la velocidad se reduce. Para crear la función de delay_micros(), se usan los conceptos de temporizadores extraídos de Temporización mediante el temporizador del sistema SysTick en microcontroladores ARM Cortex-M [6].

```
 //Bucle para mover
 while(i>0 && fc2==1){
     if(i<100){
         GPIO_SetBits(GPIOD,GPIO_Pin_3);
         delay_micros(1.3*cycle);
         GPIO_ResetBits(GPIOD,GPIO_Pin_3);
         delay_micros(1.3*cycle);

     } else{ GPIO_SetBits(GPIOD,GPIO_Pin_3);
                     delay_micros(cycle);
                     GPIO_ResetBits(GPIOD,GPIO_Pin_3);
                     delay_micros(cycle);}
 i--;
 }
```

Condiciones una vez el bucle ha finalizado:

- **Si el bucle de giro ha finalizado por pulsar el final de carrera**, se envía un mensaje a la aplicación para resetear el ángulo en la interfaz, la función devuelve el ángulo de giro real, se realiza un cambio de sentido y la articulación se separa 5º para que el final de carrera no se encuentre pulsado
- **Si el ángulo demandado es el ángulo actual**, se devuelve como valor el ángulo actual.
- **Si el sentido de giro es positivo**, el ángulo devuelto se calcula con el número de steps sumando el ángulo anterior.
- **Si el sentido de giro es negativo**, se le resta al ángulo anterior el ángulo que se ha girado calculado con el número de steps.

```
//Si se pulsa el FC
```

```
    if(fc2==0){
        //Envia mensaje de vuelta que esta a 0
        USARTSend(puesta_cero2, sizeof(puesta_cero2));
        ang_devuelto=29;//angulo real 29º
        //Realiza una vuelta de 5º
        GPIO_ResetBits(GPIOD,GPIO_Pin_1);
        while(steps_vuelta<44){
            GPIO_SetBits(GPIOD,GPIO_Pin_3);
            delay_micros(cycle);
            GPIO_ResetBits(GPIOD,GPIO_Pin_3);
            delay_micros(cycle);
            steps_vuelta++;
        }


        fc2=1;
        return ang_devuelto;

    }else if(angulo-angulo_ant==0){
        //si se pide el mismo ángulo en el que se encuentra devuelve el angulo
        return angulo;

    }else if(angulo-angulo_ant>0){
        //si el sentido de giro es normal: cálculo del angulo devuelto
        ang_devuelto=(steps)*STEP_ANGLE2+angulo_ant;
        return ang_devuelto;

    }else if(angulo-angulo_ant<0){
        //si el sentido de giro es contrario
        ang_devuelto=angulo_ant-(steps)*STEP_ANGLE2;
        return ang_devuelto;

}
}
```

Las modificaciones de la función para cada articulación, a parte de cambiar los pines para el tren de pulsos y sentido, son los siguientes:

- **Articulación 1:** para esta articulación se le añade el cálculo de la relación de engranajes que se ha visto en el apartado de selección de motores.

Ecuación 14 - Cálculo relación de engranajes inverso

$$3.2 = \frac{1}{0.3125}$$

```
double steps=  fabs(3.2*(angulo-angulo_ant))/STEP_ANGLE2;
```

Para esta articulación, el ángulo devuelto cuando se pulsa el final de carrera es -85º.

- **Articulación 3:** el único cambio destacable de esta articulación es el ángulo de vuelto al pulsar el final de carrera, cuyo valor es de 29º.

- **Finales de carrera:**

Los finales de carrera son pulsadores cuya utilidad es la de finalizar el movimiento en alguna dirección de un motor. Por ejemplo, para impresoras 3D se coloca al principio de cada eje para determinar el 0 en los ejes de coordenadas y empezar a imprimir a partir de esa posición. Se trata de una solución barata y sencilla de implementar, por lo que su uso es muy popular.

Para implementar esta funcionalidad se han utilizado las interrupciones del microcontrolador STM32. Estas interrupciones se activan cuando se cumplen ciertas condiciones, paran la ejecución normal del programa y ejecutan el *handler*. En el siguiente diagrama (Ilustración 17) se explica el funcionamiento de las interrupciones:

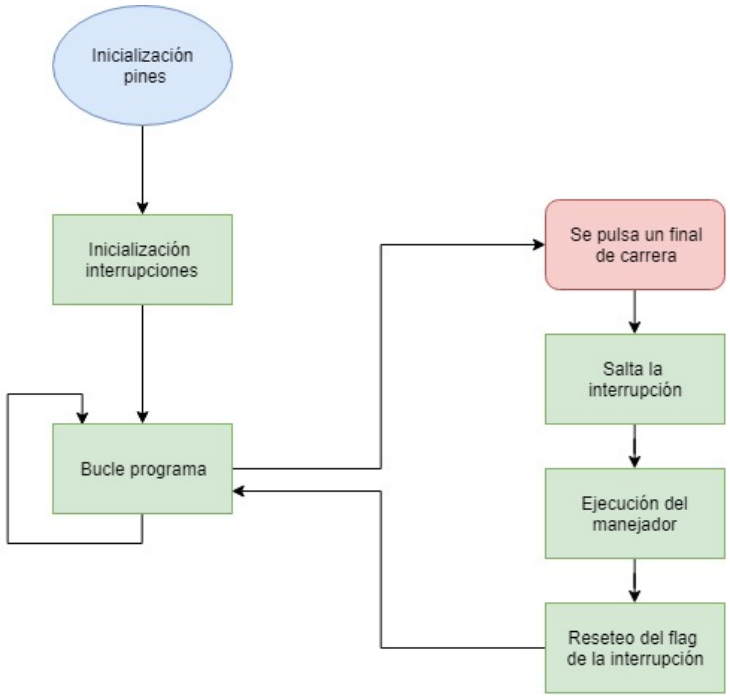


Ilustración 17 - Diagrama manejo interrupciones

De esta forma, se ha conectado la señal del final de carrera de cada articulación a un pin de la tarjeta. Por lo tanto, cuando se activa el pulsador, se envía la señal a la tarjeta y se ejecutan las interrupciones. En la siguiente tabla (Tabla 11) se adjuntan los pines de cada final de carrera y su manejador de la interrupción:

Articulación	Pin microcontrolador	Manejador de la interrupción
Articulación 1	PC3	EXTI3_IRQHandler
Articulación 2	PC4	EXTI4_IRQHandler

Articulación 3	PC5	EXTI9_5IRQHandler

Tabla 11 - Pines final de carrera

Las funciones manejadoras de la interrupción se encuentran en el programa *stm32f4xx_it.c* de la siguiente forma:

```
extern volatile uint8_t fc1;
void EXTI3_IRQHandler(void){

    if(EXTI_GetITStatus(EXTI_Line3)!=RESET){
        fc1=0;
        GPIO_SetBits(GPIOD, GPIO_Pin_12);


    }

EXTI_ClearITPendingBit(EXTI_Line3);
}
```

La variable fc1 se declara como externa para que se pueda utilizar en el *main.c*, así cuando se ejecuta el manejador, esta variable pasa a 0 y finaliza la condición de giro de la función *stepper1* visto anteriormente.

Una vez, se ejecuta el manejador, se borra el *flag* de la interrupción. A cada articulación le corresponde un manejador y una variable diferente, todas declaradas en este programa.

4.1.6. Pinza herramienta

Para que el robot pueda realizar tareas de pick and place se ha desarrollado una pinza servo controlada. El diseño de la pinza se puede observar en el apartado 4.2.3.11(Herramienta pinza servo). Parte del código se ha extraído de la librería de (Majerle, n.d.)

Por lo tanto, se debe realizar un control de apertura y cierre de la pinza para poder agarrar objetos. Este movimiento se controla con un servo al que se le pasa una señal PWM desde el microcontrolador para determinar el ángulo de giro.

4.1.6.1. Configuración pines

Según el datasheet del servo, se obtiene que para girar a un ángulo de 90º se necesita un pulso en alto de 2 ms en un ciclo de 20 ms. Y para girar -90º, un pulso de 1 ms en un ciclo de 20 ms. El pin que enviará esta señal es el PD12 que se configura como función alternativa de la siguiente forma:

```
//PRueba PWM
        GPIO_InitTypeDef GPIO_PWM;

    /* Clock for GPIOD */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

    /* Alternating functions for pins */
    GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_TIM4);

    /* Set pins */
```

```
GPIO_PWM.GPIO_Pin = GPIO_Pin_12;
GPIO_PWM.GPIO_OType = GPIO_OType_PP;
GPIO_PWM.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_PWM.GPIO_Mode = GPIO_Mode_AF;
GPIO_PWM.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOD, &GPIO_PWM);
```

4.1.6.2. Configuración señal PWM

A continuación, se procede a configurar el timer encargado de enviar la señal PWM con el ancho de pulso y la frecuencia deseada.

Al necesitar un pulso de 20 ms, la frecuencia será de 50 Hz. La frecuencia de reloj del microcontrolador es de 84 MHz, por lo que hay que reducirla de la siguiente forma:

- Se divide por un preescaler de 999 para obtener una frecuencia de 84 KHz.
- Se utiliza la siguiente ecuación para el cálculo del periodo de la señal:

Ecuación 15 - Cálculo del periodo de la señal PWM

$$TIM_{Period} = \frac{timertick_{frequency}}{PWM_{frequency}} - 1 = \frac{84000}{50} - 1 = 1679$$

En el siguiente código se puede observar cómo se emplean estos valores para la configuración del timer:

```
void TM_TIMER_Init(void) {
    TIM_TimeBaseInitTypeDef TIM_BaseStruct;

    /* Enable clock for TIM4 */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
/*
    TIM4 is connected to APB1 bus, which has on F407 device 42MHz clock
    But, timer has internal PLL, which double this frequency for timer, up to 84MHz
    Remember: Not each timer is connected to APB1, there are also timers connected
    on APB2, which works at 84MHz by default, and internal PLL increase
    this to up to 168MHz

    Set timer prescaller
    Timer count frequency is set with

    timer_tick_frequency = Timer_default_frequency / (prescaller_set + 1)

    In our case, we want a max frequency for timer, so we set prescaller to 0
    And our timer will have tick frequency

    timer_tick_frequency = 84000000 / (0 + 1) = 84000000
*/
    TIM_BaseStruct.TIM_Prescaler = 999;
    /* Count up */
    TIM_BaseStruct.TIM_CounterMode = TIM_CounterMode_Up;
/*
    Set timer period when it have reset
    First you have to know max value for timer
    In our case it is 16bit = 65535
    To get your frequency for PWM, equation is simple
```

```
    PWM_frequency = timer_tick_frequency / (TIM_Period + 1)

    If you know your PWM frequency you want to have timer period set correct

    TIM_Period = timer_tick_frequency / PWM_frequency - 1

    In our case, for 10Khz PWM_frequency, set Period to

    TIM_Period = 84000000 / 10000 - 1 = 8399

    If you get TIM_Period larger than max timer value (in our case 65535),
    you have to choose larger prescaler and slow down timer tick frequency
*/
    TIM_BaseStruct.TIM_Period = 1679; /* 50Hz PWM */
    TIM_BaseStruct.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_BaseStruct.TIM_RepetitionCounter = 0;
    /* Initialize TIM4 */
    TIM_TimeBaseInit(TIM4, &TIM_BaseStruct);
    /* Start count on TIM4 */
    TIM_Cmd(TIM4, ENABLE);
}
```

Por último se crea la función *TM_PWM_Init(int8_t enab)* para el control del ancho del pulso de la señal. Se calcula el ancho de pulso para las dos posiciones de apertura y cierre de la pinza a partir del valor de periodo calculado antes de la siguiente forma:

Ecuación 16 - Cálculo del ancho de pulso de la señal PWM

$$90^{o} \rightarrow 2ms\ pulso$$

$$\frac{2}{20} = 0.1$$

$$0.1 \cdot 1679 = 125$$

Por lo tanto para un ángulo de 90º, el perido es de 1679 y el pulso de 125; y para un ángulo de -90º, el pulso será de 83. Esto se ha implementado de la siguiente forma:

```
void TM_PWM_Init(int8_t enab) {
    TIM_OCInitTypeDef TIM_OCStruct;

    /* Common settings */

    /* PWM mode 2 = Clear on compare match */
    /* PWM mode 1 = Set on compare match */
    TIM_OCStruct.TIM_OCMode = TIM_OCMode_PWM2;
    TIM_OCStruct.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCStruct.TIM_OCPolarity = TIM_OCPolarity_Low;

/*
    To get proper duty cycle, you have simple equation

    pulse_length = ((TIM_Period + 1) * DutyCycle) / 100 - 1

    where DutyCycle is in percent, between 0 and 100%

    25% duty cycle:     pulse_length = ((8399 + 1) * 25) / 100 - 1 = 2099
    50% duty cycle:     pulse_length = ((8399 + 1) * 50) / 100 - 1 = 4199
```

```
    75% duty cycle:     pulse_length = ((8399 + 1) * 75) / 100 - 1 = 6299
    100% duty cycle:    pulse_length = ((8399 + 1) * 100) / 100 - 1 = 8399

   Remember: if pulse_length is larger than TIM_Period, you will have output HIGH all
the time
 */

 if(enab == 0){TIM_OCStruct.TIM_Pulse = 0 ;}
 else if(enab == 20){TIM_OCStruct.TIM_Pulse = 125; /* 25% duty cycle */}
 else if(enab == -90){TIM_OCStruct.TIM_Pulse = 83; }
 else if(enab == 90){TIM_OCStruct.TIM_Pulse = 167; }


     TIM_OC1Init(TIM4, &TIM_OCStruct);
     TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Enable);

 }
```

4.1.6.3. Alimentación servomotor

Para alimentar el servo se necesitan 5V, sin embargo el microcontrolador no es capaz de proporcionar suficiente corriente para alimentarlo, por lo que se usa un regulador de tensión LM317T para obtener los 5V a partir de los 29V de la fuente de alimentación desde donde se alimentan los motores.

El circuito empleado es el siguiente:

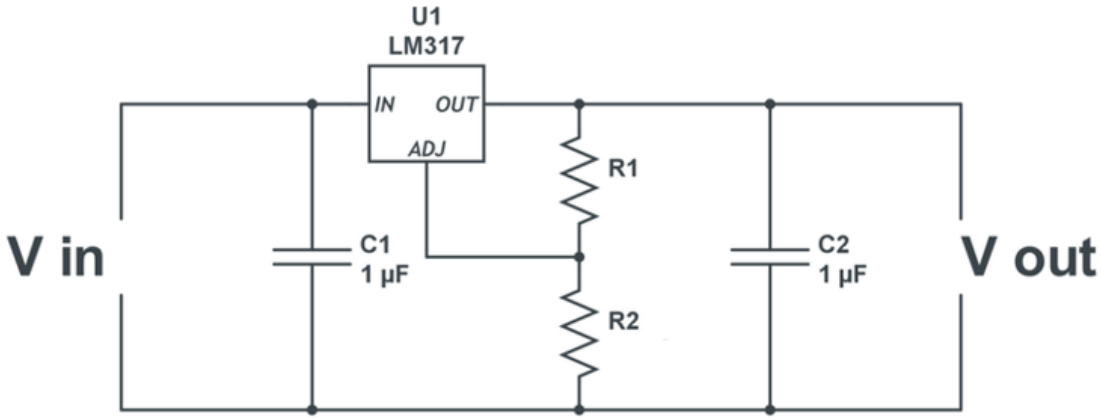


Ilustración 18 - Esquema circuito regulador de tensión LM317T

Para este caso, las resistencias serán R1 = 270 Ω y R2 = 1000 Ω, y son calculadas con la siguiente fórmula proporcionada por el fabricante, en la que el I_{ADJ} se considera cero para la aproximación.

Ecuación 17 - Fórmula de cálculo de tensión de salida para LM317T

$$Vout = 1.25 \cdot \left(1 + \frac{R2}{R1}\right) + I_{ADJ} \cdot R2$$

4.2. Diseño e impresión de piezas

Para el diseño de las piezas que unen las articulaciones del robot se utiliza el programa de Autodesk Fusion 360. Se usa un diseño modular en el que cada pieza se imprime por separado para evitar reimprimir todo el robot en el caso de que una pieza no cumpla con su cometido o exista algún error de impresión.

Este es uno de los mayores problemas de la impresión 3D, en el programa se diseña con unas medidas exactas, pero a la hora de imprimir siempre hay problemas para que las piezas encajen. La causa de este problema es que el PLA puede llegar a expandirse al irse aplastando capa sobre capa y al no encajar dos piezas hay que lijar las partes afectadas o rediseñar teniendo en cuenta un margen.

Por lo tanto, las piezas que se muestran en el siguiente apartado son fruto de la iteración entre diseño e impresión de varias veces hasta llegar a un componente óptimo. Sin embargo, muchas de las piezas han sido diseñadas correctamente a la primera gracias a poder tomar medidas reales con un pie de rey y poder realizar pruebas de ensamblaje en el software.

Para este apartado del proyecto se han usado las siguientes herramientas:

Software:

- **Autodesk Fusion 3D:** programa usado para el diseño y ensamblaje de piezas.
- **Simplify 3D:** programa encargado de configurar la impresora para cada pieza y generar el archivo GCODE.

Hardware:

- **Pie de rey:** herramienta para tomar medidas de forma precisa e implementarlas en el diseño en CAD.

Ilustración 19 - Pie de rey

- **Impresora 3D:** en este caso se ha utilizado la impresora Creality Ender 3.

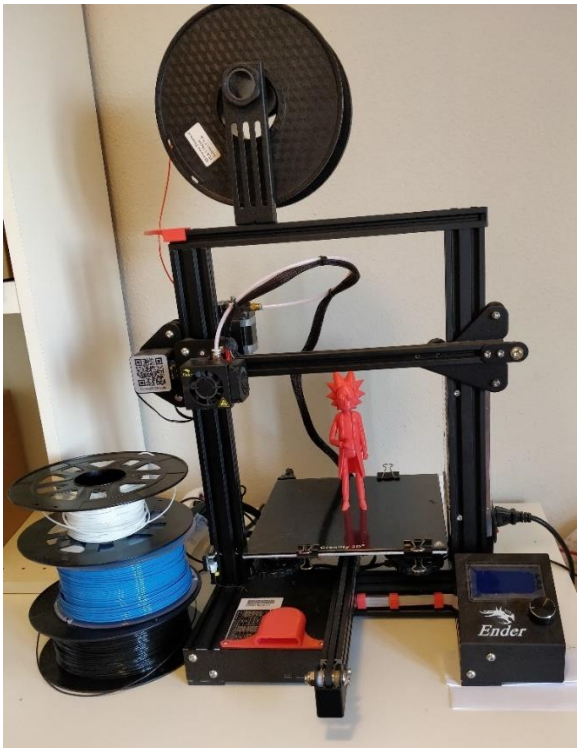


Ilustración 20 - Impresora 3D Creality Ender 3

4.2.1. Proceso de diseño

Para comenzar a diseñar una de las piezas, primeramente, se realiza un boceto a mano de la pieza. Seguidamente, se realiza una toma de medidas con un pie de rey además de utilizar la información que ofrece el fabricante sobre las medidas de los motores.

Una vez las medidas necesarias han sido tomadas se procede a diseñar con el software Autodesk Fusion 360. En este programa se diseña creando un boceto en dos dimensiones mediante las diversas herramientas de dibujo que se ofrecen y configurando las medidas deseadas.

A continuación, mediante la herramienta Extruir, se añade la profundidad a la pieza. A partir de esta pieza, se puede volver a realizar un boceto sobre alguna cara de las caras para modificarla o volver a realizar una extrusión. En la siguiente imagen, se puede observar la interfaz del programa de diseño.

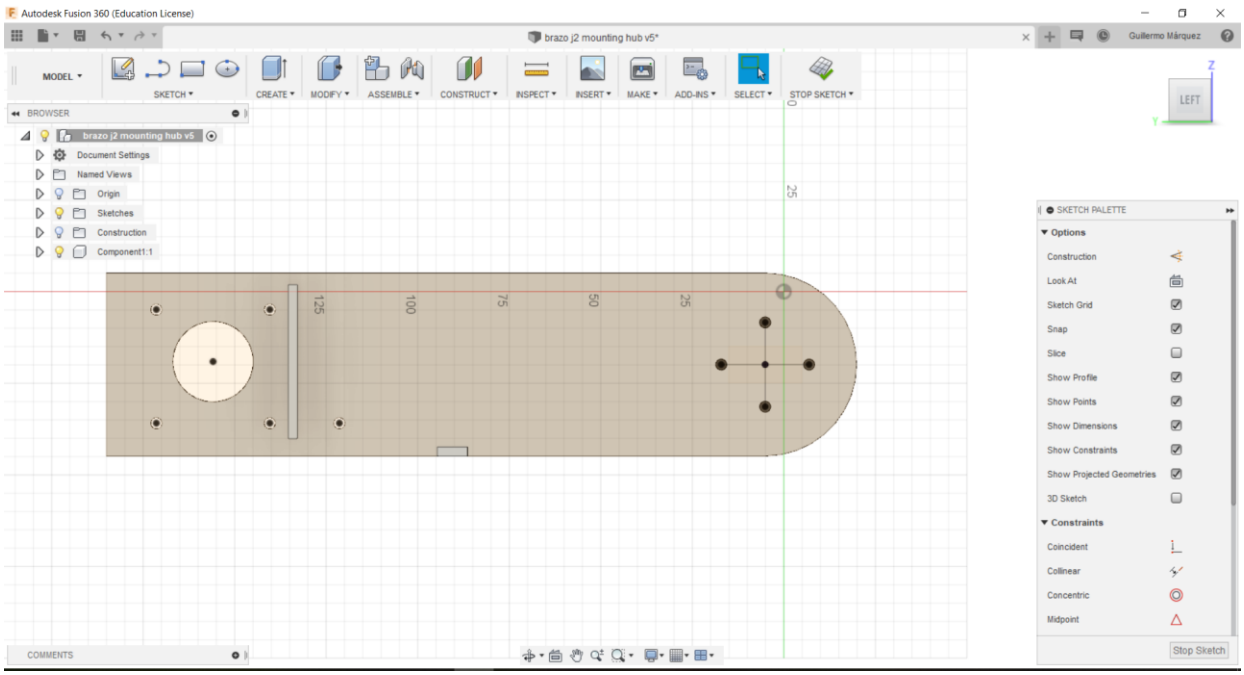


Ilustración 21 - Interfaz del programa Fusion 360

Una vez se tiene el objeto diseñado, se exporta el archivo en formato .stl para ser procesado mediante el software Simplify 3D, encargado de configurar los parámetros de impresión de la impresora y generar el GCODE. Este tipo de archivo se compone de una serie de coordenadas que son leídas por la impresora para crear el objeto.

4.2.2. Ajustes de impresión

Dentro del software Simplify 3D, se tiene la siguiente interfaz:

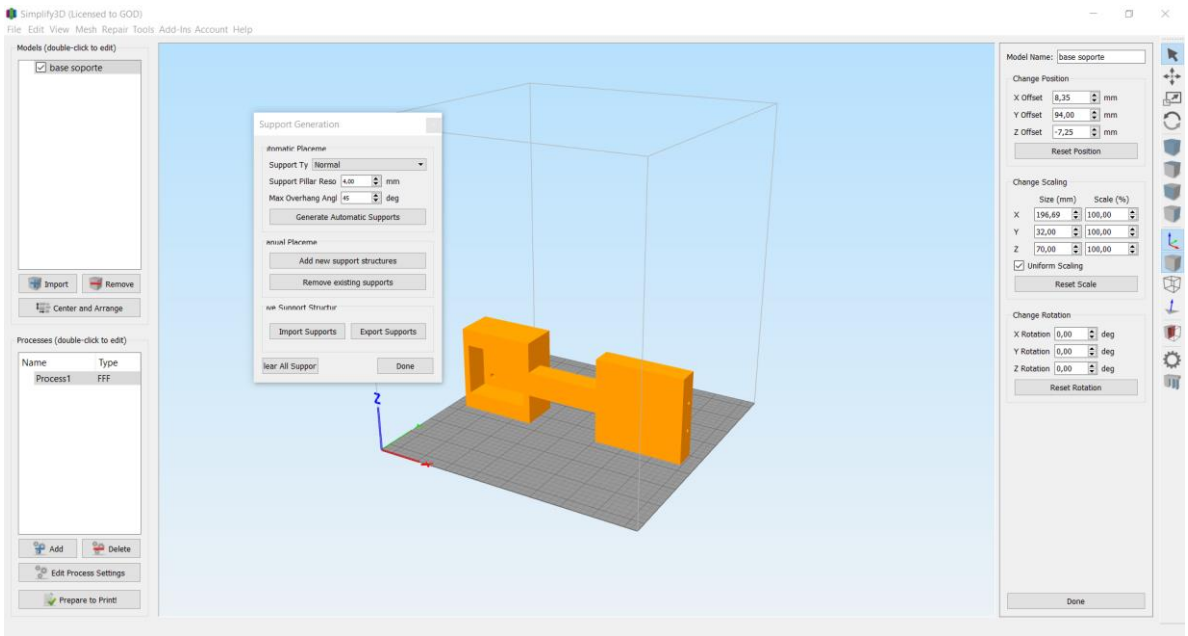


Ilustración 22 - Interfaz Simplify 3D

Este programa presenta multitud de herramientas para tratar los objetos que se han diseñado. Un aspecto importante es la creación de soportes para imprimir piezas cuya forma presenta alguna complejidad. Con estos soportes, se crea una base para depositar el material si la figura presenta algún voladizo, y una vez se ha impreso, los soportes son fáciles de retirar.

Existen multitud de configuraciones para imprimir una pieza en 3D y cada configuración funciona mejor con cada impresora. En este caso, las configuraciones de los aspectos más importantes han sido las siguientes:

- **Layer: (0.2 mm)** este parámetro determina la altura de la capa de material depositado. Cuanto mayor sea la capa, menor será el tiempo de impresión, pero el nivel de detalle será peor.
- **Relleno: (10 %)** este parámetro es la cantidad de material que lleva el objeto por dentro. Con este porcentaje es suficiente puesto que se crean estructuras lo bastante fuertes como para soportar esfuerzos mecánicos y al mismo tiempo ahorra gran cantidad de material.
- **Temperatura de la boquilla: (220º)** la temperatura depende principalmente del material, por eso es el fabricante el encargado de determinar cuál es la adecuada.
- **Temperatura de la base: (50º)** en principio el PLA es un material que no necesita de temperatura en la base para imprimirse, sin embargo, es recomendable para que la pieza no se despegue durante la impresión.
- **Ventilador de capa: (100%)** este ventilador consigue que el material se enfríe rápidamente, consiguiendo una mejora en el acabado estético y evitar problemas durante la impresión.
- **Velocidad de impresión: (40-60 mm/s)** cuanto más rápido se imprima la pieza, más posibilidades hay de que se produzcan fallos, por lo que hay que buscar el equilibrio entre eficacia y rapidez. En este caso, al tratarse de piezas que no presentan mucho detalle, se puede aumentar la velocidad en algunas impresiones.

4.2.3. Componentes necesarios

Se pasa a enumerar el listado de piezas necesarias que han sido diseñadas e impresas en 3D empezando desde la base y acabando en la herramienta. En este caso se trata de imágenes extraídas del software de diseño Fusion 360, pero en el documento de planos se encuentran todas estas figuras acotadas y con su escala.

4.2.3.1. Base soporte articulación 1

La base es donde se apoya todo el peso del robot y necesita ser estable para no caer mientras se encuentra en movimiento.

Observando la imagen de la pieza, en el cuadrado de la izquierda se han diseñado 4 orificios para tornillos M5 que se encargan de sujetar el rodamiento de la base. En el otro extremo se ha realizado un orificio cuadrado donde va encajado el motor nema 17 encargado de mover la articulación 1. Se han añadido orificios en un extremo para poder sujetar el motor presionando con unos tornillos de cabeza M4.

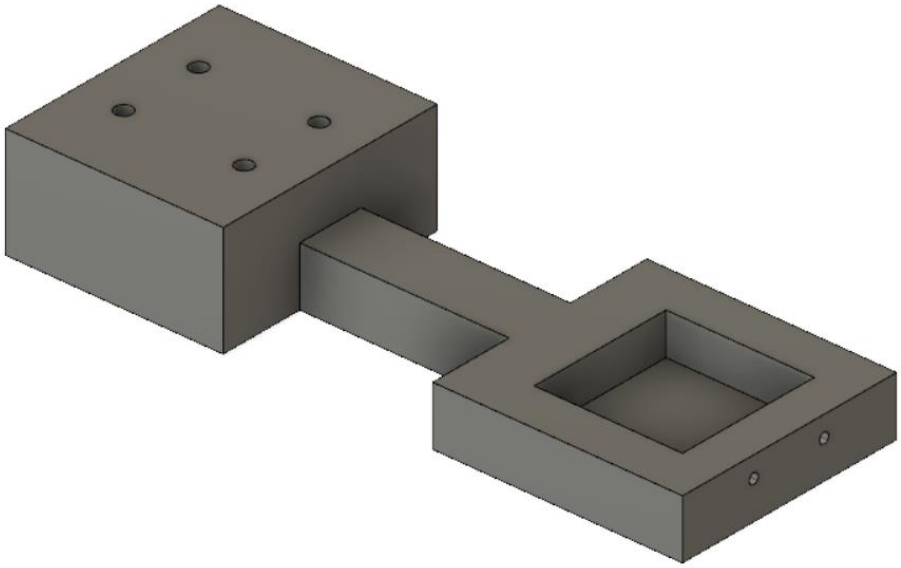


Ilustración 23 - Base soporte articulación 1

Esta pieza ha sido impresa en PLA de color negro y sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 6 horas 45 minutos
- Longitud filamento usado: 34735.4 mm
- Peso del material usado: 104.44 g
- Coste material: 2.09 €

4.2.3.2. Soporte final de carrera articulación 1

Esta pieza se encarga de sujetar el final de carrera de la articulación 1 para que contacte con el eje en el ángulo indicado.

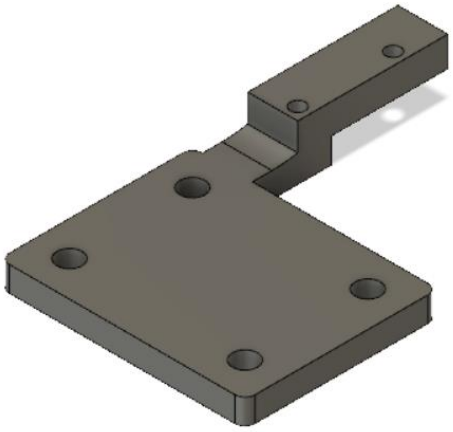


Ilustración 24 - Soporte final de carrera articulación 1

Esta pieza ha sido impresa en PLA de color naranja y sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 48 minutos
- Longitud filamento usado: 2579.3 mm
- Peso del material usado: 8.85 g
- Coste material: 0.18 €

4.2.3.3. Enganche engranaje

Esta pieza se encarga de sujetar el engranaje que se sitúa bajo la base del robot y de mantener toda la pieza en posición horizontal. Como se observa en la siguiente imagen se ha diseñado con 3 orificios para tornillos M5.

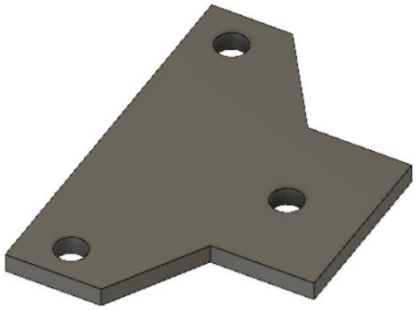


Ilustración 25 - Enganche engranaje

Esta pieza ha sido impresa en PLA de color blanco y sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 23 minutos
- Longitud filamento usado: 1379.6 mm
- Peso del material usado: 4.15 g
- Coste material: 0.08 €

4.2.3.4. Engranaje 64 Thingiverse

Este archivo stl se ha descargado de la biblioteca online *Thingiverse*[1], por lo que no ha hecho falta diseñarlo.

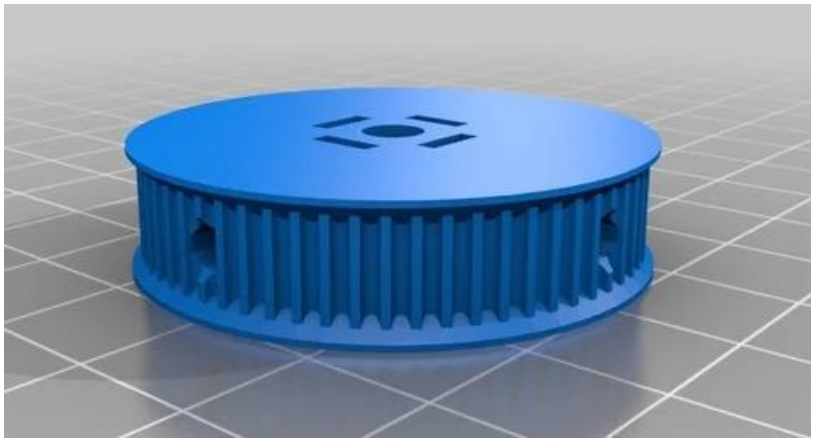


Ilustración 26 - Engranaje GT2 64 dientes

Esta pieza ha sido impresa en PLA de color blanco y sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 48 minutos
- Longitud filamento usado: 2579.3 mm
- Peso del material usado:7.75 g
- Coste material: 0.16 €

4.2.3.5. Base para articulación 2

Esta pieza es de las más importantes, puesto que se encarga de conectar la base con el rodamiento y soportar el motor de más peso. La cara vertical es donde va atornillado el motor y por el orificio circular es por donde sale el eje al que va atornillado todo el brazo.

En la base se puede observar que hay unos soportes circulares donde el peso del motor puede descansar sin provocar esfuerzos mecánicos que provoquen alguna rotura. También se han

[1] https://www.thingiverse.com/

añadido 3 orificios que coinciden con los orificios del enganche del engranaje por lo que pasan 3 tornillos M5.

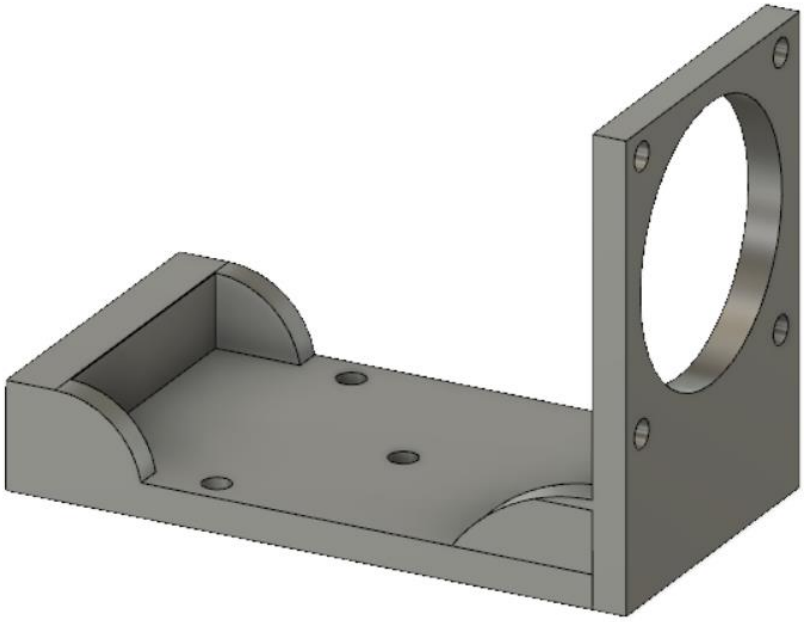


Ilustración 27 - Base para articulación 2

Esta pieza ha sido impresa en PLA de color naranja y sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 4 horas 17 minutos
- Longitud filamento usado: 22854.4 mm
- Peso del material usado: 68.71 g
- Coste material: 1.37 €

4.2.3.6. Soporte final de carrera articulación 2

Sobre este soporte va atornillado el final de carrera de la articulación 2, de esta forma al realizar el giro en este sentido la articulación coincide con esta pieza, pulsando el final de carrera y pudiendo descansar en la posición de reposo. Esta es una de las piezas que ha tenido que ser reimpresa porque durante las pruebas de funcionamiento se rompió.

Ilustración 28 - Soporte final de carrera articulación 2

Esta pieza ha sido impresa en PLA de color blanco y sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 53 minutos
- Longitud filamento usado: 2677.3 mm
- Peso del material usado: 8.05 g
- Coste material: 0.16 €

4.2.3.7. Soporte motor Nema 24

Este soporte se diseña para añadir estabilidad al motor Nema 24 por su peso. Este componente se acopla con tornillos M5 a la base para articulación 2 y el motor va encajado en su interior.

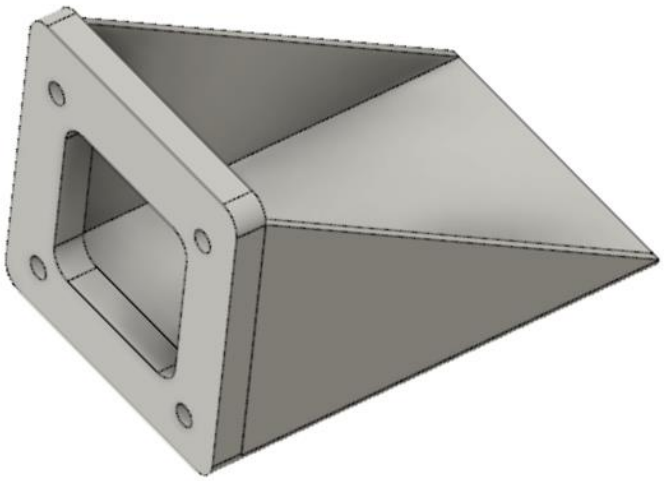


Ilustración 29 - Soporte motor Nema 24

Esta pieza ha sido impresa en PLA de color blanco y sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 3 horas 55 minutos
- Longitud filamento usado: 17891.7 mm
- Peso del material usado: 53.79 g
- Coste material: 1.08 €

4.2.3.8. Brazo articulación 2

Esta pieza es un buen ejemplo sobre el diseño iterativo, debido a que los primero diseños disponían de un orificio para insertar el eje del motor de la articulación 2. Sin embargo, se sustituyó por una pieza metálica debido al peso que tenía que soportar. En el apartado de ensamblaje se detallan todos estos componentes no impresos en 3D.

En la parte derecha de la imagen se puede observar que se han diseñado 4 orificios para montar un acople del eje del motor Nema 24.

En la parte de la izquierda es donde va acoplado el motor para la articulación 3. Por el orificio central es por donde pasa el eje del motor y se acopla con tornillos M3. También se ha añadido otro orificio abajo donde va acoplado el final de carrera de la articulación 3.

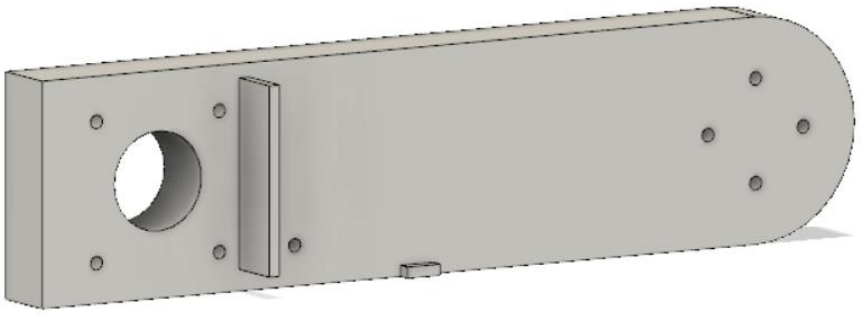


Ilustración 30 - Brazo articulación 2

Esta pieza ha sido impresa en PLA de color negro y sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 4 horas 2 minutos
- Longitud filamento usado: 20620 mm
- Peso del material usado: 62 g
- Coste material: 1.24 €

4.2.3.9. Brazo articulación 3

Esta articulación se diseña basándose en la articulación del brazo 2. También dispone de 4 orificios para el acople del eje del motor Nema 17, además de una pestaña para pulsar correctamente el final de carrera de la articulación 3. En el extremo derecho se han creado unos orificios para acoplar otro motor en el caso de querer ampliar el número de articulaciones, pero en este caso es donde se acopla el enganche para la herramienta y la pinza.

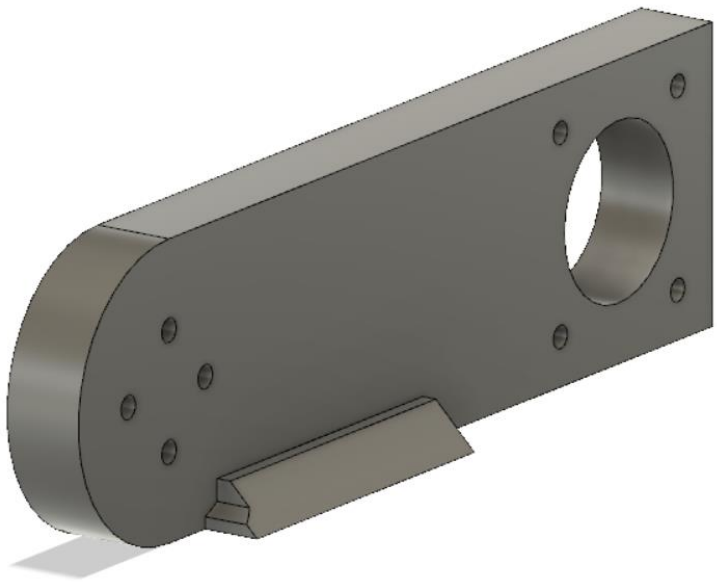


Ilustración 31 - Brazo articulación 3

Esta pieza ha sido impresa en PLA de color negro y sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 2 horas 51 minutos
- Longitud filamento usado: 9149 mm
- Peso del material usado: 27.51 g
- Coste material: 1.24 €

4.2.3.10. Enganche herramienta

Este enganche ha sido diseñado para acoplar el brazo de la articulación 3 con la pinza que se ha descargado de *Thingiverse.* Simplemente son dos pestañas con los orificios necesarios para acoplar las dos piezas.

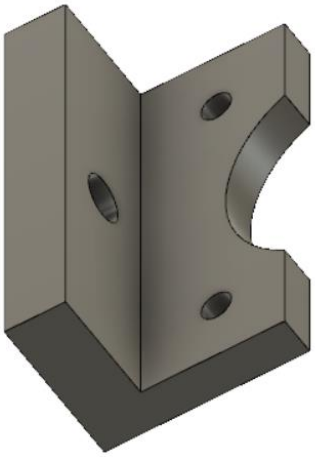


Ilustración 32 - Enganche pinza herramienta

Esta pieza ha sido impresa en PLA de color naranja y sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 29 minutos
- Longitud filamento usado: 1575 mm
- Peso del material usado: 4.74 g
- Coste material: 0.09 €

4.2.3.11. Herramienta pinza servo

Debido a la complejidad de diseñar una pinza desde cero se opta por usar un archivo ya diseñado desde la plataforma *Thingiverse*. La pinza que se ha seleccionado está diseñada para ser controlada por dos servos, pero sólo se usará el que controla la apertura y cierre de la pinza.

La pinza está compuesta por diversas piezas unidas entre si por tornillos M3 y M5. Se imprimen por separado y se ensamblan como muestra la siguiente figura:

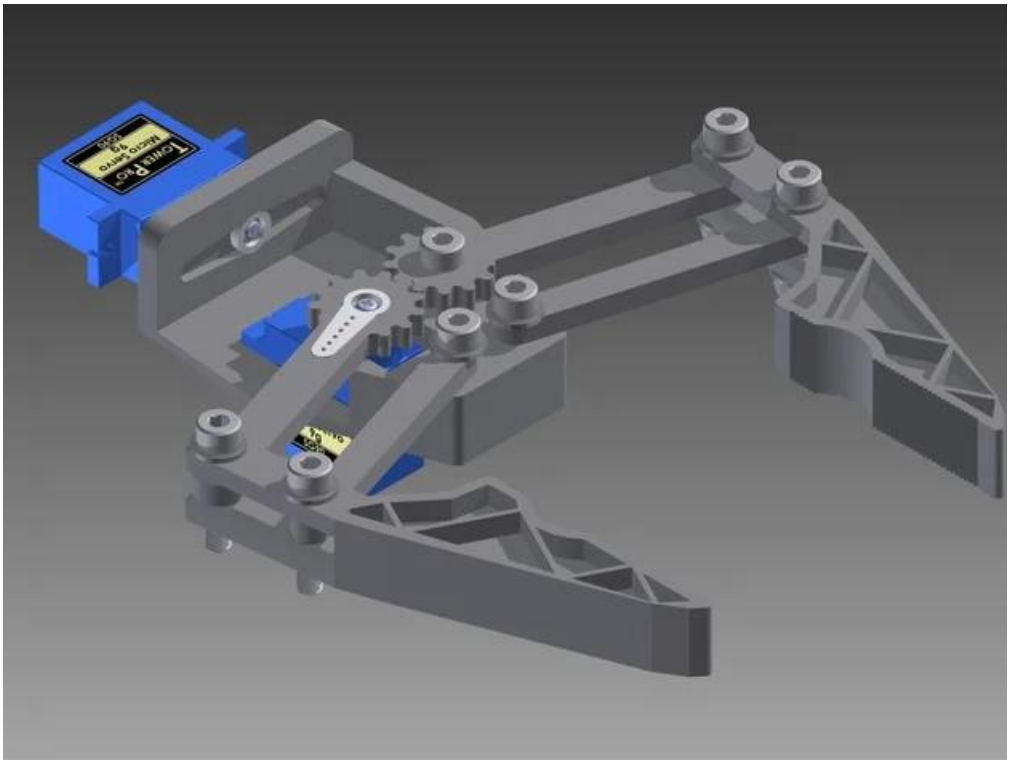


Ilustración 33 - Herramienta pinza servo

Estas piezas han sido impresas en PLA de color naranja y en total, sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 4 horas
- Longitud filamento usado: 12437 mm
- Peso del material usado: 39 g
- Coste material: 0.78 €

4.2.3.12. Encapsulado para los circuitos

Se ha diseñado una caja para guardar todos los circuitos y que los cables queden recogidos. La caja presenta en la base una serie de pines para poder encajar las PCBs y que no se muevan. Se trata de dos piezas, una base y la tapa que encajan perfectamente una vez impresas.

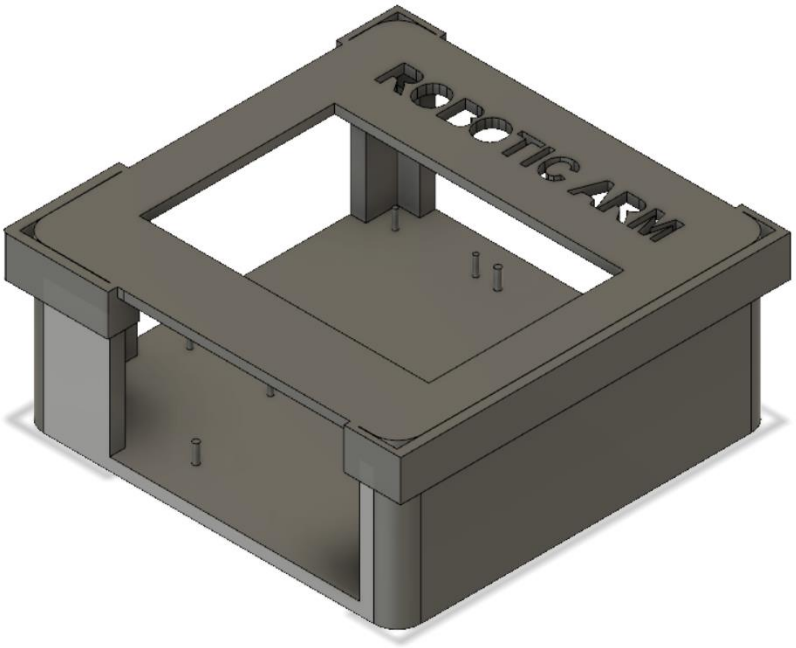


Ilustración 34 - Encapsulado de los circuitos

Estas piezas han sido impresas en PLA de color negro y en total, sus datos de impresión según el programa *Simplify 3D* son los siguientes:

- Tiempo de impresión: 9 horas
- Longitud filamento usado: 43532 mm
- Peso del material usado: 118 g
- Coste material: 2.38 €

4.3. Comunicación bluetooth

La tecnología bluetooth se utiliza para crear un canal de comunicación entre el móvil y el microcontrolador. De esta forma, es posible enviar mensajes en las dos direcciones y controlar el brazo robot telemáticamente y de forma sencilla desde una aplicación.

4.3.1. Módulo bluetooth y configuración

Puesto que el microcontrolador no lleva incorporado un módulo de bluetooth, se utiliza el módulo bluetooth DSD Tech HC-05, muy popular en este tipo de aplicaciones.

Para configurar el dispositivo, se utiliza el microcontrolador Arduino Uno con el siguiente código extraído de la página Prometec[2]:

```
#include <SoftwareSerial.h>

SoftwareSerial BT1(10, 11); // RX | TX
void setup()
   { pinMode(8, OUTPUT);         // Al poner en HIGH forzaremos el modo AT
     pinMode(9, OUTPUT);         // cuando se alimente de aqui
     digitalWrite(9, HIGH);
     delay (500) ;               // Espera antes de encender el modulo
     Serial.begin(38400);
     Serial.println("Levantando el modulo HC-05");
     digitalWrite (8, HIGH);     //Enciende el modulo
     Serial.println("Esperando comandos AT:");
     BT1.begin(38400);
   }

void loop()
   {  if (BT1.available())
            Serial.write(BT1.read());
      if (Serial.available())
            BT1.write(Serial.read());
   }
```

Con este sencillo código se configura una comunicación serial del módulo bluetooth con el microcontrolador Arduino y se habilita el Serial para poder enviar y recibir mensajes para configurar el módulo.

A partir del datasheet del módulo HC-05, se determinan los comandos a enviar para la configuración del módulo. Estos son los siguientes:

- "**AT+NAMEbrazo**": este comando configura el nombre a "brazo" para que, al buscar dispositivos bluetooth desde el móvil, se reconozca por el nombre.
- "**AT+BAUDx**": configura la velocidad de comunicación entre el módulo y el microcontrolador de acuerdo a la siguiente tabla:

1 configura 1200bps

[2] https://www.prometec.net/bt-hc05/

2 configura 2400bps

3 configura 4800bps

4 configura 9600bps (Default)

5 configura 19200bps

6 configura 38400bps

7 configura 57600bps

8 configura 115200bps

En este caso, se envia "AT+BAUD4" para configurarlo por defecto.

- "**AT+PIN**": con este comando se muestra por el serial la contraseña para conectarse al módulo bluetooth. Se deja en el valor que viene por defecto, "1234".
- "**AT+ROLE**": se muestra por el serial si el módulo está configurado como maestro (1) o como esclavo (0). Se le envía "AT+ROLE0" para configurarlo como esclavo.

Una vez configurado con los parámetros anteriores, ya está listo para poder conectarlo al microcontrolador STM32.

4.3.2. Comunicación del módulo bluetooth con el microcontrolador

Para poder comunicar el módulo bluetooth con el STM32, se deben configurar unos pines como *USART* para poder enviar y recibir mensajes. Además, se crea una interrupción, como en los finales de carrera, que se activa cada vez que recibe un mensaje y desde la cual podremos manejar lo que recibimos del móvil. La información y parte del código para la configuración de los pines se extrae de los artículos de la página *solderer.tv* [7].

La comunicación USART [8] (Universal Asynchronous Receiver-Transmitter), es una forma de comunicación serial entre dos dispositivos. Se denomina serial porque los datos son enviados en paquetes de 8 o 9 bits pero de bit en bit, es decir, en serie. La comunicación se realiza mediante dos pines del microcontrolador, uno de los pines hace la función de receptor (Rx) y otro hace de transmisor (Tx).

Los pines han sido escogidos buscando en el datasheet [9]del microcontrolador un par de pines que soporten la tecnología USART, los cuales han sido **PD8 de transmisor (Tx) y PD9 de receptor (Rx)**. La configuración de los pines como *USART3* y de la interrupción se realiza en la función llamada *bluetooth_init (void)* de la siguiente manera:

```
void bluetooth_init(void){

//Inicialización de estructuras
    GPIO_InitTypeDef     GPIO_InitStruct1;
```

```
    GPIO_InitTypeDef     GPIO_InitStruct2;

// Habilitar reloj para GPIOD
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

// Determinar que puerto de comunicación se usa
GPIO_PinAFConfig(GPIOD, GPIO_PinSource8, GPIO_AF_USART3);
GPIO_PinAFConfig(GPIOD, GPIO_PinSource9, GPIO_AF_USART3);
// Configuración de los pines
GPIO_InitStruct1.GPIO_Pin = GPIO_Pin_8;
GPIO_InitStruct1.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct1.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStruct1.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOD, &GPIO_InitStruct1);

GPIO_InitStruct2.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStruct2.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct2.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct2.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStruct2.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOD, &GPIO_InitStruct2);

//Inicialización del puerto de comunicación y de la interrupción
    USART_InitTypeDef USART_InitStruct;
    NVIC_InitTypeDef NVIC_InitStruct;

//Habilitar reloj para el puerto de comunicación
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);

//Configuración del USART3
USART_InitStruct.USART_BaudRate = baudrate;
USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStruct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_InitStruct.USART_Parity = USART_Parity_No;
USART_InitStruct.USART_StopBits = USART_StopBits_1;
USART_InitStruct.USART_WordLength = USART_WordLength_8b;
USART_Init(USART3, &USART_InitStruct);
USART_Cmd(USART3, ENABLE);

//Habilitar la interrupción para el puerto de comunicación USART3
USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);

//Definir el canal y la prioridad de la interrupción
NVIC_InitStruct.NVIC_IRQChannel = USART3_IRQn;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
NVIC_Init(&NVIC_InitStruct);

}
```

4.3.3. Recepción de mensajes desde el microcontrolador

Como se ha explicado en el apartado anterior, la recepción de mensajes se realiza mediante una interrupción, la cual se produce cuando se detecta que el flag del USART3 se encuentra “lleno”, debido a que hay un mensaje.

Por lo tanto, como se trata de una comunicación serial, los mensajes se van a recibir letra por letra. Este es uno de los ejemplos en lo que es necesario realizar dos tareas simultáneas, como se ha comentado en el apartado de metodología, porque se necesita saber qué tipo de mensaje se va a recibir desde la aplicación para poder tratarlo de la manera correcta.

Los mensajes serán tratados dependiendo de si son enviados desde la pantalla de cinemática directa, desde la pantalla de cinemática inversa o desde la de joystick. Para diferenciarlos, la primera letra del mensaje enviado será diferente para cada aplicación.

Al tratarse de comunicación serial, las letras del mensaje se irán enviando de una en una, es por eso que se utiliza una letra específica para determinar cuando comienza el mensaje. Al detectar el comienzo de un mensaje, se entra en un modo determinado y se comienza a guardar las letras del mensaje en un array. Es necesario determinar un array lo suficientemente grande para guardar todo el mensaje, por lo que se declaran tres variables para guardar el mensaje dependiendo del modo: *text_array_n[n]*, *text_array_m[m]* y *text_array_c[c_cont]*. En este caso, no es necesario un símbolo específico para cerrar el mensaje porque los contadores se reinician cuando detectan el inicio.

También se han añadido otros tres modos para controlar el movimiento de la pinza, el primero para abrir la pinza, el segundo para cerrar la pinza y el tercero para apagarla.

Por último, se utiliza un modo para realizar una puesta a cero de todas las articulaciones.

El código del manejador de la interrupción para la recepción de mensajes que incluye todo lo comentado anteriormente sería el siguiente:

```
void USART3_IRQHandler(void){
    if ((USART3->SR & USART_FLAG_RXNE) != (u16)RESET)
    {
        text = USART_ReceiveData(USART3);

       if(text == 't'){
                    n=0;
                    modo=0; } //Joystick
             else if(text == 'd'){
                    m=0;
                    modo=1; } //Cinematica directa
             else if(text == 'i'){
                    c_cont=0;
                    modo=2; } //Cinematica inversa
             else if(text == 'A'){  modo=3; } //Abrir pinza
             else if(text == 'C'){  modo=4; } //Cerrar pinza
             else if(text == 'O'){  modo=5; } //Apaga la pinza
             else if(text == 'R'){  modo=6; } //Puesta a cero

            if(modo == 0){ //Joystick
                if(n<24){
                            text_array_n[n]=text;
                            n++;
                printf("%s\n",text_array_n);
                }

                angle_bl=return_angle(text_array_n, X_value, Y_value, Z_value);
                printf("%d\n",angle_bl);

            }else if (modo == 1){ //Cinemática directa
                        if(m<35){
                            text_array_m[m]=text;
                            m++;
                            }
            }else if (modo == 2){ //Cinemática inversa
```

```
                if(c_cont<38){
                    text_array_c[c_cont]=text;
                    c_cont++;
                    }

            }else if (modo == 3){
                printf("Abrir pinza");//Abrir pinza
                         TM_PWM_Init(-90);

            }else if (modo == 4){
                printf("Cerrar pinza");
                         TM_PWM_Init(90);//Cerrar pinza


            }else if (modo == 5){
                printf("OFF pinza");
                         TM_PWM_Init(0);//OFF

            }else if (modo == 6){

            }else{modo=10;}

}}
```

La variable modo es la encargada de determinar la función que se va a ejecutar, y también se lee en el main dentro del bucle, por lo que se detalla una tabla con los valores que toma y el funcionamiento que tienen, en este caso dentro del manejador.

Valor *modo*	**Letra de inicio del mensaje**	**Modo**	**Funcionamiento dentro de USART3_IRQHandler**
0	‘t’	Joystick	Recepción mensaje
1	‘d’	Cinemática directa	Recepción mensaje
2	‘i’	Cinemática inversa	Recepción mensaje
3	‘A’	Abrir pinza	Activar señal PWM PD12 (5%)
4	‘C’	Cerrar pinza	Activar señal PWM PD12 (10%)
5	‘O’	Apagar pinza	Desactivar señal PWM
6	‘R’	Puesta a cero	Posición inicial hasta final de carrera

Tabla 12 - Mensajes y modos de recepción de mensajes del manejador USART3

Anteriormente, en el apartado 4.1.6 (Pinza herramienta), se detalla el funcionamiento de la pinza con la activación de la señal.

4.3.4. Envío de mensajes desde el microcontrolador

Para enviar mensajes desde el microcontrolador al teléfono se usa el pin PD8 en configuración de transmisor. De la misma forma, estos mensajes que se envían desde la tarjeta utilizan la comunicación serial y se envían letra por letra.

Para este cometido se usa la función *void USARTSend(const unsigned char *pucBuffer, uint8_t ulCount)* la cual tiene como entradas el mensaje a enviar y el número de bits que ocupa el mensaje. El código de esta función es el siguiente:

```
void USARTSend(const unsigned char *pucBuffer, uint8_t ulCount){
    //
    // Bucle mientras haya más carácteres por enviar
    //
    while(ulCount>0)
    {
        USART_SendData(USART3, (uint16_t) *pucBuffer++);
        /* Bucle hasta que acabe la transmisión */
        while(USART_GetFlagStatus(USART3, USART_FLAG_TC) == RESET)
        {
        }
    }
}
```

Se trata de un bucle que envía todos los bits que se le han indicado y dentro del envío se espera a que se haya enviado el bit anterior hasta enviar el siguiente.

A la hora de implementar esta función se haría de la siguiente forma:

```
USARTSend(envia_mess, sizeof(envia_mess));
```

En este caso, *envia_mess* sería el array de caracteres a enviar y la función *sizeof()* se usa para determinar el tamaño del mensaje en bits.

4.4. App móvil

Se ha creado una aplicación móvil con *MIT App Inventor*, que se encarga de controlar el robot y hacer de interfaz para el usuario. Este entorno de desarrollo utiliza lenguaje basado en *scratch*, en el que se utilizan bloques ya predefinidos para controlar la aplicación. Sin embargo, aunque parece poco complejo, tiene una gran cantidad de herramientas para realizar cualquier funcionalidad.

Este entorno de desarrollo presenta dos pestañas:

- ***Designer*:** en esta pestaña se determinan las pantallas de la aplicación, se añaden *layouts* para organizar la información y se introducen las herramientas necesarias. En este proyecto se han usado botones, tablas, *sliders*, relojes internos, cliente bluetooth, *canvas* y etiquetas para los mensajes. En la siguiente imagen se puede observar a la izquierda todas las herramientas disponibles, en el centro una representación de cómo quedaría la aplicación y a la derecha las opciones que ofrece cada herramienta.

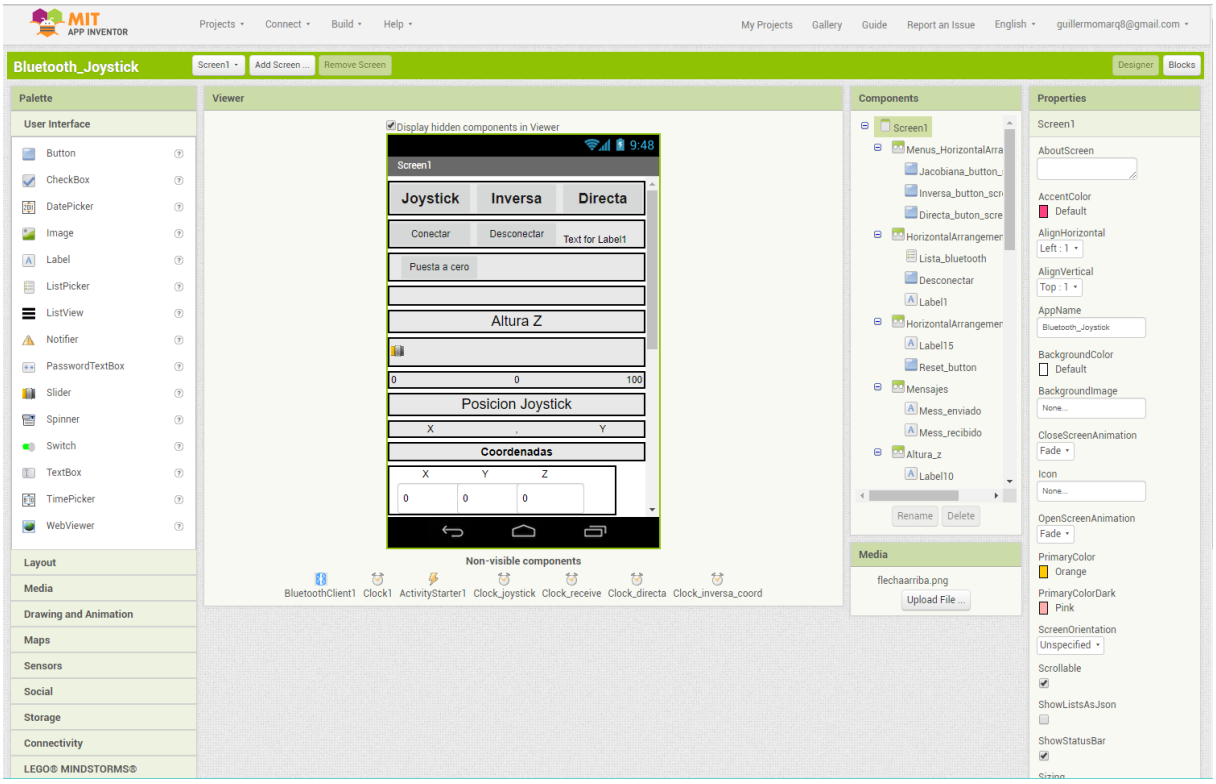


Ilustración 35 - Interfaz designer Mit App Inventor

- ***Blocks:*** en esta pestaña se pueden crear lógicas, bucles, cálculos y demás procedimientos para programar los objetos que se han dispuesto en la pestaña anterior.

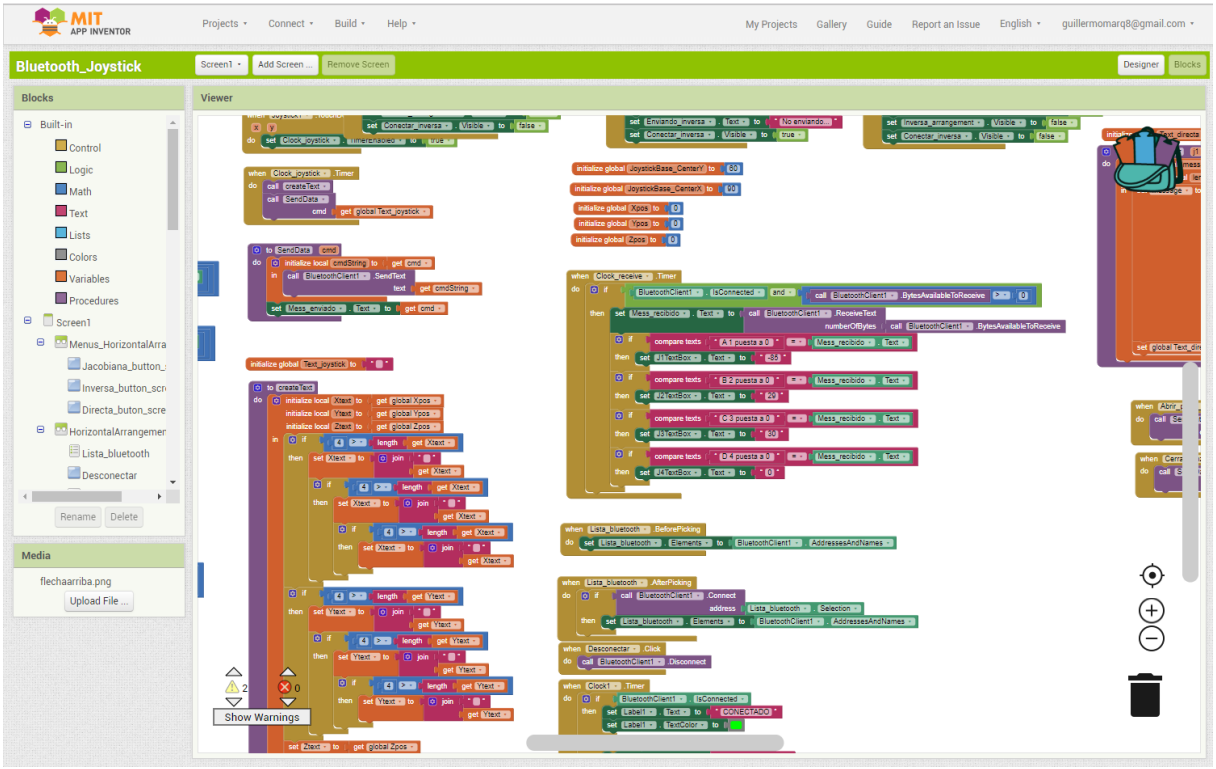


Ilustración 36 - Interfaz blocks Mit App Inventor

A continuación, se pasa a describir el funcionamiento de la aplicación y cómo se ha desarrollado.

4.4.1. Interfaz

La aplicación se ha desarrollado de forma paralela al diseño del robot, como ya se ha visto en el apartado 1.4 (Metodología de trabajo). La interfaz de la aplicación se detalla por niveles, comenzando desde el nivel superior.

- **Nivel 1:** Al disponer de tres modos de funcionamiento principalmente, se ha diseñado una pantalla para cada modo, pudiendo cambiar de modo en cualquier momento mediante unos botones dispuestos en la parte superior.

- **Nivel 2:** Bajo estos se encuentran los botones para conectarse con el módulo bluetooth. Con el botón de 'Conectar' aparece una lista de todos los dispositivos bluetooth que han sido emparejados y se puede seleccionar uno de ellos para conectarse. Al centro se encuentra el botón de 'Desconectar' y a la derecha una etiqueta que muestra el estado de la conexión con el dispositivo.

- **Nivel 3:** En un nivel inferior, se encuentra el botón de 'Puesta a cero', el cual envía al microcontrolador un mensaje para situar las articulaciones en la posición inicial.

Al pulsar este botón, el envío del mensaje se realiza con la función SendData de la siguiente forma:

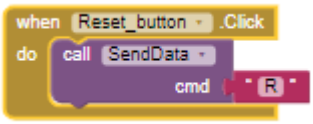


Ilustración 37 - Envío de mensaje de Puesta a cero

- **Nivel 4:** A continuación, se dispone de dos etiquetas de texto en las que aparecerán a la izquierda los mensajes enviados y a la derecha los mensajes recibidos. De esta forma, se puede asegurar el correcto funcionamiento de la aplicación por parte del usuario.

- **Nivel 5:** En este nivel se encuentra el funcionamiento de cada modo que se detalla en los siguientes apartados.

- **Nivel 6:** Este se trata del último nivel, en el que se tienen tres botones para controlar el funcionamiento de la pinza. De izquierda a derecha, el primer botón sirve para abrir la pinza, el siguiente para apagarla y el último para cerrar la pinza.

Estos tres botones, al ser pulsados, envían cada uno un mensaje que será codificado por el microcontrolador.

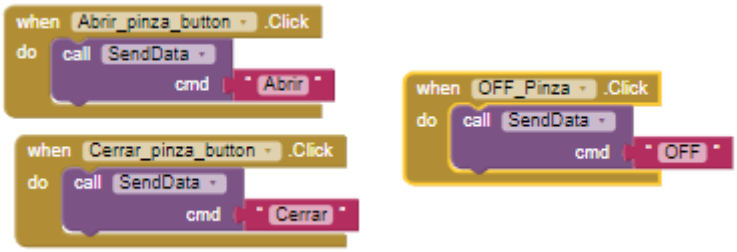


Ilustración 38 – Envío de mensajes control de la pinza

En la siguiente imagen se pueden observar todos estos niveles de la interfaz que se han detallado.

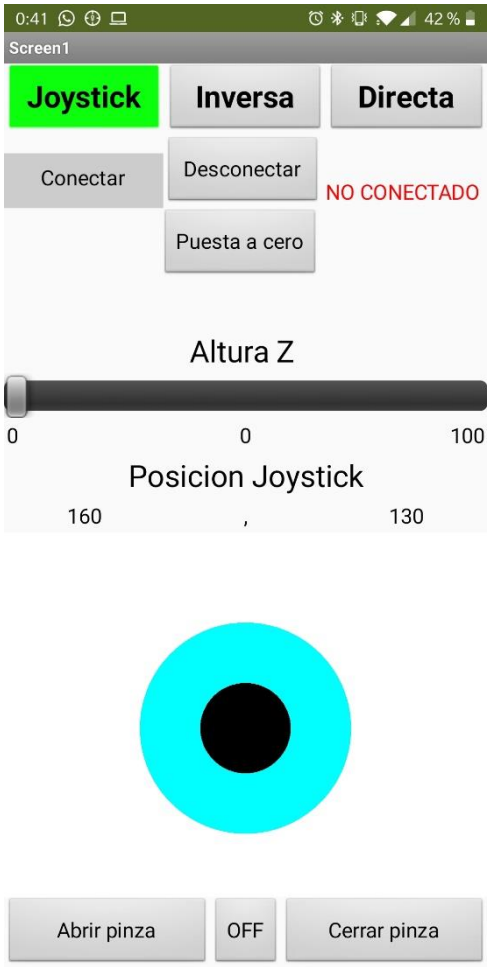


Ilustración 39 - Interfaz aplicación

4.4.2. Modo Joystick

Este primer modo, que aparece en la imagen anterior, se encargaría de controlar el movimiento de la punta del robot gracias a un joystick creado con dos imágenes que se pueden mover y un *slider* para determinar la altura. A partir de la posición determinada por el joystick, se calcularían los ángulos para el control del robot, sin embargo, como se detalla en Conclusiones, no se ha desarrollado esta parte del proyecto por falta de tiempo.

A pesar de no estar desarrollada en el microcontrolador, esta parte se ha implementado en la aplicación y se puede enviar el mensaje con la información necesaria para los posteriores cálculos en el caso de implementar este modo como futura mejora.

4.4.2.1. Interfaz

La interfaz del modo joystick se basa en dos imágenes de círculos, en las cuales el circulo interior se mueve al tocarlo mientras que el circulo mayor de encuentra fijo. A partir de este movimiento, se crea un sistema de coordenadas X/Y con el cero en el centro del circulo mayor.

Para el movimiento en el eje Z, se ha añadido un slider que sirve para modificar la altura del robot de 0 a 100. De esta forma, al mover el circulo y el slider, se obtienen unas coordenadas que se enviarán al microcontrolador.

Como se puede observar en la siguiente imagen, las coordenadas enviadas se muestran bajo el slider y sobre el joystick para luego ser transformadas en un mensaje para enviar a la tarjeta como se observa en la etiqueta de enviar mensaje.

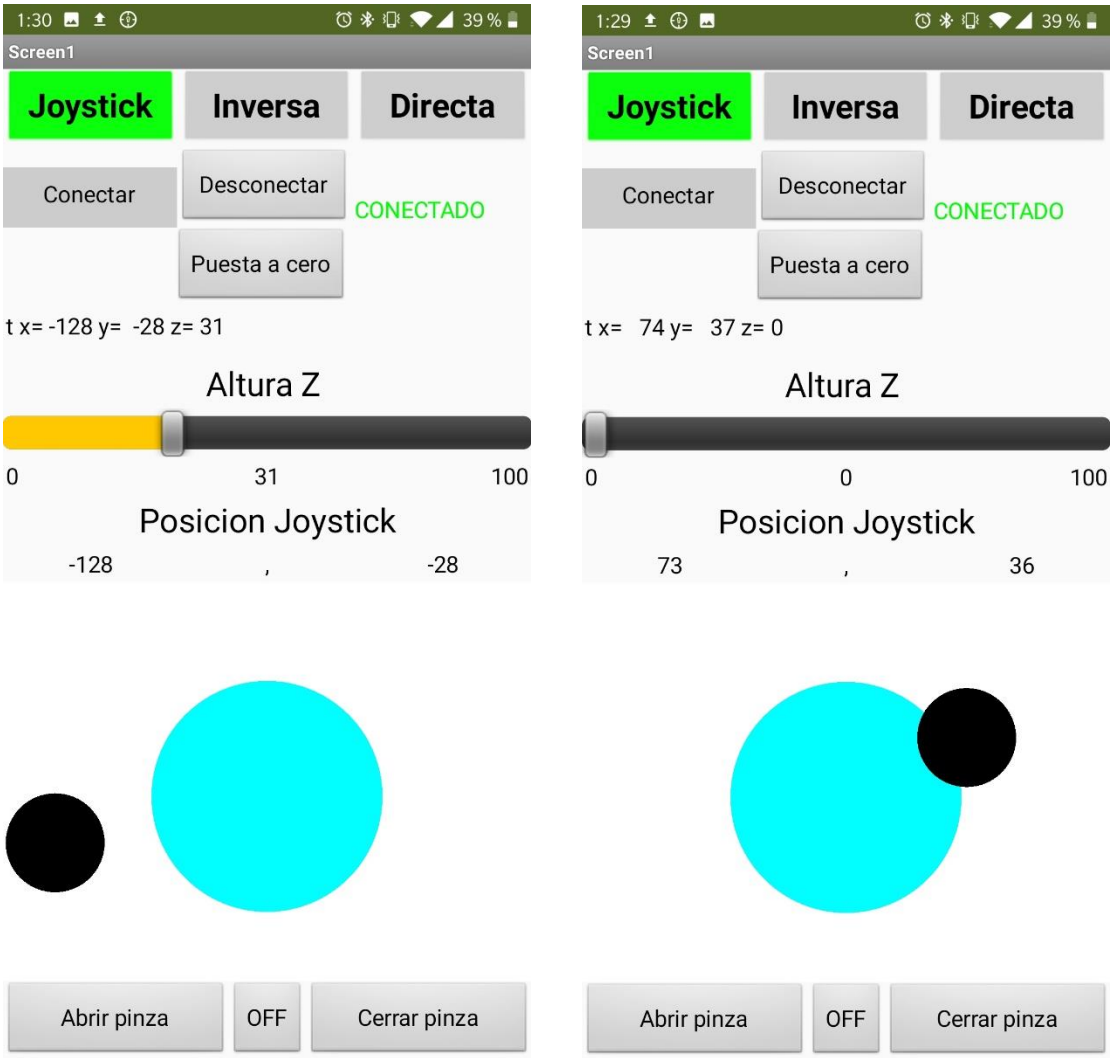


Ilustración 40 - Coordenadas en modo joystick

4.4.2.2. Programación

En primer lugar, al pulsar el botón *Joystick*, se deshabilitan todos los objetos que no sean relativos a este menú y se habilitan el joystick, el slider y las etiquetas para observar las coordenadas. Esto se realiza con el siguiente bloque:

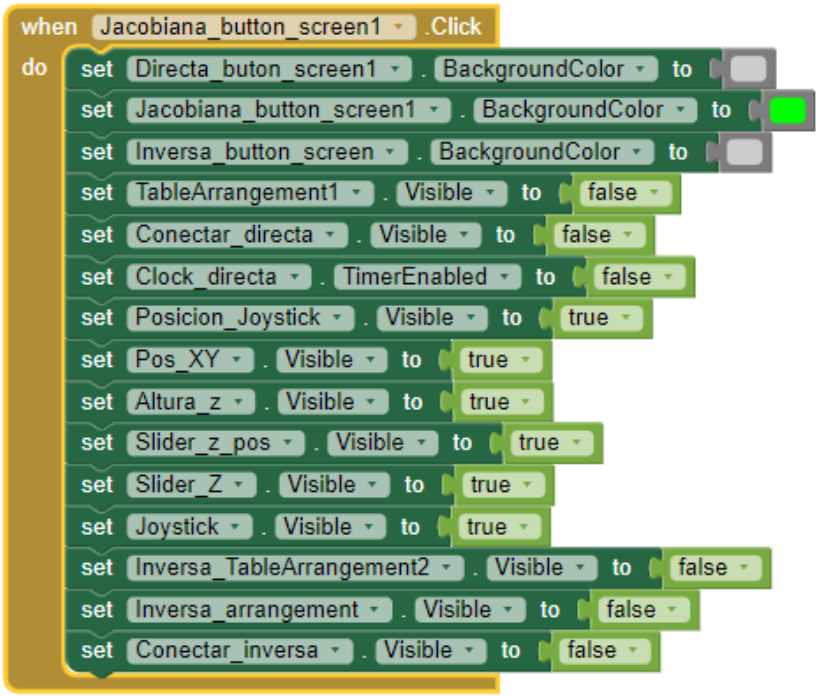


Ilustración 41 - Programación menú Joystick

Una vez dentro de la pantalla *Joystick*, se declaran las siguientes variables globales para guardar los datos de las coordenadas:

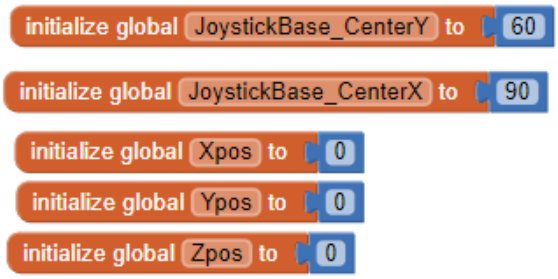


Ilustración 42 - Variables globales Joystick

Estas variables cambiarán a medida que se mueva el círculo pequeño respecto el grande, por lo que se usa el siguiente bloque para mover el círculo y calcular el valor de las coordenadas mientras el circulo esté pulsado y moviéndose.

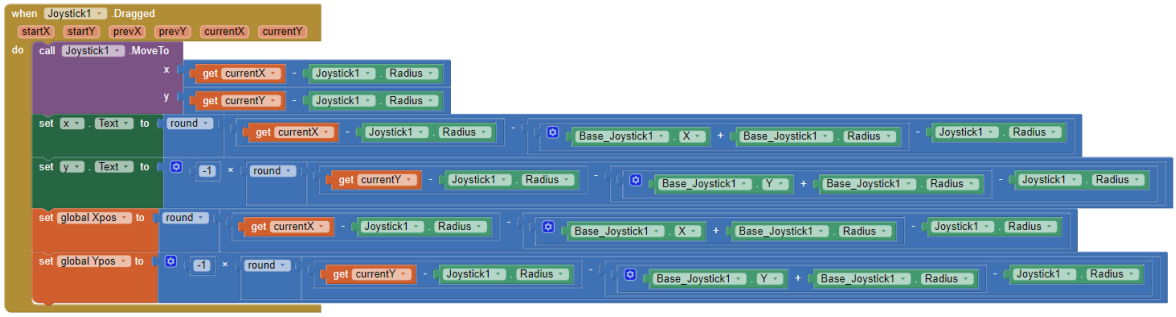


Ilustración 43 - Joystick dragged

También se ha programado un reloj interno que envía el mensaje cada 200 milisegundos y se activa al pulsar el joystick o cuando se desplaza el slider Z, por lo que sólo se envía información cuando hay un cambio de posición para evitar la saturación del módulo de bluetooth. En la siguiente imagen se observan los casos en los que se activa el reloj:

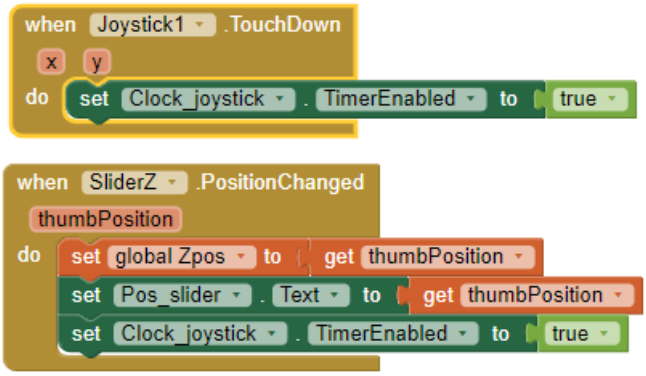


Ilustración 44 - Activación reloj interno joystick

Siguiendo con el joystick, al soltar el círculo pequeño, este se moverá a la posición central, pero se guardará la última posición en la que se encontraba, de esta manera el robot no se moverá a la posición inicial cada vez que se suelte el *joystick*. También se desactivará el reloj interno para evitar la saturación. En la siguiente imagen se observa el bloque que se utiliza:

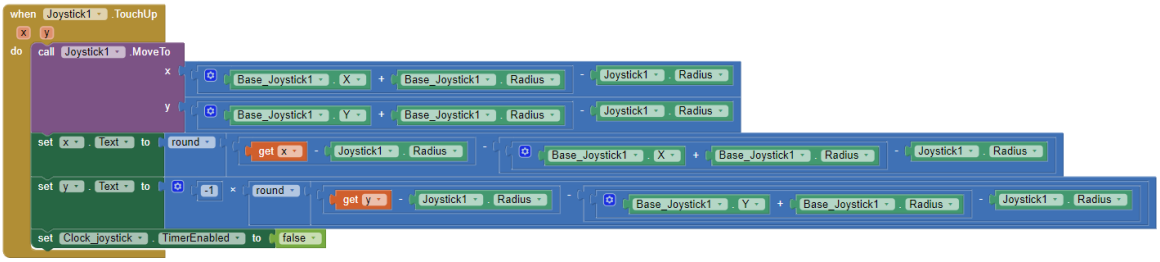


Ilustración 45 - Soltar joystick

Con las coordenadas obtenidas, se procede a crear el mensaje con la función *createText*, la cual se encarga de concatenar los datos con los espacios necesarios para que luego el mensaje pueda ser leído fácilmente por las funciones del microcontrolador.

Con el mensaje creado, se pasa a la función *SendData*, la cual envía el mensaje introducido y lo muestra por la pantalla.

Estas dos funciones se ejecutan con cada cuenta del reloj interno del joystick como se muestra en la siguiente imagen:

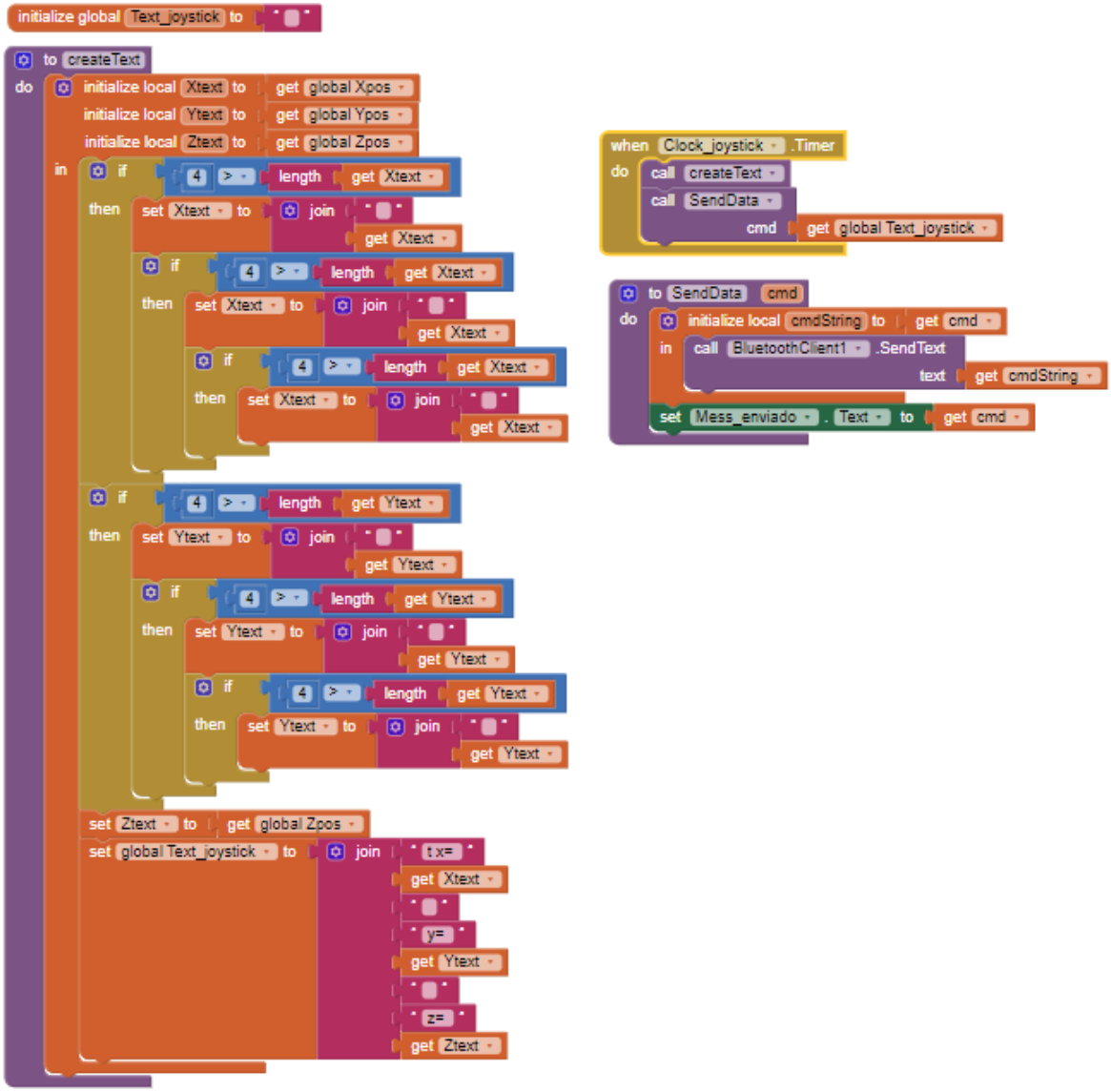


Ilustración 46 - Creación mensaje Joystick

4.4.3. Modo Cinemática Inversa

4.4.3.1. Interfaz

En este modo, se pueden observar tres TextBox para introducir cada una de las 3 coordenadas (x,y,z) que serán las que se enviarán al microcontrolador donde se realizarán los cálculos de la cinemática inversa del robot.

Una vez introducidas las coordenadas, se pulsará el botón 'Enviar mensaje', que se encuentra sobre los botones para controlar la pinza, para enviar las coordenadas. Este botón se encarga de habilitar o deshabilitar el reloj interno que envía el mensaje para evitar la saturación o el envío de datos erróneos. La interfaz de este modo quedaría de la siguiente forma:

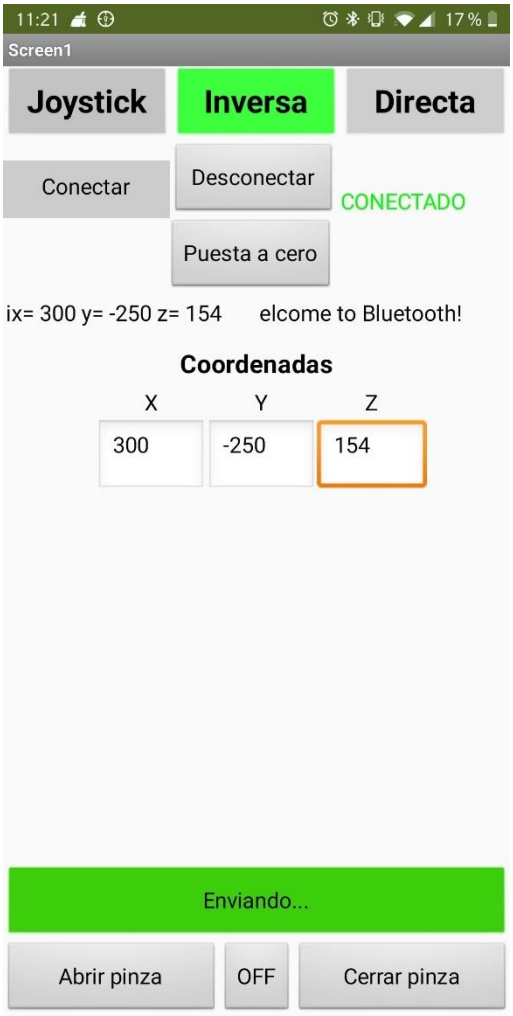


Ilustración 47 - Interfaz modo cinemática inversa

4.4.3.2. Programación

Siguiendo un procedimiento parecido al descrito en el apartado 4.4.2.2 (Programación), primeramente, al pulsar el botón 'Inversa' para cambiar de modo se deshabilitan los objetos no necesarios en este modo y se habilitan los siguientes:

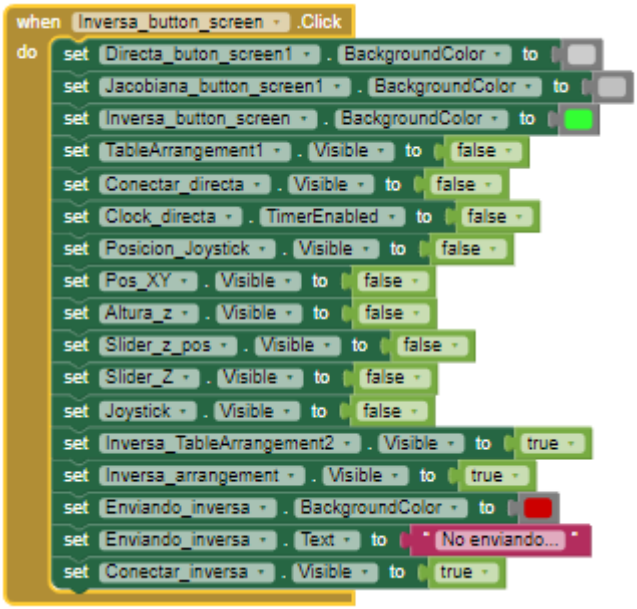


Ilustración 48 - Programación interfaz modo Inversa

Al tratarse de una interfaz más sencilla, se utilizarán los bloques de la siguiente imagen:

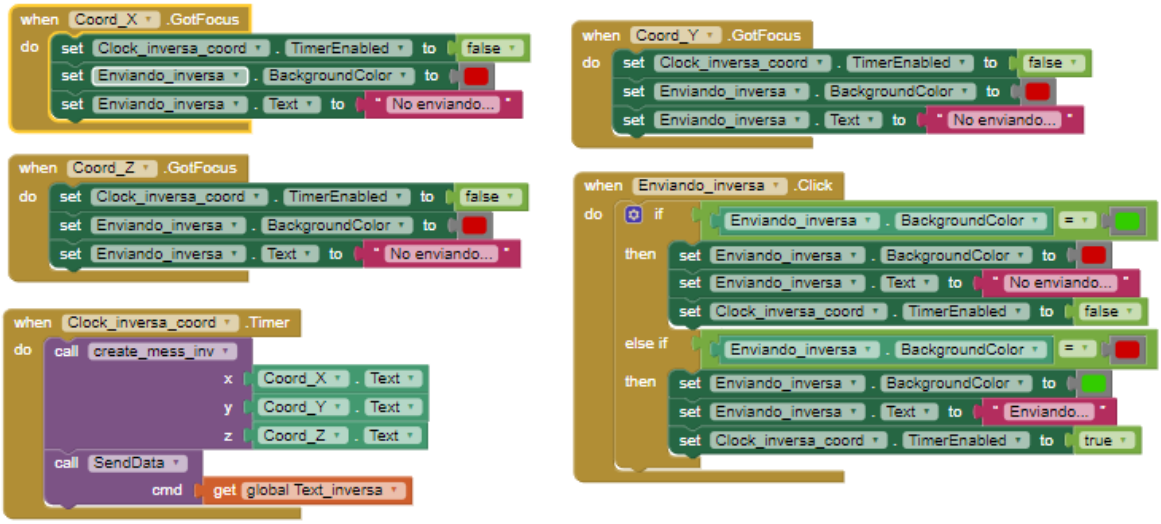


Ilustración 49 - Programación modo Inversa

El funcionamiento será el siguiente:

En las cajas de texto se pueden introducir las coordenadas que se desee, pero cuando se pulsa una de las cajas, se desactiva el envío del mensaje para evitar la lectura de datos erróneos mientras se introduce el dato.

A continuación, al pulsar el botón de enviar mensaje, se cambia el color y el texto de este botón y se habilita el reloj que se encarga de enviar estos mensajes cada 200 milisegundos. Si se pulsa otra vez el botón, cambiará al estado contrario y dejará de enviar mensajes.

Si el reloj está habilitado, se llama a la función *create_mess_inv*, que se encarga de crear el mensaje con la información de las coordenadas introducidas y con la función *SendData* se envía el mensaje. La función create_mess_inv funciona de la siguiente forma:

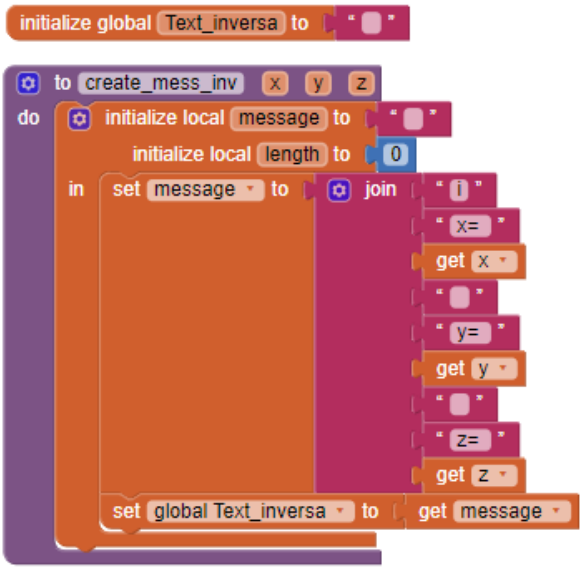


Ilustración 50 - Crear mensaje menú inversa

4.4.4. Modo Cinemática Directa

En este modo, se le envía al microcontrolador el ángulo de cada articulación directamente, por lo que el funcionamiento es muy similar al modo Cinemática Inversa.

4.4.4.1. Interfaz

Esta interfaz consta de una tabla donde se presenta para cada articulación: una etiqueta, una caja de texto, un botón para aumentar el ángulo y otro para disminuirlo. Cabe destacar que, al igual que en el modo de cinemática inversa, en las cajas de texto solo se pueden introducir números positivos o negativos para evitar posibles fallos como se muestra en la siguiente imagen.

Este modo también presenta en la parte inferior un botón que controla el envío de los mensajes.

En la imagen se puede observar que hay cuatro articulaciones, sin embargo, el robot tiene tres, esto es debido a la posibilidad de aumentar el número de articulaciones como posible futura mejora.

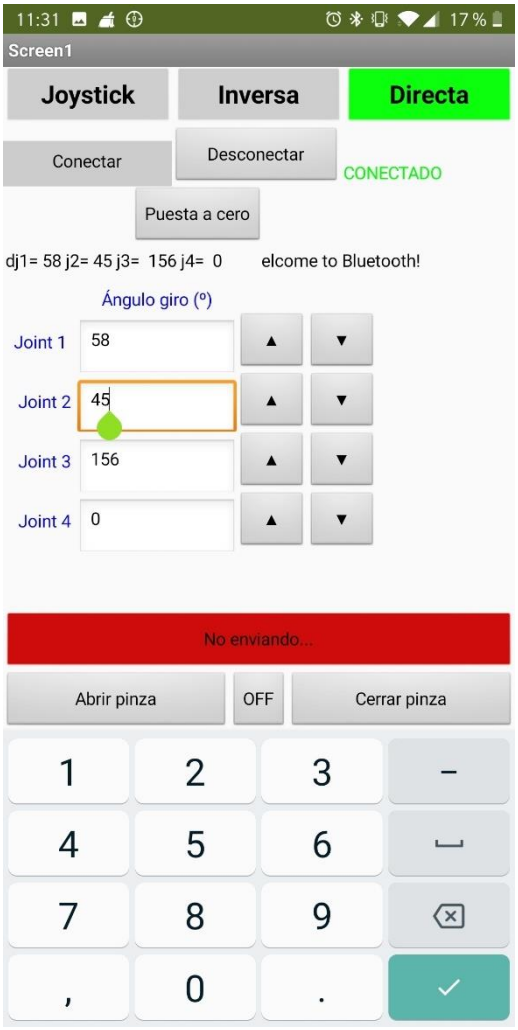


Ilustración 51 - Interfaz modo Cinemática directa

4.4.4.2. Programación

Al igual que la interfaz, la programación interna de estos objetos es similar a la de cinemática inversa.

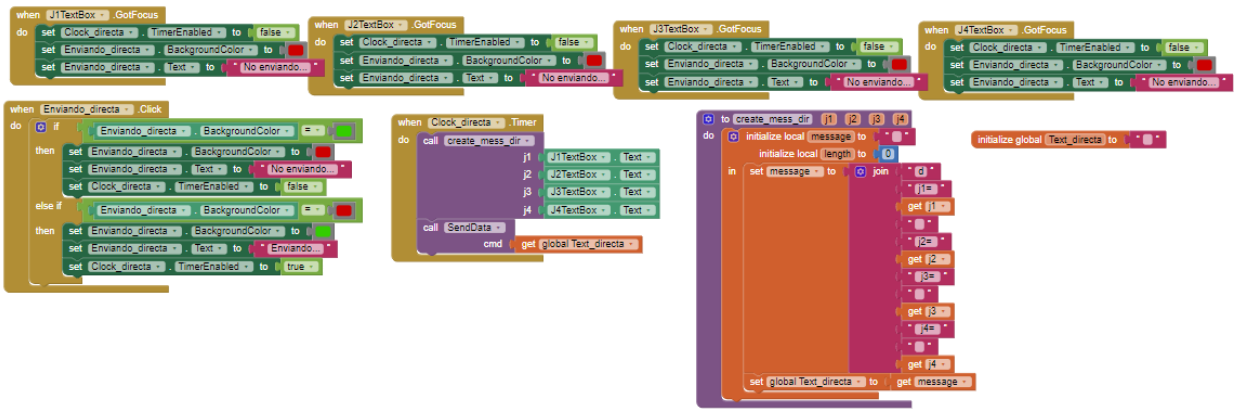


Ilustración 52 - Programación modo cinemática directa

Como se puede observar en la imagen, los cuatro primeros bloques son los encargados de bloquear el envío de mensajes en el caso de estar introduciendo valores en las cajas de texto.

En la segunda fila, el primero empezando por la izquierda es el que se encarga del funcionamiento del botón de enviar mensaje, idéntico al del modo de inversa. El segundo es el funcionamiento del reloj, en el que se crea el mensaje y se envía según los datos de las cajas de texto. El tercero es la función usada para crear el mensaje y el último es la declaración de la variable donde se guarda el texto.

4.4.4.3. Recepción de mensajes

En este modo, se ha añadido una funcionalidad en la que se recibe un mensaje desde el microcontrolador cuando el final de carrera ha sido pulsado y se cambia el valor de las articulaciones en la aplicación para observar el ángulo en el que se encuentran. Esto se realiza con el siguiente bloque:

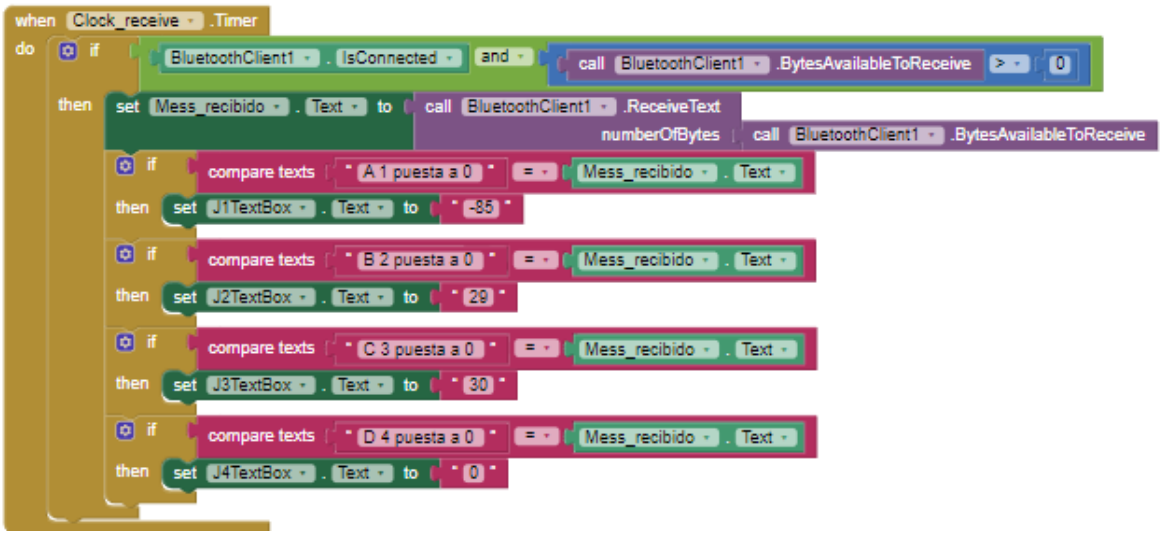


Ilustración 53 - Recepción de mensajes

En la imagen se puede observar que se utiliza un reloj interno, y en cada ciclo del reloj se comprueba si hay bytes para recibir y el bluetooth está conectado. En el caso de que se cumpla esta condición, se imprime por pantalla el mensaje recibido y dependiendo del mensaje se determina el ángulo de cada articulación. Estos ángulos corresponden al ángulo real de cada articulación respecto al suelo.

4.5. Funcionamiento general

El funcionamiento del robot se basa en dos modos principales:

- **Cinemática Inversa:** en este modo, el robot se posiciona de una determinada manera para que la punta de la herramienta se sitúe en la posición indicada por las coordenadas que se envían desde la app.
- **Cinemática Directa:** en este modo, a cada articulación se le indica el ángulo de giro determinado y se puede controlar cada articulación por separado.

Todo el robot se controla desde la app del móvil, por lo que en el siguiente diagrama, se pueden observar funcionamiento del programa, y las comunicaciones entre la app y el microcontrolador.

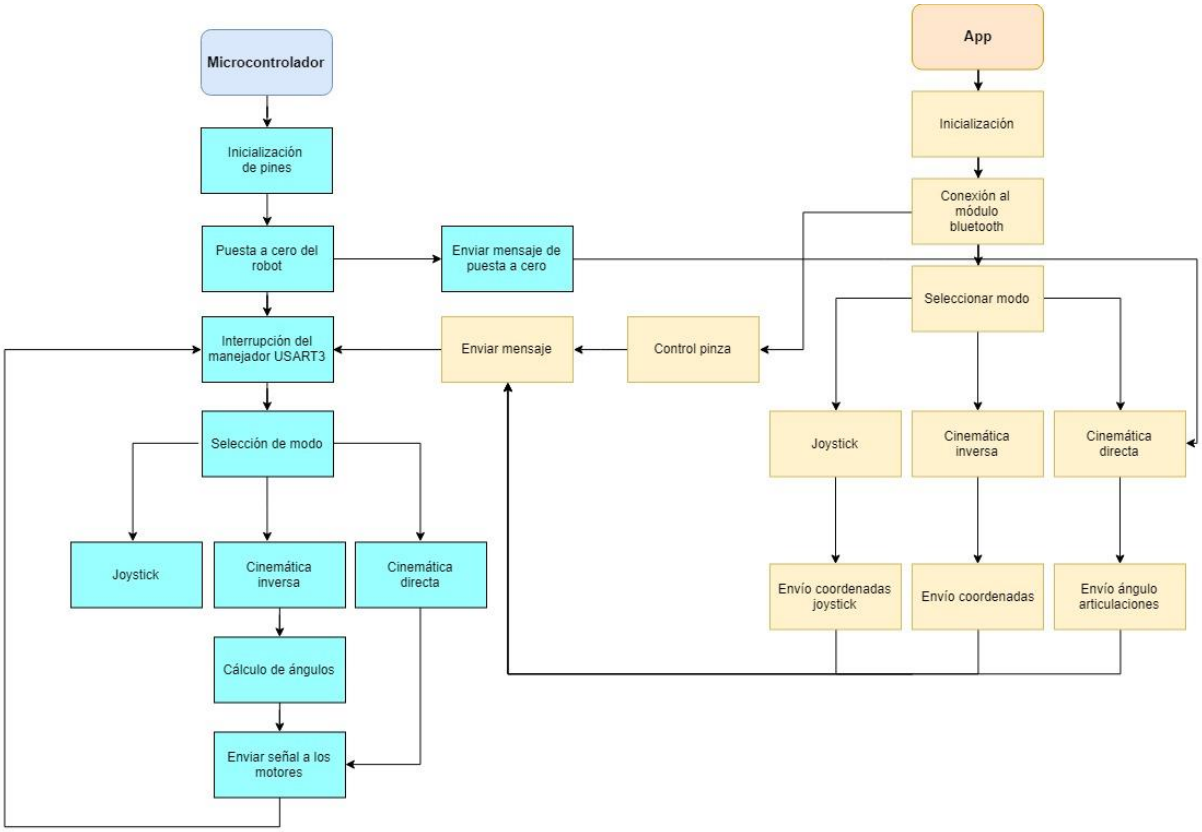


Ilustración 54 - Diagrama funcionamiento programa

En la parte del diagrama de color azul, que corresponde al funcionamiento del programa del microcontrolador, se puede observar que la parte de Joystick no hace nada. Esto es debido a que no ha sido posible desarrollar esta parte del proyecto debido a la falta de tiempo, por lo que se comenta como futura mejora en el apartado de Conclusiones.

4.5.1. Inicialización

A continuación, se adjunta el código del programa principal del microcontrolador, sobre el que se irá explicando el funcionamiento de cada modo y de cada función por partes:

```
int main(void){

    //Inicialización de pines
     pin_inicializar();

    //Configuración de las interrupciones de final de carrera
    interrupt_config();

    //Inicialización comunicación bluetooth
    usart_rxtx();

    //Inicialización de la señal PWM para el servo
    TM_TIMER_Init();
```

La primera parte es la inicialización de los pines necesarios para el funcionamiento del robot. En la función *pin_inicializar()*, se configuran los pines encargados del control de los motores como salidas, los pines para leer los finales de carrera como entradas y el pin para controlar el servo como función alternativa de señal PWM. En los anexos se puede observar con más detalle estas configuraciones.

En segundo lugar, se configuran las interrupciones de los finales de carrera con la función *interrupt_config()*. Se trata de una configuración en la que se crea un manejador para cada interrupción como ya se ha explicado en el apartado 4.1.5.2 (Implementación en C).

En tercer lugar, se declara la función *usart_rxtx()*, donde va incluida la función *bluetooth_init()*, explicada en el apartado 4.3.2 (Comunicación del módulo bluetooth con el microcontrolador), donde se configuran los pines para la comunicación serial con el módulo bluetooth y se inicia el manejador de la interrupción USART3 para la recepción de mensajes.

En cuarto y último lugar, se inicializa la señal PWM para el funcionamiento del servo con la función *TM_TIMER_Init()* extraída de las librerias de Tilen Majerle [10].

Después de las inicializaciones, se llevan los motores a la posición inicial, la cual es hasta que se pulse el final de carrera. Esto se realiza para obtener una referencia inicial del ángulo en el que se encuentra cada articulación para poder controlarlos de manera precisa.

```
    //Puesta a cero inicial
    angle_ant_J1=stepper1(-360,0,25);
    angle_ant_J3=stepper3(-360,0,100);
    angle_ant_J2=stepper2(-360,0,100);
```

4.5.2. Joystick

Una vez se han realizado todas las declaraciones necesarias se pasa al bucle while, donde se programa el funcionamiento de cada modo. Dependiendo del valor de la variable *modo*, la cual cambia su valor dentro del manejador de la comunicación serial USART3, se entrará en modo *Joystick*, modo *Cinemática inversa* o modo *Cinemática directa*.

Como ya se ha comentado antes en el apartado 4.4.2 (Modo Joystick), en la parte del Joystick se pensaba realizar un control de la posición de la punta del robot mediante la jacobiana pero debido a la falta de tiempo, se han centrado los esfuerzos en conseguir construir un prototipo funcional con cinemática inversa y cinemática directa.

Pese a no haber desarrollado la jacobiana, en el modo de joystick dentro del bucle, se ha añadido una función que devuelve el ángulo de la arcotangente de los valores las coordenadas X e Y que se obtienen desde la aplicación y se pasa este valor a las tres articulaciones.

A continuación, el código correspondiente al modo Joystick:

```
  while (1) {

       pulsador_PA0=GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0);

      //Modo joystick
      if(modo==0){
           if(pulsador_PA0==1){

           GPIO_ResetBits(GPIOD, GPIO_Pin_12);
           GPIO_ResetBits(GPIOD, GPIO_Pin_13);
           GPIO_ResetBits(GPIOD, GPIO_Pin_14);

                      angle_ant_J1=stepper1(angle_bl,angle_ant_J1,50);
                      angle_ant_J3=stepper3(angle_bl,angle_ant_J3,50);
                      angle_ant_J2=stepper2(angle_bl,angle_ant_J2,50);

           delay_ms(200);
       }
```

4.5.3. Cinemática directa

El siguiente modo disponible es el de cinemática directa, en el que se entra cuando la variable *modo* tiene un valor de 1. En este caso, se utiliza la variable c para guardar el array con los valores de ángulo de las tres articulaciones devuelto por la función *return_angle_directa (text_array_m)*. Dentro de esta función, el código es el siguiente:

```
  int16_t *return_angle_directa(char text_to_convert[]){
//esta funcion lee los valores recibidos de la cinematica directa
    char j1[3],j2[2],j3[2],j4[2];
    static int16_t angles[4];
    int ang_j1,ang_j2, ang_j3, ang_j4;
    sscanf(text_to_convert,"%s %d %s %d %s %d %s %d", j1, &ang_j1, j2, &ang_j2,
     j3, &ang_j3, j4, &ang_j4);
    printf("%s\n",text_to_convert);
    angles[0]=ang_j1;
    angles[1]=ang_j2;
    angles[2]=ang_j3;
    angles[3]=ang_j4;
    return angles;
}
```

Esta función, recoge el mensaje recibido por el módulo bluetooth y extrae los valores de las articulaciones mediante un *sscanf* para guardarlos en un *array*.

Una vez se obtienen los valores, se pasan directamente a la función *stepper1*, *stepper2* y *stepper3* para que se encarguen del giro de las articulaciones. A continuación, se puede observar el código del modo cinemática directa:

```
      }else if(modo==1){
         //Cinematica DIRECTA
```

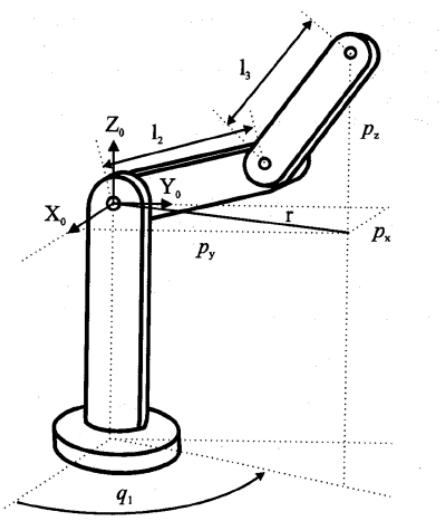
```
c=return_angle_directa(text_array_m);
printf("%d %d %d %d \n", c[0], c[1], c[2], c[3]);
angle_ant_J1=stepper1(c[0],angle_ant_J1,vel_stepper1);
angle_ant_J2=stepper2(c[1],angle_ant_J2,vel_stepper2);
angle_ant_J3=stepper3(c[2],angle_ant_J3,vel_stepper3);

printf("ang2= %d ang3= %d\n", angle_ant_J2, angle_ant_J3);
delay_ms(200);
```

4.5.4. Cinemática inversa

El siguiente modo codificado es el de cinemática inversa, en el cual se realiza el cálculo de los ángulos de las articulaciones según la posición en coordenadas enviada desde la *app*.

De forma muy parecida a la realizada en la cinemática directa, se utiliza la función *return_angle_inversa(text_array_c)* para extraer la información del mensaje recibido por bluetooth. Una vez extraída, se procede con el cálculo de la cinemática inversa del robot de tres articulaciones mediante trigonometría.

Basándose en los cálculos de un robot de tres articulaciones expuestos en la siguiente imagen (Ilustración 55), se realizan algunos cambios en las fórmulas para adaptarlos al robot del proyecto.



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

$$p_x{}^2 + p_y{}^2 + p_z{}^2 = l_2^2 + l_3^2 + 2l_2l_3\cos q_3$$

$$\cos q_3 = \frac{p_x{}^2 + p_y{}^2 + p_z{}^2 - l_2^2 - l_3^2}{2l_2l_3}$$

$$senq_3 = \pm\sqrt{1-\cos^2 q_3}$$

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

Ilustración 55 - Cálculos cinemática inversa 1

Estos cambios se realizan debido a que se ha situado el origen de coordenadas en la base del robot y a que en la articulación 1, existe una distancia Ly en el eje Y (según la imagen anterior) desde el eje de rotación hasta la punta de la herramienta. Los cambios en los cálculos son los siguientes:

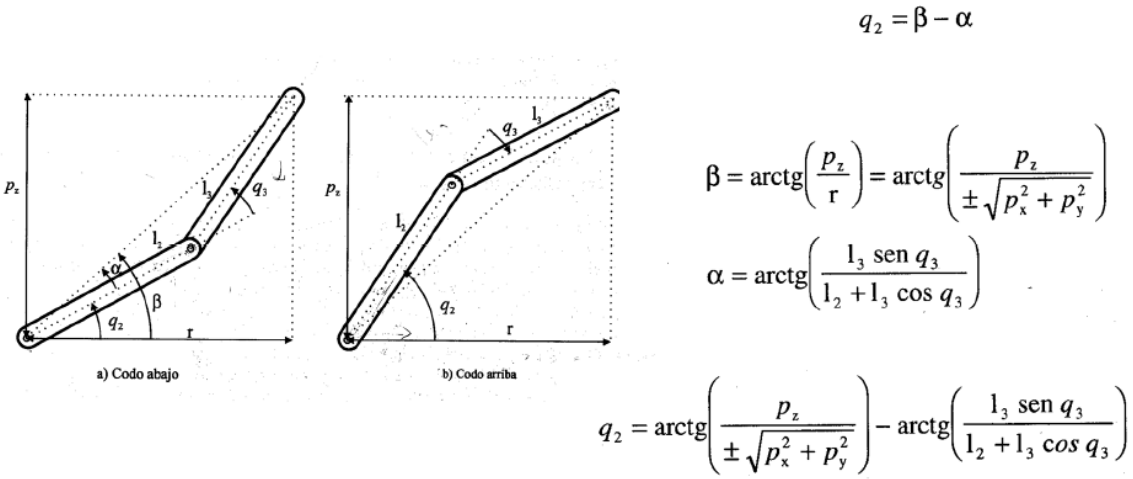
Ecuación 18 - Cálculos cinemática inversa

$$r = \sqrt{px^2 + py^2 - Ly^2}$$

$$q_1 = \tan^{-1}(\frac{py}{px}) + \tan^{-1}(\frac{Ly}{r})$$

$$pz = pz - L1$$

En esta segunda parte de los cálculos, mostrada en la siguiente imagen (Ilustración 56), no se ha realizado ningún cambio.



$$q_2 = \beta - \alpha$$

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

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

$$q_2 = \text{arctg}\left(\frac{p_z}{\pm\sqrt{p_x^2 + p_y^2}}\right) - \text{arctg}\left(\frac{l_3 \text{ sen } q_3}{l_2 + l_3 \cos q_3}\right)$$

Ilustración 56 - Cálculos cinemática inversa 2

Una vez se tienen los cálculos, se implementan en el código del microcontrolador mediante las siguientes funciones: *q1_inversa*, *q2_inversa* y *q3_inversa*. Estas funciones no tienen mayor complicación que la de realizar los cálculos de las ecuaciones anteriores gracias a la librería *math.h* por lo que se incluye el código en Anexo I: Código microcontrolador.

El siguiente paso dentro del modo de cinemática inversa, es el de enviar estos ángulos a cada articulación y realizar el movimiento mediante las funciones *stepper1*, *stepper2* y *stepper3*, por lo que el código quedaría de la siguiente forma:

```
}else if(modo==2){
        //Cinematica INVERSA
         if(pulsador_PA0==1){

             printf("%s\n", text_array_c);
             c=return_angle_inversa(text_array_c);
             printf("%d %d %d %d \n", c[0], c[1], c[2], c[3]);

        //Se calculan los angulos
            q1=q1_inversa(c[0],c[1],c[2]);
            q3=q3_inversa(c[0],c[1],c[2]);
            q2=q2_inversa(c[0],c[1],c[2],q3);

            printf("ang1= %f ang2 = %f  ang3 = %f\n",q1,q2,q3);
            delay_ms(1000);
```

```
            angle_ant_J1=stepper1(q1,angle_ant_J1,25);
            angle_ant_J3=stepper3(180-q3,angle_ant_J3,50);
            angle_ant_J2=stepper2(q2,angle_ant_J2,50);

        delay_ms(200);

      }
```

4.5.5. Puesta a cero

Por último, el modo de puesta a cero se realiza cuando se pulsa en la aplicación el botón de puesta a cero y valor de la variable *modo* cambia a 6. Cuando esto pasa, se mueven los motores hasta que se pulsan los correspondientes finales de carrera. El código de esta parte sería el que se expone a continuación:

```
} else if(modo==6){

        //Modo de puesta a cero
        angle_ant_J1=stepper1(-360,0,25);
        angle_ant_J2=stepper2(-360,0,100);
        angle_ant_J3=stepper3(-360,0,100);
         }
     }
   return(0);
}
```

4.6. Ensamblaje final del robot

Para el ensamblaje del robot articulado se necesitan los siguientes materiales:

- Piezas impresas en 3D
- Rueda giratoria
- Fuente de alimentación
- Drivers DRV8825
- Placas PCB
- LM317T
- Componentes electrónicos: resistencias, condensadores, cables…
- Placa STM32F4 Discovery
- Motor paso a paso Nema 17 x 2
- Motor paso a paso Nema 24
- Finales de carrera
- Módulo bluetooth HC-05
- Servomotor
- Tornillería

4.6.1. Electrónica

Para la parte electrónica, en primer lugar, se soldarán los componentes a las placas PCB para evitar ruido y que se suelten los cables.

Se crean 3 PCBs: la primera con dos drivers DRV8825; la segunda con un driver y los pines de GND y VCC; y la última con el regulador de voltaje para el servomotor.

Se procederá colocando los componentes por la parte superior de la placa y soldando por la parte inferior como se muestra en las siguientes imágenes:

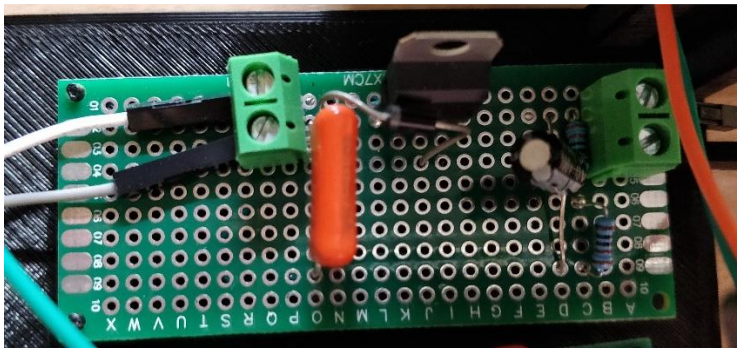


Ilustración 57 -PCB regulador de tensión servomotor

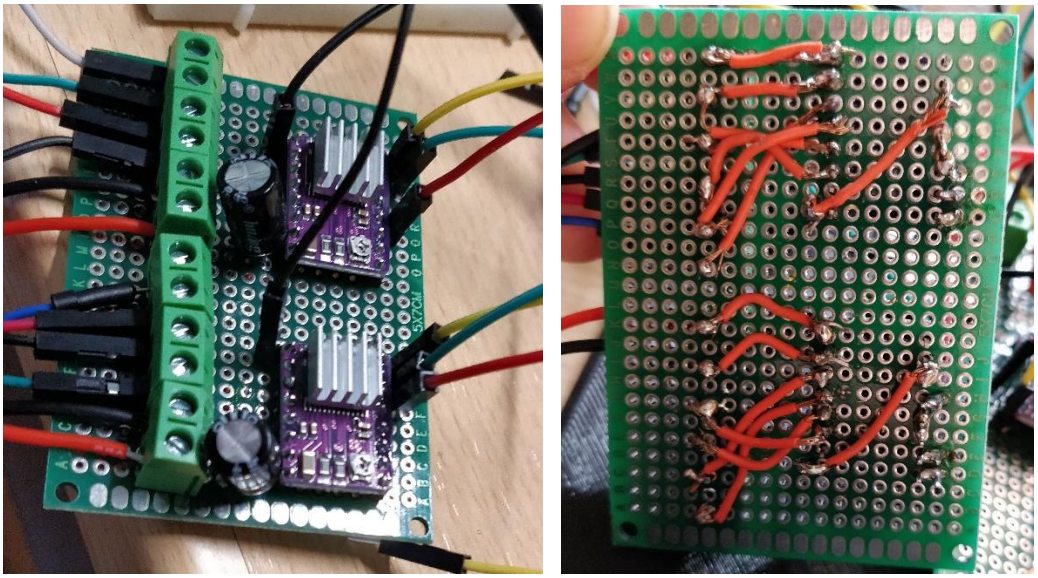


Ilustración 58 - PCB drivers DRV8825

Una vez realizada la soldadura, se procede a introducir las placas en la caja de encapsulado que se ha impreso en 3D. Esta caja dispone de pines para asegurar que las PCBs no se mueven fácilmente.

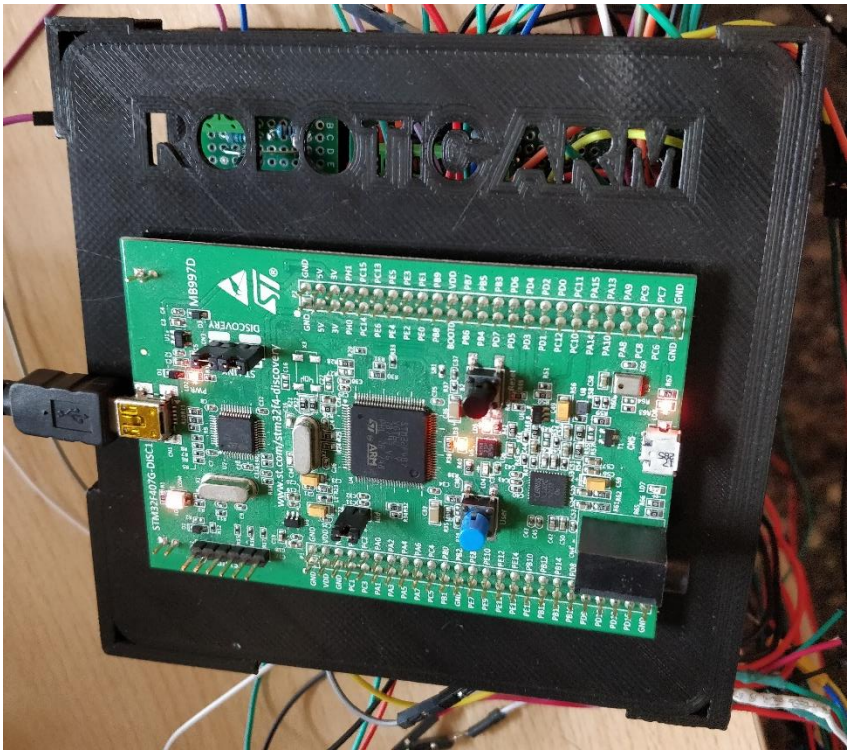


Ilustración 59 - Encapsulado circuitos

Por último, se conectan los pines como se muestra en el siguiente esquemático:

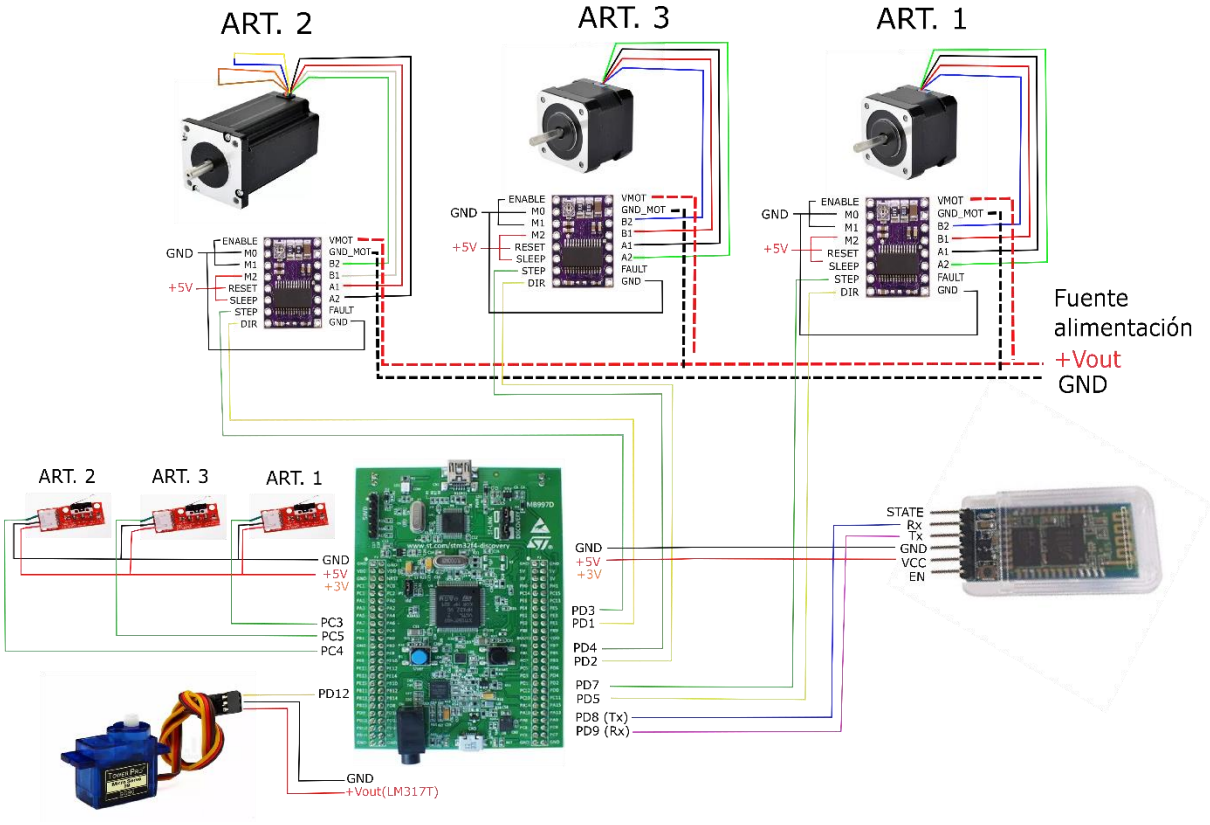


Ilustración 60 - Esquemático conexiones proyecto

4.6.2. Montaje brazo

Para el montaje del brazo articulado, se comenzará con la pieza base soporte atornillando la rueda, modificada como se explica en el apartado 4.1.2.1Articulación 1Articulación 1), en esta pieza con 4 tornillos M5 y 4 tuercas. Entre la rueda y la base irá atornillada la pieza soporte final de carrera 1 como indica el apartado 4.2.3.2 (Soporte final de carrera articulación 1).

A continuación, se atornillará la pieza enganche engranaje junto con el engranaje y la base para articulación 2 con tornillos y tuercas M5 como se muestra en la siguiente imagen:

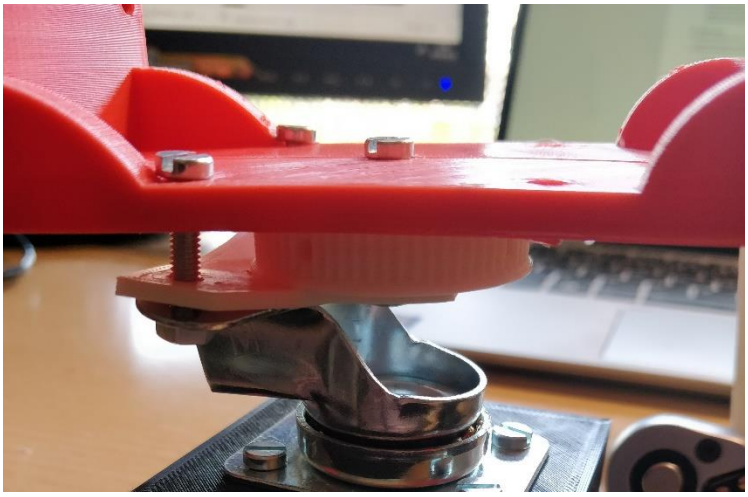


Ilustración 61 - Ensamblaje engranaje

El siguiente paso es montar el motor de la articulación 2, con el soporte Nema 24 y el soporte final de carrera articulación 2 con tornillos y tuercas M5.

A continuación, se pasa a montar el brazo comenzando por la pieza articulación 2, atornillando el *mounting hub* para el motor Nema 24 en un extremo, y en el otro extremo el final de carrera de la articulación 3 junto con el motor 3 como se muestra en la siguiente imagen:

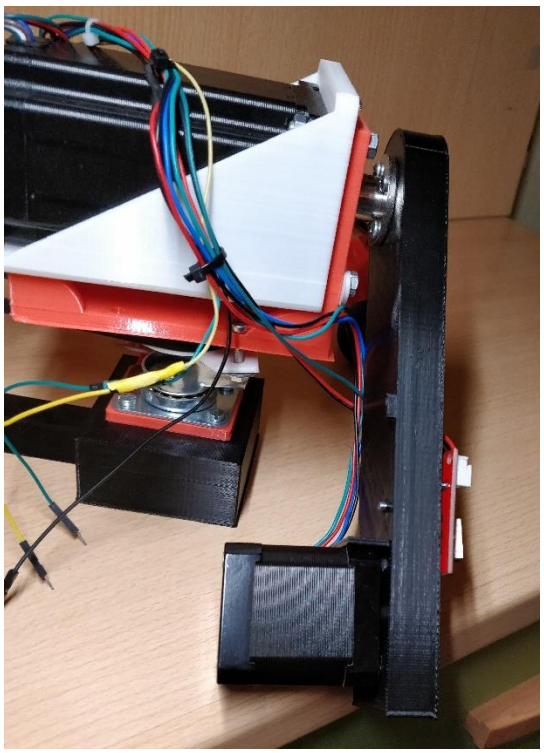


Ilustración 62 - Brazo articulación 2 ensamblado

Por último, se atornilla el mounting hub para el motor Nema 17 y se monta la herramienta como se muestra en el apartado 4.2.3.11 (Herramienta pinza servo).

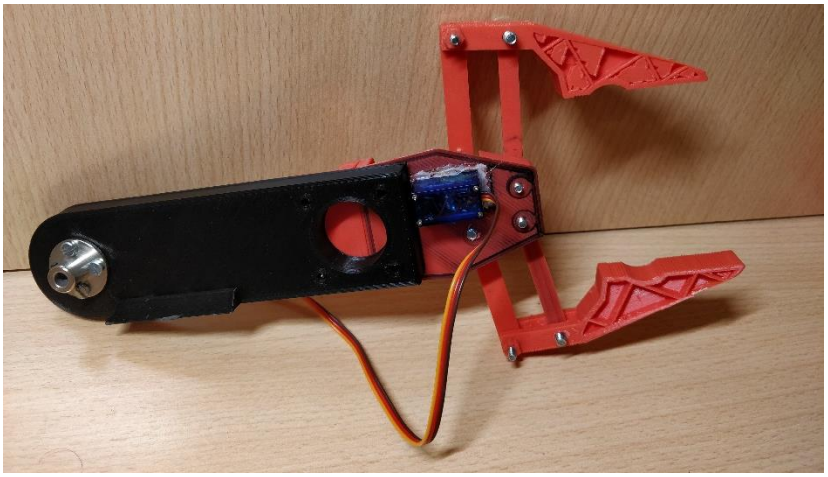


Ilustración 63 - Brazo articulación 3 ensamblado

Una vez montadas todas las partes, el prototipo del robot quedaría de la siguiente forma:

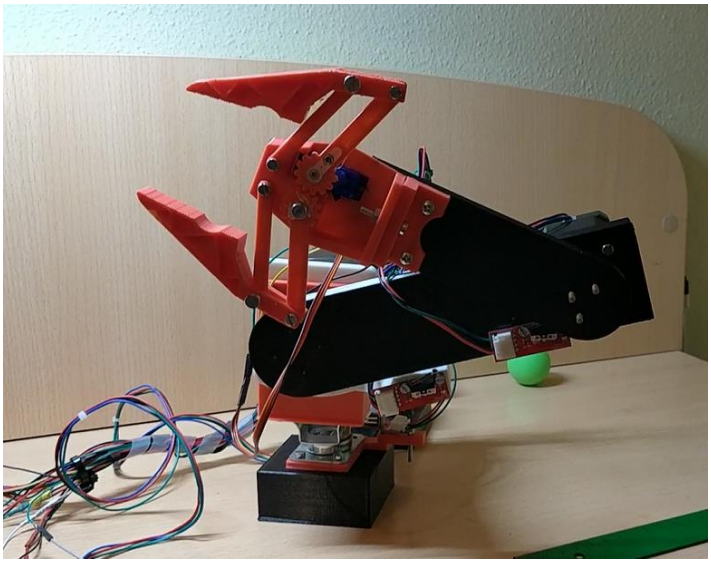


Ilustración 64 - Robot ensamblado

4.7. Pruebas de funcionamiento

El último paso del desarrollo del proyecto es realizar pruebas de funcionamiento para comprobar si realmente se han cumplido los objetivos y funciona acorde a estos. Las pruebas se realizan para los dos modos de funcionamiento con velocidades bajas para evitar roturas o fallos del sistema

4.7.1. Cinemática directa

Con el robot ensamblado y conectado a la fuente de alimentación, se carga el programa en la tarjeta, lo que inicia el funcionamiento de este.

Se observa como el robot comienza a moverse en dirección a los finales de carrera empezando por la articulación 1 y finalizando con la 3. Una vez se pulsa un final de carrera, la

articulación se separa un poco para dejar de pulsarlo. El robot en su posición inicial queda como se muestra en la siguiente imagen:

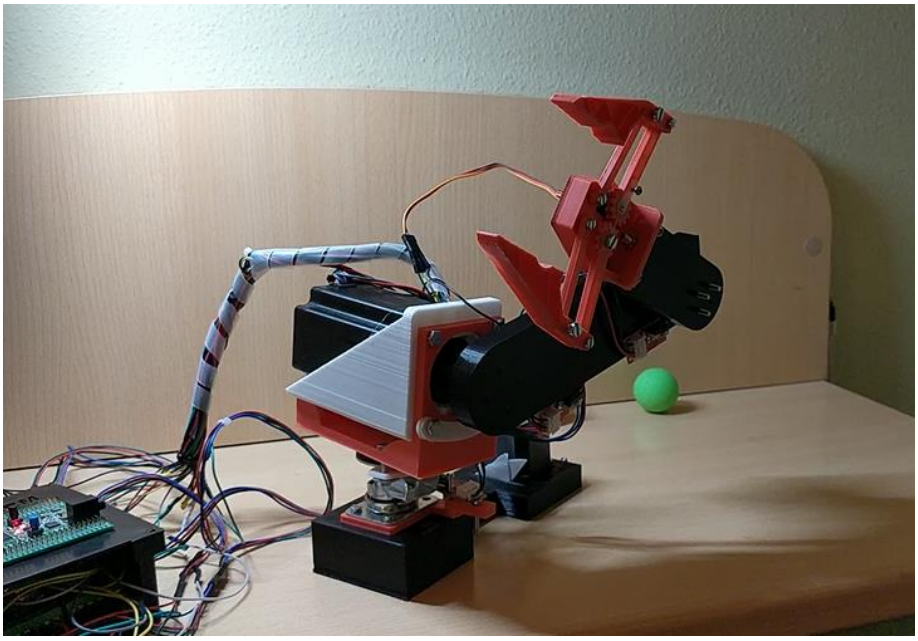


Ilustración 65 - Robot en posición inicial

En la aplicación del móvil, se puede observar cómo los ángulos del modo de cinemática directa han cambiado al valor que tienen las articulaciones en posición inicial como se muestra en la siguiente imagen:

Ilustración 66 - Feedback puesta a cero del robot

En la aplicación, se puede comprobar también que los mensajes recibidos son correctos, como el que se puede observar en la imagen anterior en la que se muestra “B 2 puesta a 0”, lo que quiere decir que la articulación 2 ha sido puesta a la posición inicial.

A continuación, se introducen los siguientes ángulos: q2 = 60 y q3 = 90. De esta forma el robot se mueve a la posición con el ángulo que se ha determinado como se muestra en la siguiente imagen:

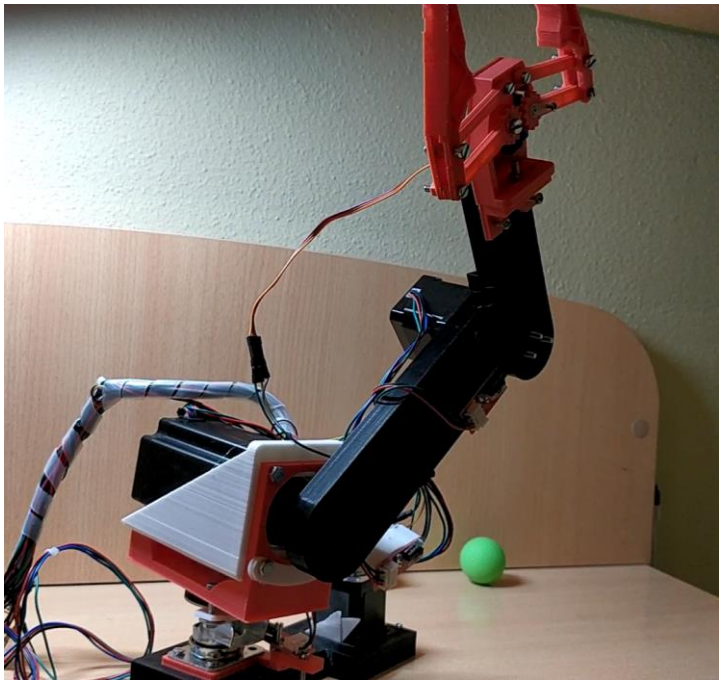


Ilustración 67 - Posición robot q1=-85, q2=60 y q3=90

Se comprueba visualmente que los ángulos son correctos, ya que la articulación 2 tiene un ángulo de 60º con respecto al suelo y la articulación 3 tiene un ángulo de 90º con respecto a la articulación 2.

Los siguientes ángulos que se introducen en la aplicación son, como se muestra en la siguiente imagen: q1=-84, q2=90 y q3=270.

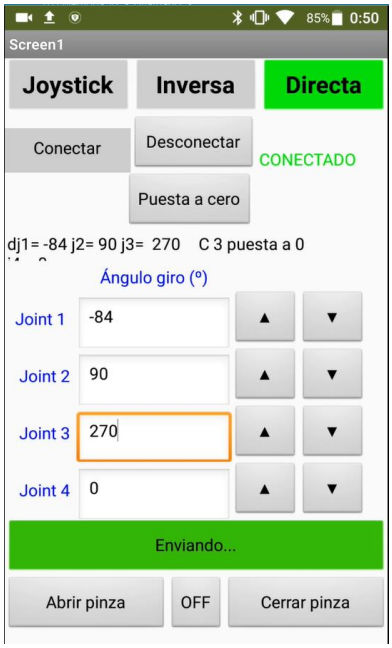


Ilustración 68 – Ángulo enviado q1=-84, q2=90 y q3=270

Por consiguiente, el robot se mueve a la posición deseada como se muestra en la siguiente imagen:

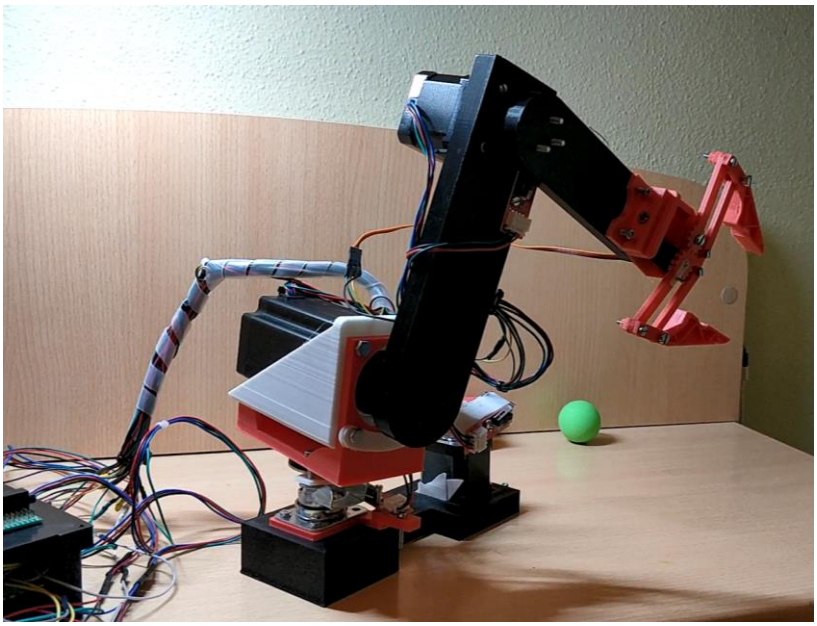


Ilustración 69 - Posición robot q1=-84, q2=90 y q3=270

Para acabar, se pulsa el botón de puesta a cero de la aplicación y se comprueba que el robot vuelve a su posición inicial como se indica en la siguiente imagen:

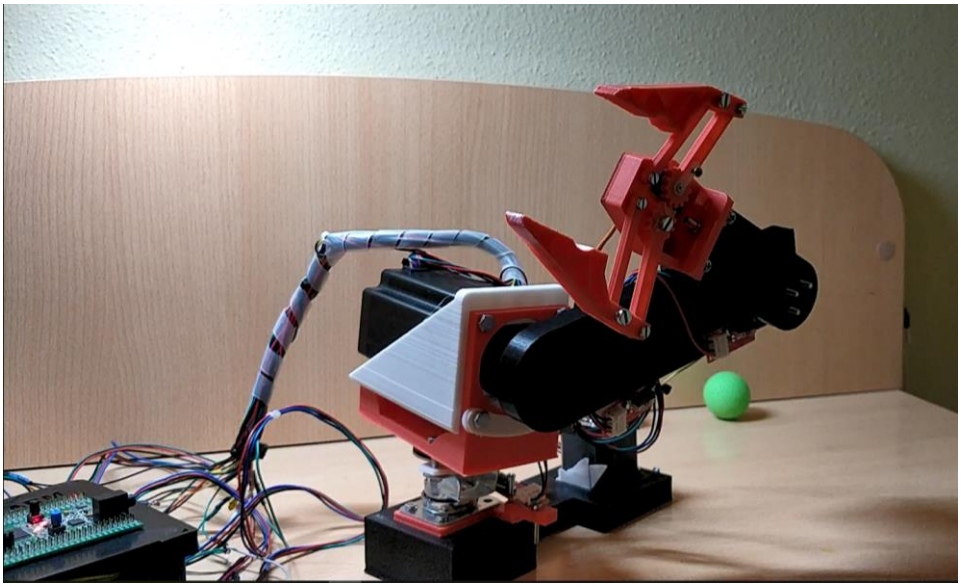


Ilustración 70 - Posición inicial por "Puesta a cero"

4.7.2. Cinemática inversa

Para probar esta funcionalidad, se introducirán dos coordenadas distintas y se comprobará si la posición de la pinza al final del movimiento es la correcta. Los valores que se introducen en la aplicación corresponden a medidas en milímetros desde el eje de giro del robot. Desde la posición inicial, se entra en la aplicación, en la pantalla de cinemática inversa y se introducen los valores que se muestran en la siguiente imagen:

Ilustración 71 - Coordenadas enviadas por cinemática inversa 1

Una vez se envían los valores desde la aplicación, se pulsa el botón de PA0 (botón azul del microcontrolador) y el robot comienza a moverse empezando por la primera articulación. El resultado final es el que se muestra en la siguiente imagen:

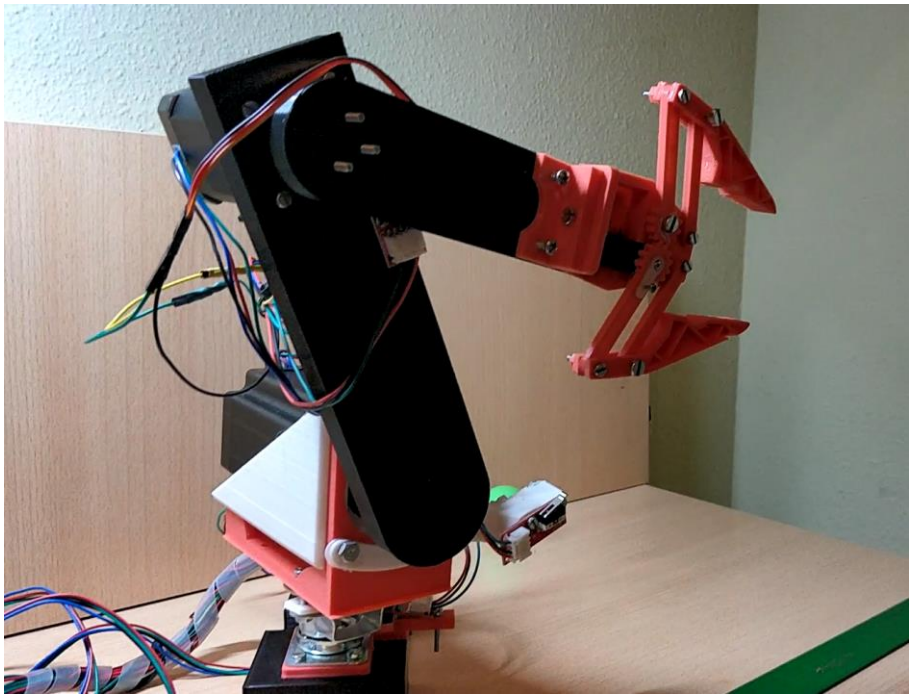


Ilustración 72 - Posición cinemática inversa 1

Si se imprimen por el debug del microcontrolador los ángulos para estas coordenadas serían los siguientes:

Como se puede observar en la imagen, los ángulos son los correctos, además de que se pudo comprobar que las medidas desde el eje de la base hasta la punta de la herramienta son correctas.

A continuación, el robot se vuelve a enviar a la posición inicial y se prueban otras coordenadas, las cuales son las que se muestran en la siguiente imagen:

Ilustración 73 - Coordenadas enviadas por cinemática inversa 2

Al pulsar nuevamente el botón PA0, el robot comienza a moverse quedando en la posición que muestra la siguiente imagen:

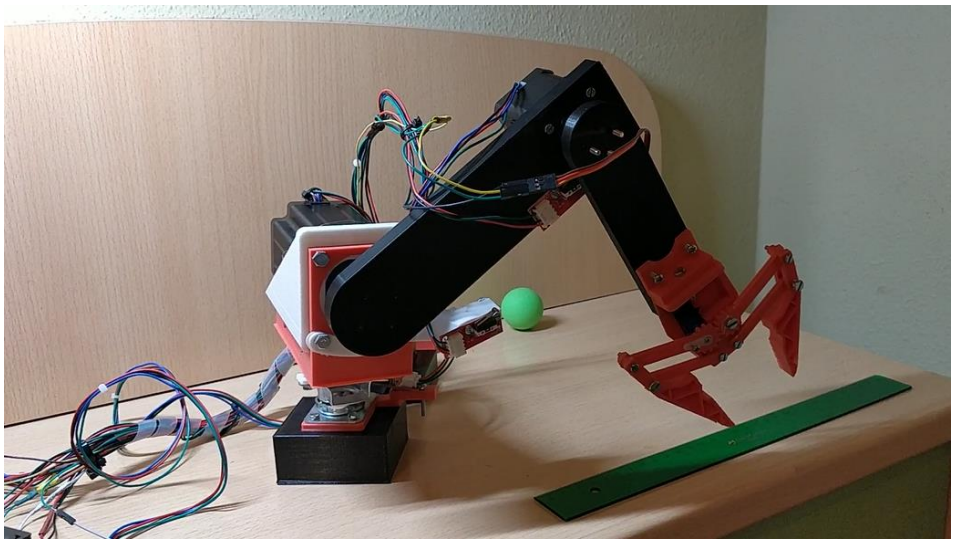


Ilustración 74 - Posición cinemática inversa 2

Como se observa en la siguiente imagen, el robot baja la altura de la herramienta con respecto a la anterior configuración y estira más el brazo porque la coordenada X ha aumenta su valor hasta 250.

A partir de aquí, se han probado otras configuraciones diferentes de coordenadas y ha resultado que en algunas de ellas el robot no consigue posicionarse correctamente por dos motivos principales:

- Al tratarse de un robot de 3 ejes, el rango de movimiento y de coordenadas que puede alcanzar es bastante reducido.
- Existen ciertas coordenadas que no son calculadas correctamente por las fórmulas y resultan en una posición errónea.

4.7.3. Pinza servo motor

El funcionamiento de la pinza es correcto aunque no perfecto, puesto que no era un objetivo del proyecto en principio, sino que ha sido un añadido extra.

En las siguientes imágenes se puede observar la pinza abierta y la pinza cerrada por la acción del servo. También se ha conseguido que pueda agarrar pequeños objetos, sin embargo es una de las líneas de mejora que se exponen el apartado 5.1 Conclusión.

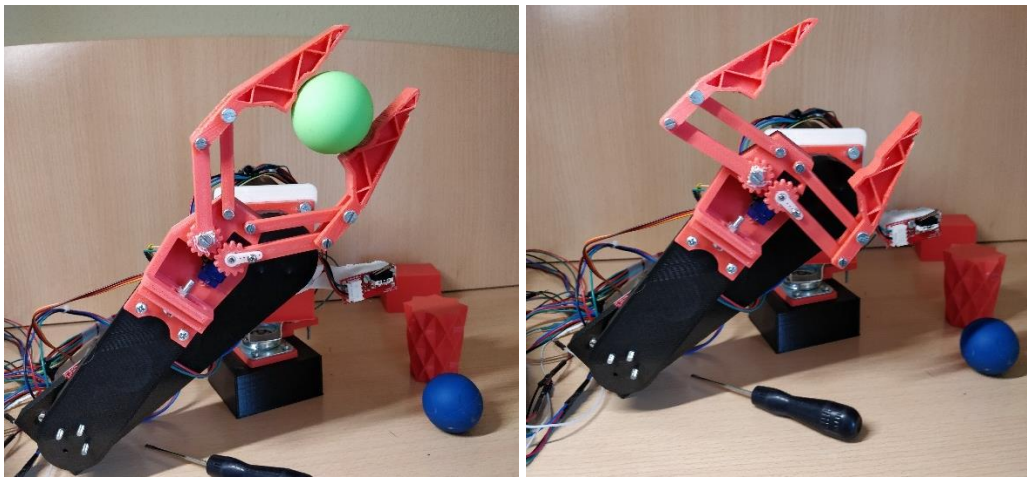


Ilustración 75 - Pinza servo abierta y cerrada

5. Conclusiones

5.1. Conclusión

En general, se puede afirmar que se ha conseguido un prototipo funcional cumpliendo con los objetivos propuestos al comienzo del proyecto. Los componentes seleccionados han sido los correctos y han sido dimensionados de forma efectiva, propiciando un buen funcionamiento y suficiente fuerza para que el robot pueda moverse libremente.

En cuanto a la elección de los motores paso a paso, se observa durante el desarrollo del proyecto que es un acierto debido a la facilidad del control y la gran precisión que se puede llegar a obtener.

Uno de los temores principales al trabajar con estos motores es la posible pérdida de pasos al no disponer de un sensor de posición y no saber realmente en qué posición se encuentra. Sin embargo, durante las pruebas de funcionamiento, se ha podido comprobar que, al realizar un dimensionamiento correcto estas pérdidas de pasos no llegan a producirse, aunque durante tiempos prolongados de funcionamiento podría darse el caso. Por eso los finales de carrera son esenciales cuando se trata de este tipo de motores.

En lo que respecta a la impresión 3D, se confirma que es la mejor herramienta para el desarrollo de prototipos de estas dimensiones. Esto es debido a la rapidez de implementar y probar diseños al tener una impresora 3D. Las piezas han cumplido su cometido con creces y no ha habido problemas de rupturas o baja calidad de impresión. Por el contrario, algunas piezas han tenido que ser reimpresas varias veces debido a fallos en el diseño inicial hasta que se ha conseguido una sinergia de todos los componentes.

Por otra parte, la comunicación bluetooth ha supuesto la necesidad de aprendizaje autónomo sobre este tema. Al final se ha conseguido crear un sistema de comunicación rápido y efectivo mediante el uso de caracteres especiales para el inicio de los mensajes. Uno de los aspectos importantes ha sido el desarrollo simultáneo de la programación de la aplicación y del microcontrolador para que puedan comunicar sin problema.

Como se ha comentado en anteriores apartados, uno de los objetivos para el proyecto era el control del brazo mediante un joystick y utilizando los cálculos de la jacobiana. Pese a estar implementado en la aplicación del móvil, no se ha podido desarrollar en el microcontrolador por falta de tiempo. Sin embargo, se ha conseguido desarrollar un control de cinemática directa y otro de cinemática inversa, como se puede observar en Pruebas de funcionamiento.

En cuanto a estas pruebas, se ha comprobado que las tareas que se habían programado funcionan correctamente. La cinemática directa funciona perfectamente al tratarse de una implementación sencilla, sin embargo, en la cinemática inversa no se conseguido un posicionamiento exacto para algunas de las coordenadas que se le enviaban por lo que se añade como futura mejora del prototipo.

En conclusión, se ha desarrollado un prototipo funcional con el cual se ha conseguido comunicar con una aplicación mediante bluetooth y realizar un control preciso del ángulo de

giro de las articulaciones. A partir de aquí y con más tiempo, se podrían añadir funcionalidades como el *pick and place* para que el robot pueda tener aplicaciones prácticas en la vida cotidiana. Una de estas aplicaciones podría ser sustituir la pinza por una herramienta de dibujo y crear, mediante una serie de coordenadas como se hace con las impresoras 3D, un dibujo.

5.2. Posibles mejoras

Por la complejidad del proyecto, se establecieron unos objetivos asequibles y la mayoría pudieron cumplirse. Es por eso que el prototipo presenta varias futuras líneas de mejora que se exponen a continuación:

- **Joystick**: la principal mejora es el desarrollo del control por jacobiana, ya que la comunicación está desarrollada y solo faltarían los cálculos de matrices y la implementación en el microcontrolador.
- **Cinemática inversa**: esta funcionalidad está desarrollada en el proyecto, pero debe mejorarse para conseguir un rango más completo de coordenadas y aumentar la precisión.
- **Control de velocidad**: junto con el desarrollo de la jacobiana se puede realizar un control de velocidad, ya que se ha desarrollado la función para el control de los motores en función de la velocidad, pero actualmente funcionan a velocidad constante.
- **Número de articulaciones**: una vez se tiene el diseño y el control de 3 articulaciones, resulta relativamente sencillo añadir grados de libertad al diseño. De esta forma, el robot tendrá más libertad de movimiento y podrá orientar la herramienta hacia la posición adecuada. La única complejidad sería la de recalcular la cinemática inversa, aunque solo habría que realizar los cálculos una vez.
- **Movimiento simultáneo de articulaciones:** la mayoría de los robots articulados son capaces de mover todas las articulaciones de forma simultánea para realizar movimientos más rápidos y coordinados. Esta funcionalidad sería interesante poder implementarla mediante el uso de señales PWM para el control de los motores.
- **Cableado interno y mejor diseño de piezas**: los motores paso a paso necesitan 4 cables por motor, a parte de los pines que necesitan los drivers para ser controlados, por lo que al final un guiado de cableado por dentro de las piezas resulta lo más atractivo.
- **Mejora de la articulación 1**: esta articulación supuso un reto por la necesidad de un diseño mecánico robusto que permitiera un giro completo del robot. Se consiguió una solución práctica con la correa de transmisión y los ejes, sin embargo se necesita un mejor diseño para evitar deslizamientos de la correa y ganar robustez y precisión.

5.3. Relación del trabajo con los estudios cursados

La idea de este trabajo se ha basado en la realización de un prototipo de robot articulado, extraído de la asignatura de Sistemas Robotizados. Se han utilizado los conocimientos de esta asignatura para el diseño del robot y el desarrollo de la cinemática directa e inversa. Sin embargo, también se han utilizado conocimientos de otras disciplinas dentro del grado.

Ha sido necesario tener conocimientos de programación en lenguaje C, además de manejo con los microcontroladores y conocer cómo funciona internamente para la configuración de los pines. Estas habilidades con los microcontroladores han servido de base para ampliar conocimientos en cuanto al desarrollo de la comunicación bluetooth.

Por otro lado, se han utilizado los conocimientos de electrónica para poder entender datasheets y extraer la información necesaria. También se han utilizado conocimientos de electrónica de potencia para el suministro de alimentación y el desarrollo del regulador de tensión para el control de la pinza servocontrolada.

En cuanto a la parte de impresión 3D, no se aprende nada de esta materia durante la carrera, sino que ha sido la experiencia personal con este tipo de tecnología la que ha ayudado al desarrollo e impresión de las piezas del proyecto.

En general, los conocimientos adquiridos durante el grado ofrecen unas herramientas y una base muy útil a la hora de desarrollar proyectos como este. Sin embargo, con esta base de conocimiento no es suficiente para cualquier proyecto, se necesitan poseer nociones básicas de diversas disciplinas, además de que estas tecnologías avanzan rápidamente. Es por eso que hay un amplio grado de autoformación y hoy en día con internet hay mucha información de la que se pueden extraer amplios conocimientos sobre infinidad de materias.

6. Bibliografía

[1] J. D. d. Usera, «HardZone,» 2 Junio 2018. [En línea]. Available: https://hardzone.es/2018/06/02/como-reducir-nanometros-procesador/. [Último acceso: Mayo 2019].

[2] S. Vinssa, «Vinssa Industrial Solutions,» 17 November 2018. [En línea]. Available: https://blog.vinssa.com/robots-industriales-historia-clasificacion-y-funcionalidad. [Último acceso: Mayo 2019].

[3] G. Iborra, «Novedades Automatización,» 20 DIciembre 2017. [En línea]. Available: https://novedadesautomatizacion.net/fanuc-m-2000ia/. [Último acceso: Mayo 2019].

[4] Á. Perles, ARM Cortex-M práctico. 1 - Introducción a los microcontroladores STM32 de St, Valencia: UPV, 2018.

[5] STMicroelectronics, «STM32F3xxx and STM32F4xxx Cortex-M4 programming manual,» 2012.

[6] J. V. C. Hernández, «Temporización mediante el temporizador del sistema SysTick en microcontroladores ARM Cortex-M,» UPV, Valencia.

[7] Anatoliy, «Solderer TV,» 1 Marzo 2013. [En línea]. Available: http://solderer.tv/communication-between-the-stm32-and-android-via-bluetooth/. [Último acceso: Mayo 2019].

[8] MrElberni, «Microcontroladores,» [En línea]. Available: http://microcontroladores-mrelberni.com/usart-pic-comunicacion-serial/.

[9] STMicroelectronics, «STM32F04x Datasheet,» 2012.

[10] T. Majerle, «STM32F4 Discovery,» [En línea]. Available: https://stm32f4-discovery.net/. [Último acceso: 2019].

DOCUMENTO 2:

PRESUPUESTO

DISEÑO, IMPLEMENTACIÓN Y PROGRAMACIÓN DE UN SISTEMA ROBOTIZADO PARA TAREAS DE TELEOPERACIÓN

Presentado por:

GUILLERMO MÁRQUEZ RUIZ

Dirigido por:

LUIS IGNACIO GRACIA CALANDÍN

Valencia, 11 de junio de 2019

7. Coste económico del proyecto

7.1. Coste material

En este apartado se pasa a enumerar todos los componentes que han sido usados en este proyecto, además de calcular el presupuesto del material.

Material	Cantidad	Precio	Total
Fuente de alimentación: *350W 36V 9.7A 115/230V Switching Power Supply*	1	35 €	35 €
Motor paso a paso Nema 24	1	65 €	65 €
Motor paso a paso Nema 17	2	12 €	24 €
Rueda giratoria	1	3 €	3 €
Correa 6 mm GT2	1	2 €	2 €
Driver DRV8825	3	4 €	12 €
Placas PCB	3	1.5 €	4.5 €
LM317T	1	2 €	2 €
Componentes electrónicos: resistencias, condensadores, plástico termorrectráctil, conectores…	1	10 €	10 €
Cableado	1	15 €	15 €
Microcontrolador STM32F407 Discovery	1	18 €	18 €
Pulsadores finales de carrera	3	3 €	9 €
Módulo bluetooth HC-05	1	8.50 €	8.5 €
Servomotor Towerpro Sg90	1	2 €	2 €
Tornillería	1	5 €	5 €
			215 €

Tabla 13 - Coste material

7.2. Coste piezas impresión 3D

A continuación, se enumera el listado de piezas que se han impreso y el precio del material usado. En la memoria se pueden encontrar más detalles sobre estas piezas como el tiempo de impresión y la cantidad de material usado.

Pieza	Cantidad	Coste	Total
Base soporte	1	2.09 €	2.09 €
Soporte final de carrera articulación 1	1	0.18 €	0.18 €
Engranaje 64 dientes *Thingiverse*	1	0.16 €	0.16 €
Enganche engranaje	1	0.08 €	0.08 €
Base para articulación 2	1	1.37 €	1.37 €
Soporte final de carrera articulación 2	1	0.16 €	0.16 €
Soporte motor Nema 24	1	1.08 €	1.08 €
Brazo articulación 2	1	1.24 €	1.24 €
Brazo articulación 3	1	1.05 €	1.05 €
Enganche herramienta	1	0.09 €	0.09 €
Herramienta pinza servo	1	0.78 €	0.78 €
Encapsulado para los circuitos	1	2.38 €	2.38 €
			10.66 €

Tabla 14 - Coste piezas impresas en 3D

Como se puede observar en la Tabla 14, el coste de todas las piezas impresas en 3D es muy bajo con respecto a las posibilidades que ofrece. Además por este bajo coste se ha conseguido el objetivo de que el coste de materiales sea menor de 300 €.

7.3. Coste recursos humanos

En cuanto al coste de recursos humanos, se toma como referencia lo que cobra un Ingeniero Junior en el mercado (20 €/h) y se hace una estimación de las horas empleadas en la realización de este trabajo.

Número de horas	Precio por hora	Total
400 horas	20 €	8000 €

Tabla 15 - Coste recursos humanos

DOCUMENTO 3:

PLANOS

DISEÑO, IMPLEMENTACIÓN Y PROGRAMACIÓN DE UN SISTEMA ROBOTIZADO PARA TAREAS DE TELEOPERACIÓN

Presentado por:

GUILLERMO MÁRQUEZ RUIZ

Dirigido por:

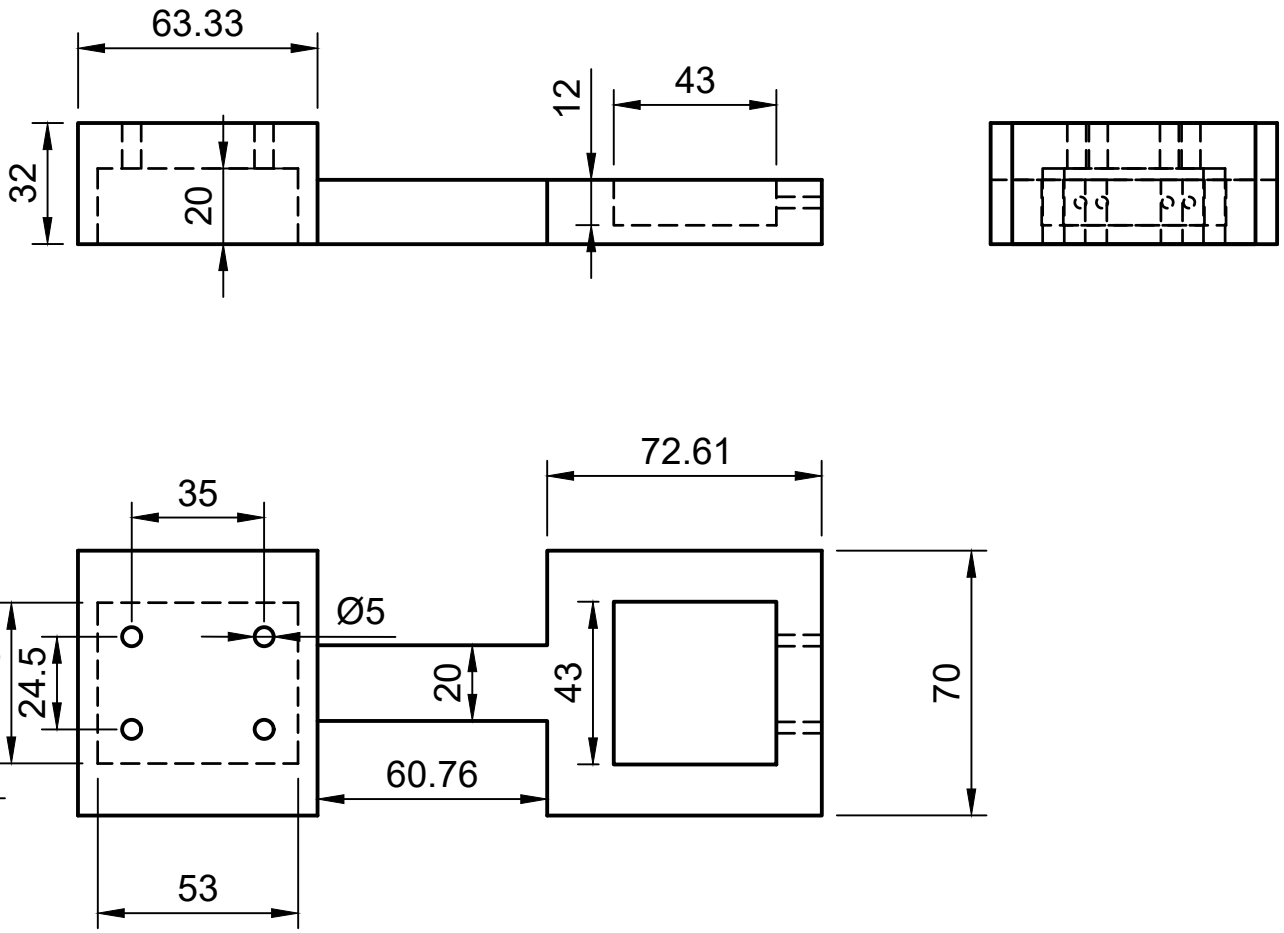
LUIS IGNACIO GRACIA CALANDÍN

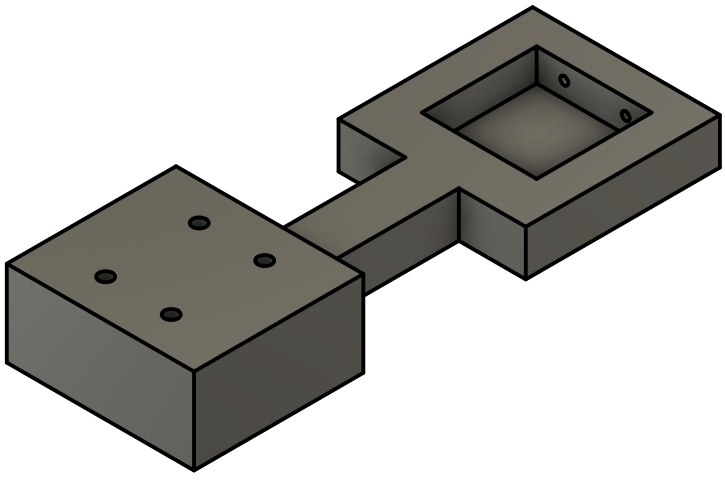
Valencia, 11 de junio de 2019

8. Planos de piezas 3D

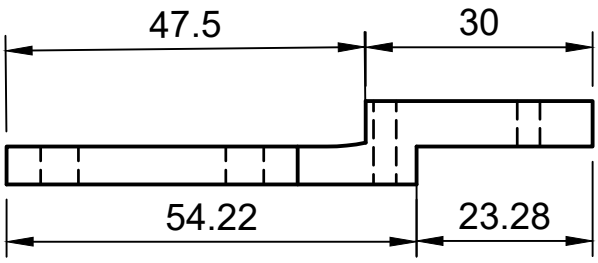
Los planos han sido realizados de las piezas que se han diseñado con el programa Fusion 360 e impreso en 3D, las cuales son:

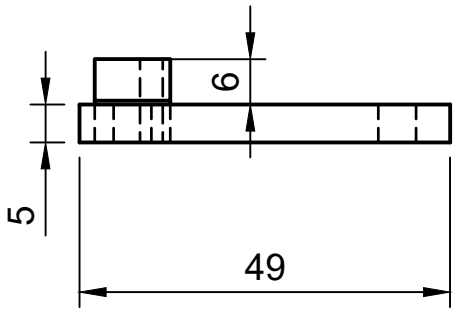
- Base soporte articulación 1
- Soporte final de carrera articulación 1
- Enganche engranaje
- Base para articulación 2
- Soporte final de carrera articulación 2
- Soporte motor Nema 24
- Brazo articulación 2
- Brazo articulación 3
- Enganche herramienta
- Encapsulado circuitos

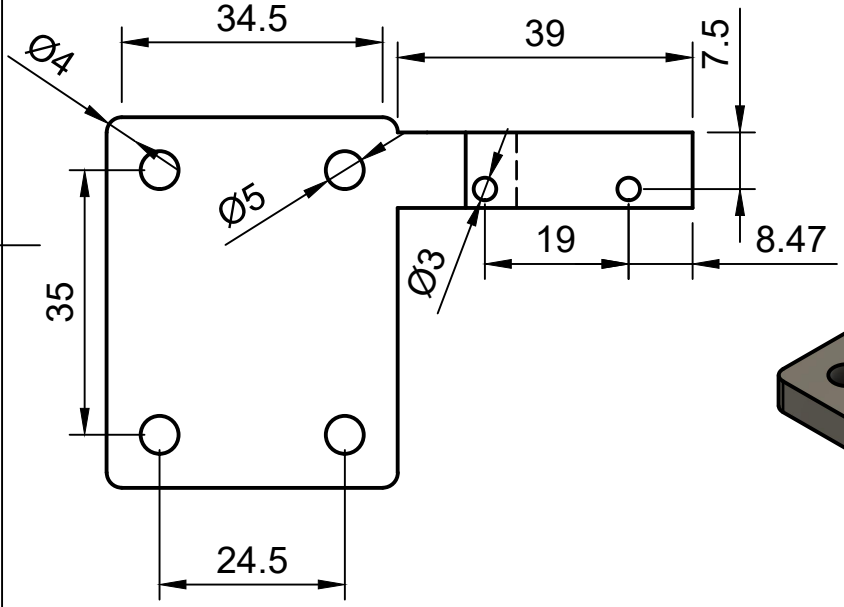


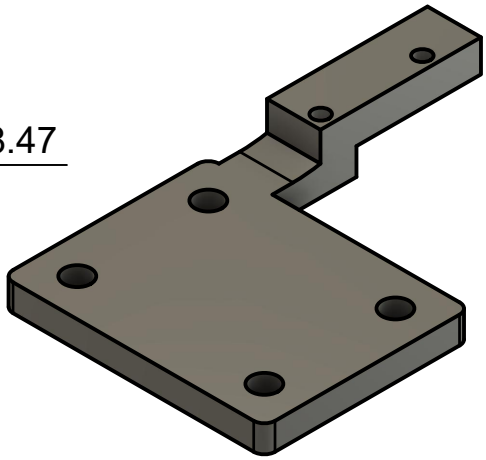


Scale	Technical reference	Created by	Approved by
1:2		Guillermo Márquez 11/06/2019	
		Document type	Document status
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Title Base soporte	DWG No. D01
			Rev. 2 / Date of issue / Sheet 116

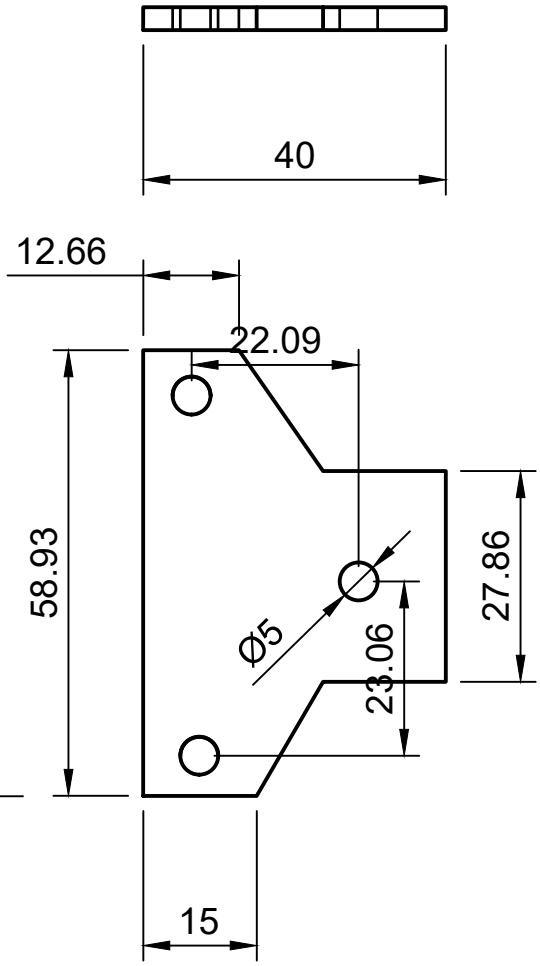


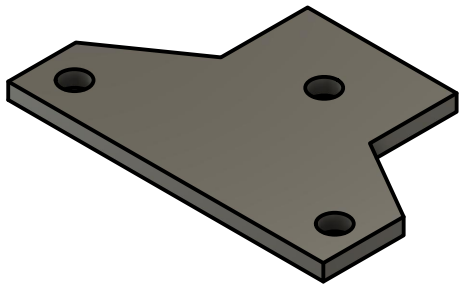






Scale	Technical reference	Created by	Approved by
1:1		Guillermo Márquez 11/06/2019	
		Document type	Document status
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Title Soporte final de carrera 1	DWG No. D09
			Rev. 5 / Date of issue / Sheet 117





Scale	Technical reference	Created by	Approved by
1:1		Guillermo Márquez 11/06/2019	

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Document type	Document status

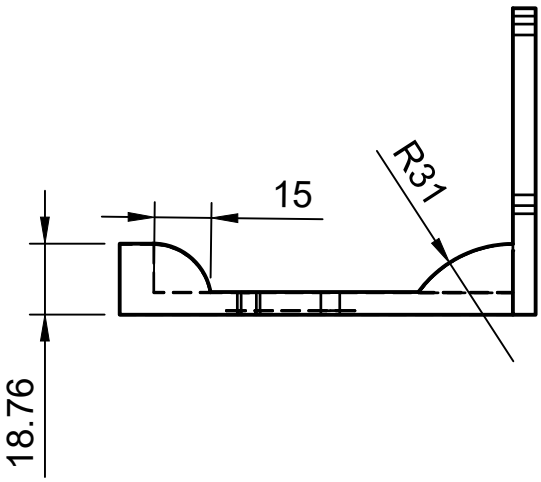
Title

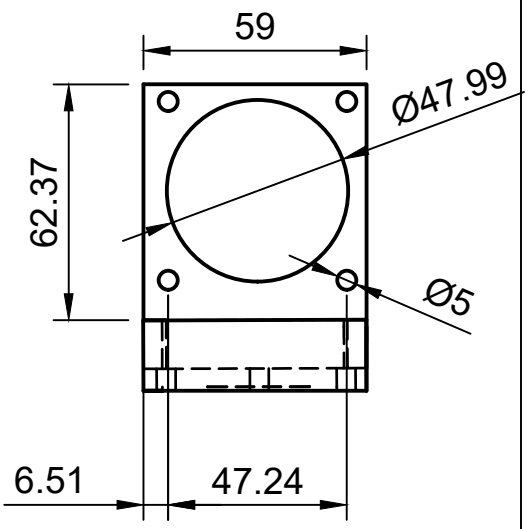
Enganche engranaje

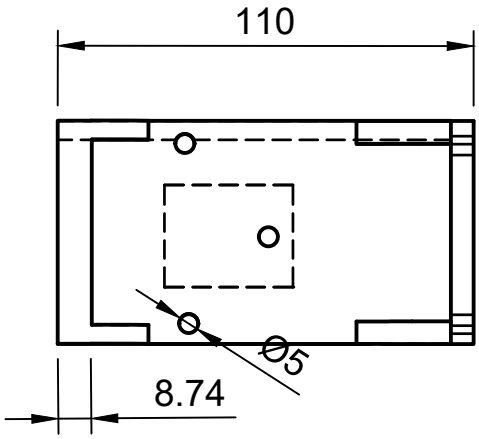
DWG No.

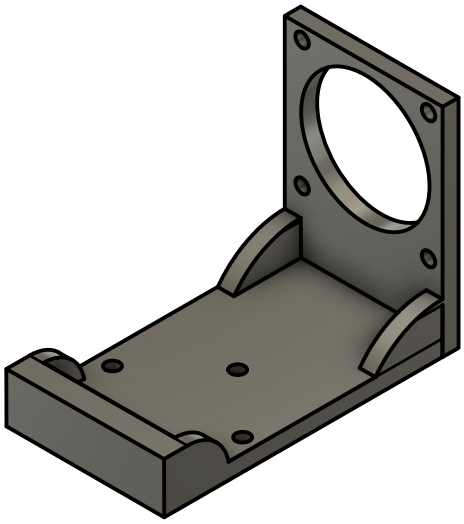
D04

Rev.	Date of issue	Sheet
8		118









Scale	Technical reference	Created by	Approved by
1:2		Guillermo Márquez 11/06/2019	

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Document type	Document status

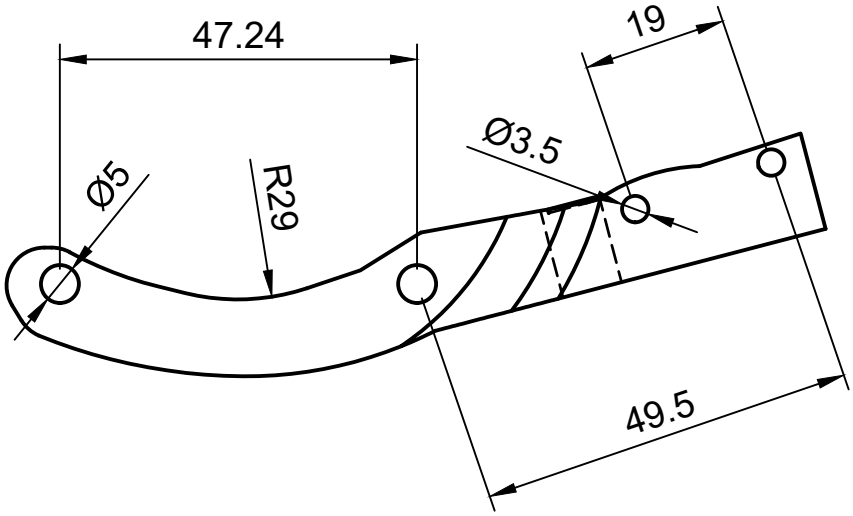
Title

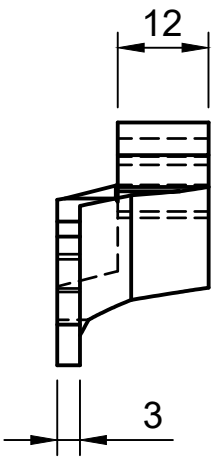
Base articulación 2

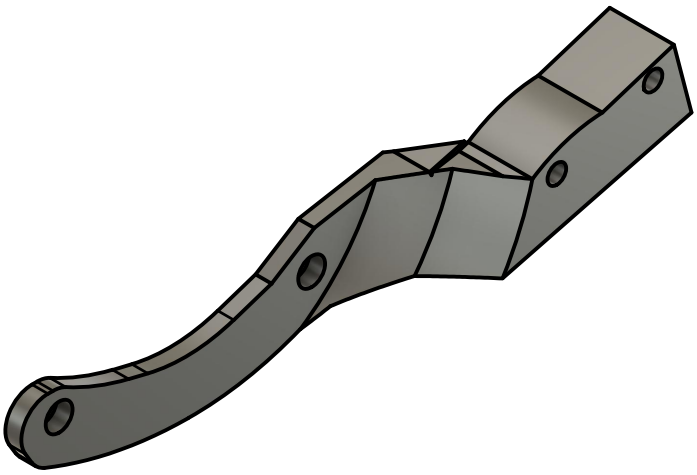
DWG No.

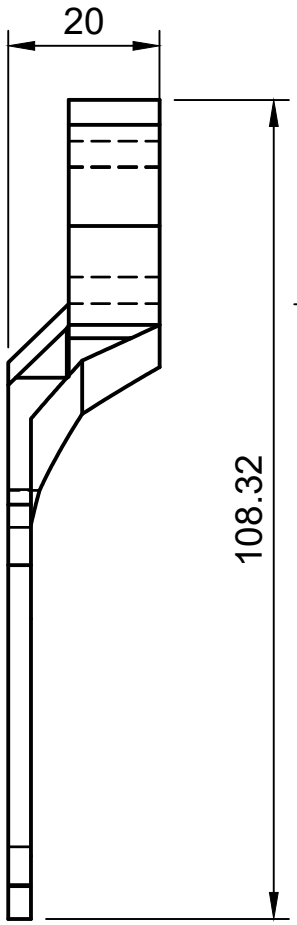
D02

Rev.	Date of issue	Sheet
8		119

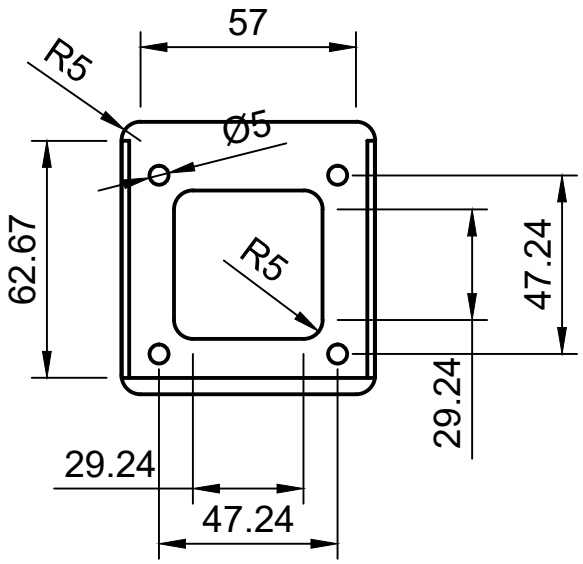


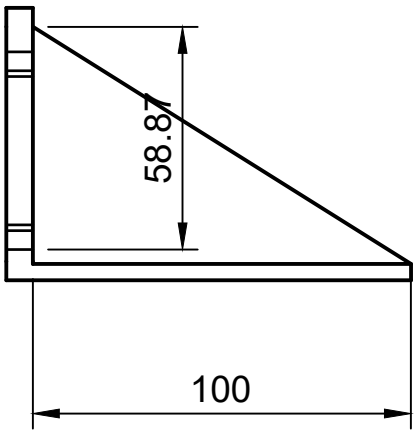


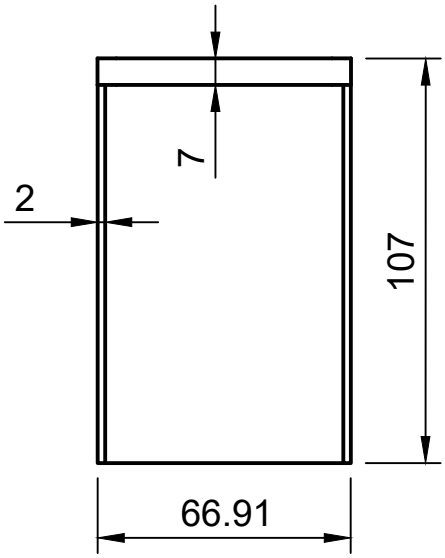


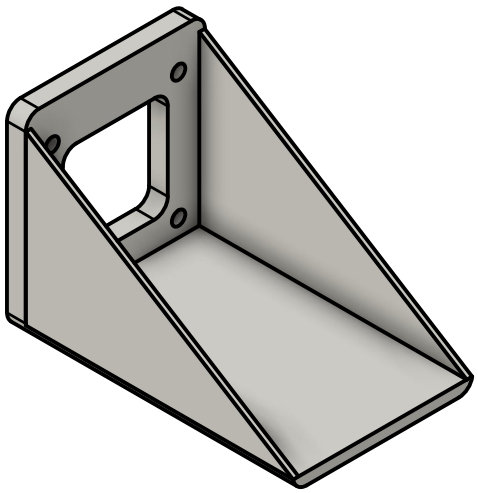


Scale	Technical reference	Created by	Approved by
1:1		Guillermo Márquez 11/06/2019	
		Document type	Document status
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Title Soporte final carrera 2	DWG No. D03
			Rev. 19 / Date of issue / Sheet 120

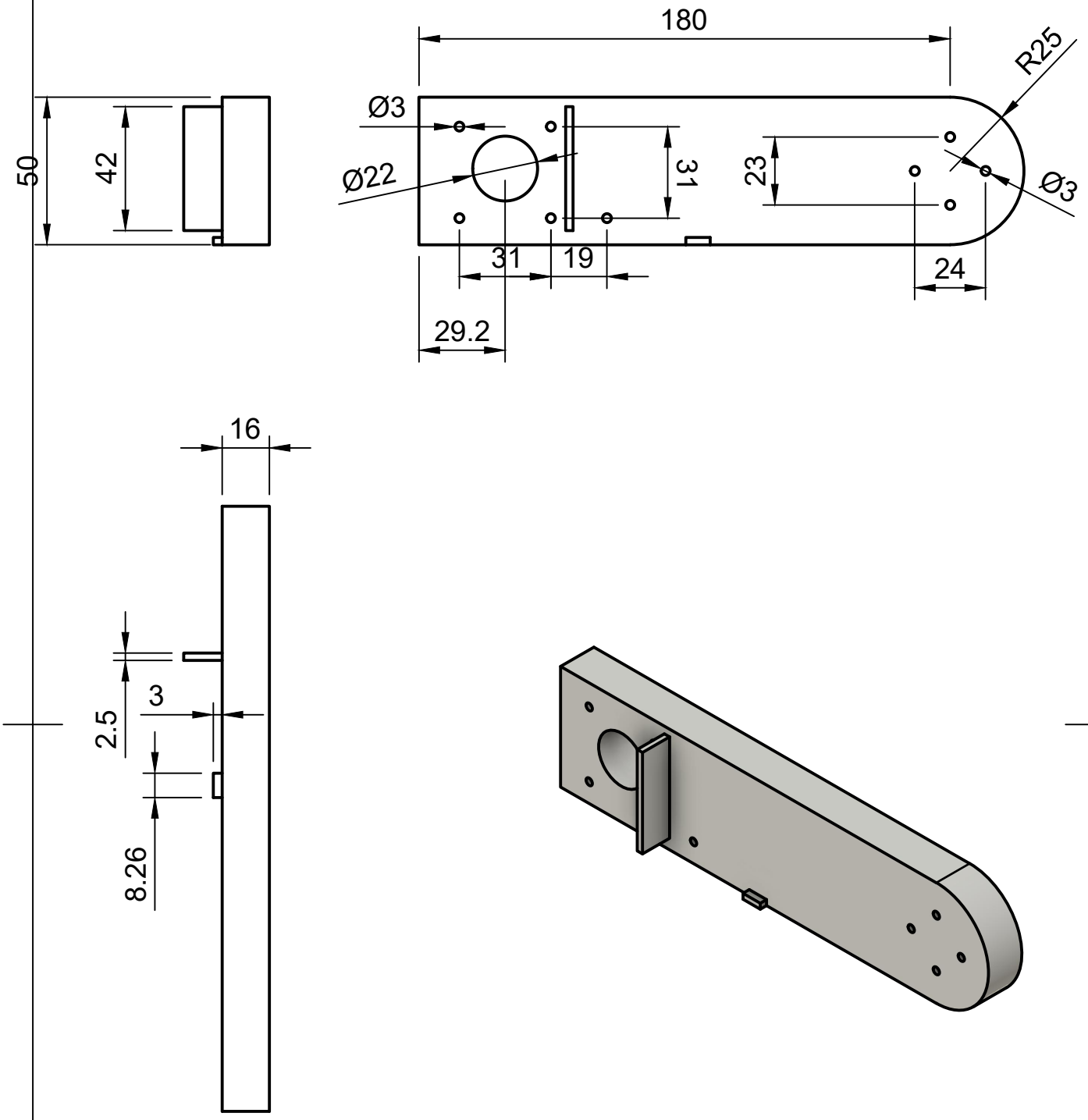




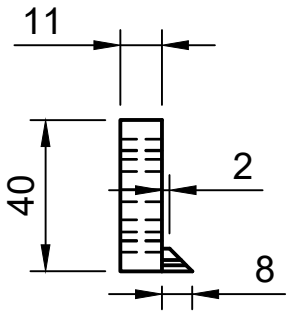


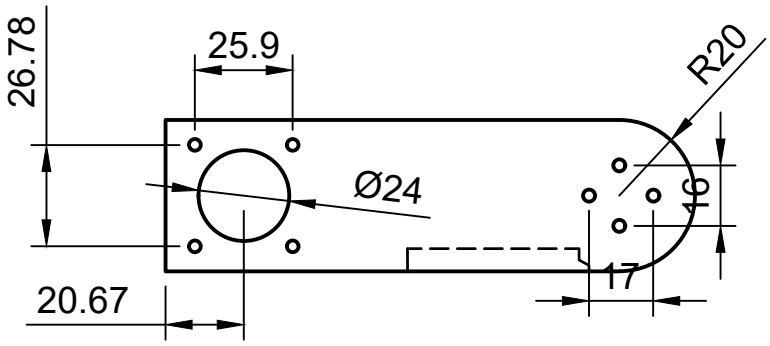


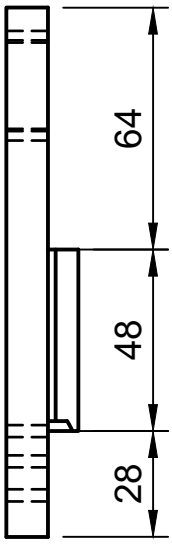
Scale	Technical reference	Created by	Approved by
1:2		Guillermo Márquez 11/06/2019	
		Document type	Document status
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Title Soporte Nema 24	DWG No. D05
			Rev. 2 / Date of issue / Sheet 121

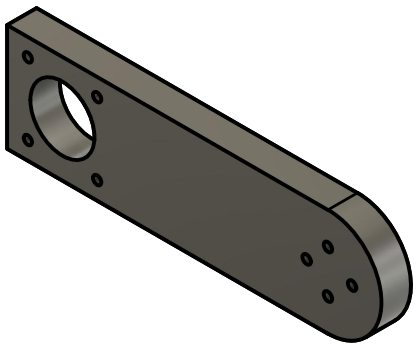


Scale 1:2	Technical reference	Created by Guillermo Márquez 11/06/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title Brazo articulación 2	DWG No. D06
		Rev. 5	Date of issue
			Sheet 122









Scale	Technical reference	Created by	Approved by
1:2		Guillermo Márquez 11/06/2019	

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Document type	Document status

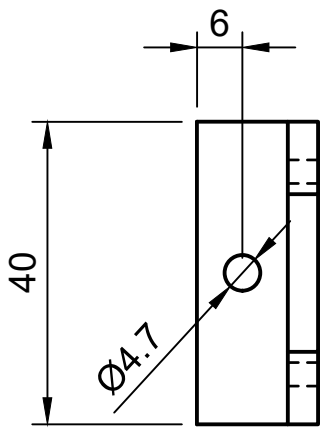
Title

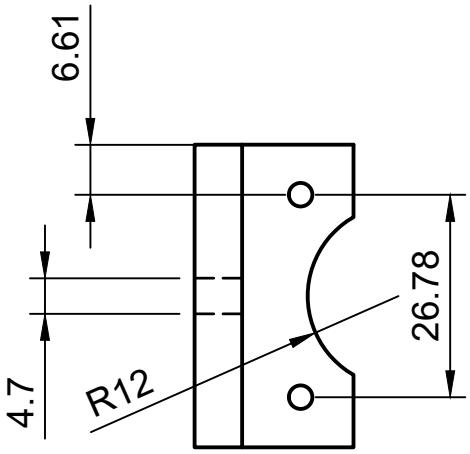
Brazo articulación 3

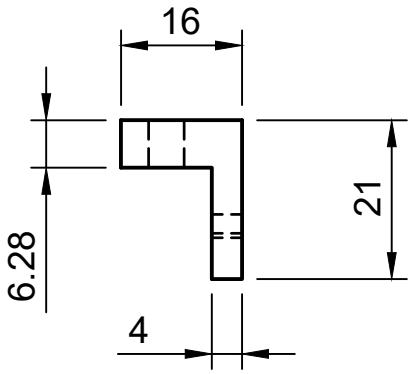
DWG No.

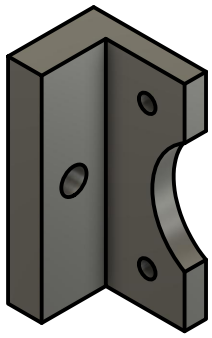
D07

Rev.	Date of issue	Sheet
10		123

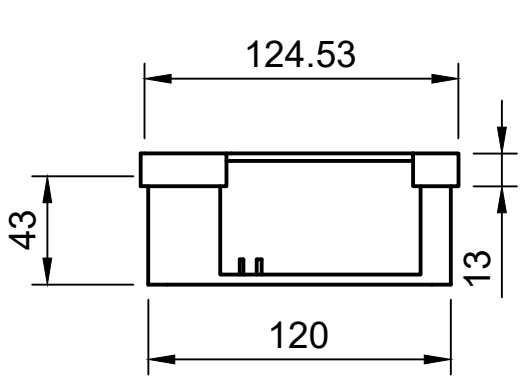


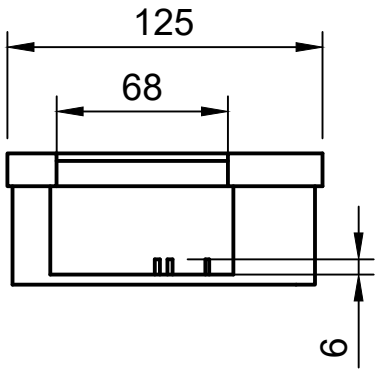


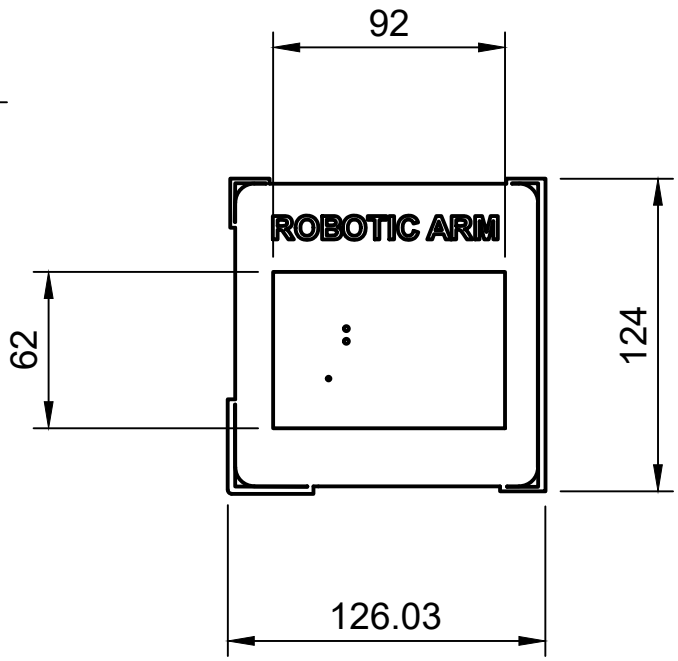


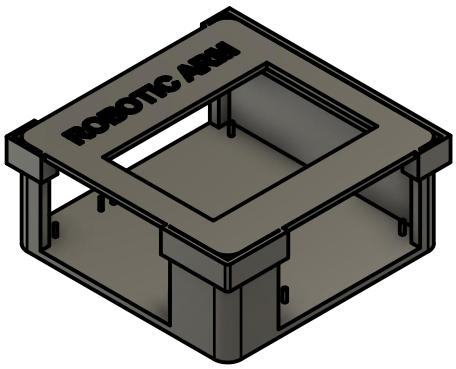


Scale 1:1	Technical reference	Created by Guillermo Márquez 11/06/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title Enganche pinza	DWG No. D08
			Rev. 4 / Date of issue / Sheet 124









Scale	Technical reference	Created by	Approved by
1:3		Guillermo Márquez 11/06/2019	

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Document type	Document status

Title

Encapsulado circuitos

DWG No.

D10

Rev.	Date of issue	Sheet
3		125

ANEXOS

Anexo I: Código microcontrolador

MAIN.C

```
/**
   @file main.c

   @brief Plantilla Keil 5 para la placa STM32F$Discovery
   Más detalles en el archivo leeme.txt

   @author Equipo ARM Power http://armpower.blogs.upv.es/
   @date 2017/03/06
*/

#include <stdio.h>
#include <stm32f4xx.h>
#include <stm32f4xx_it.h>
#include <stm32f4xx_usart.h>
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "analog_init.h"
#include "stm32f4xx_adc.h"
#include "tm_stm32f4_usart.h"
#include "delay.h"
#include <math.h>
#include "pwm_init.h"
#include "string.h"
#include "matrices.h"



#define baudrate 9600
#define STEP_ANGLE1 0.1125
#define STEP_ANGLE2 0.1125
#define STEP_ANGLE3 0.1125
#define Pi 3.14159


int32_t ejex=0;
int32_t ejey=0;
uint8_t vel_stepper1=50, vel_stepper2=30, vel_stepper3=30;


void pin_inicializar(void){
  GPIO_InitTypeDef GPIO_Motor;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

    GPIO_Motor.GPIO_Pin = GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15|GPIO_Pin_5|GPIO_Pin_1|GPIO_Pin_3|GPIO_Pin_4
|GPIO_Pin_2|GPIO_Pin_7|GPIO_Pin_6;
    GPIO_Motor.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_Motor.GPIO_OType = GPIO_OType_PP;
    GPIO_Motor.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Motor.GPIO_PuPd = GPIO_PuPd_NOPULL;

    GPIO_Init(GPIOD, &GPIO_Motor);

    GPIO_InitTypeDef GPIO_Leds2;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    GPIO_Leds2.GPIO_Pin = GPIO_Pin_11|GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14;
    GPIO_Leds2.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_Leds2.GPIO_OType = GPIO_OType_PP;
    GPIO_Leds2.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Leds2.GPIO_PuPd = GPIO_PuPd_NOPULL;

    GPIO_Init(GPIOB, &GPIO_Leds2);
```

```
64.
65.     GPIO_InitTypeDef GPIO_IN;
66.
67.     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
68.
69.     GPIO_IN.GPIO_Pin = GPIO_Pin_0;
70.     GPIO_IN.GPIO_Mode = GPIO_Mode_IN;
71.     GPIO_IN.GPIO_Speed = GPIO_Speed_100MHz;
72.
73.     GPIO_Init(GPIOA, &GPIO_IN);
74.
75.     //Fin de carrera 1 y 2
76.     GPIO_InitTypeDef GPIO_FC;
77.
78.     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
79.
80.     GPIO_FC.GPIO_Pin = GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5;
81.     GPIO_FC.GPIO_Mode = GPIO_Mode_IN;
82.     GPIO_FC.GPIO_PuPd = GPIO_PuPd_NOPULL;
83.
84.     GPIO_Init(GPIOC, &GPIO_FC);
85.
86.     //PRueba PWM
87.         GPIO_InitTypeDef GPIO_PWM;
88.
89.     /* Clock for GPIOA */
90.     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
91.
92.     /* Alternating functions for pins */
93.     GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_TIM4);
94.
95.     /* Set pins */
96.     GPIO_PWM.GPIO_Pin = GPIO_Pin_12;
97.     GPIO_PWM.GPIO_OType = GPIO_OType_PP;
98.     GPIO_PWM.GPIO_PuPd = GPIO_PuPd_NOPULL;
99.     GPIO_PWM.GPIO_Mode = GPIO_Mode_AF;
100.    GPIO_PWM.GPIO_Speed = GPIO_Speed_100MHz;
101.    GPIO_Init(GPIOD, &GPIO_PWM);
102.}
103.
104.void interrupt_config(void){
105.
106.        //Interrupt motor 1
107.    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
108.    //Conect pin PD4 to the interrupt EXTI Line 3
109.    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOC, EXTI_PinSource3);
110.
111.    EXTI_InitTypeDef EXTI_Struct1;
112.
113.    EXTI_Struct1.EXTI_Line = EXTI_Line3;
114.    EXTI_Struct1.EXTI_Mode = EXTI_Mode_Interrupt;
115.    EXTI_Struct1.EXTI_Trigger = EXTI_Trigger_Falling;
116.    EXTI_Struct1.EXTI_LineCmd = ENABLE;
117.    EXTI_Init(&EXTI_Struct1);
118.
119.    NVIC_InitTypeDef NVIC_Struct1;
120.    NVIC_Struct1.NVIC_IRQChannel = EXTI3_IRQn;
121.    NVIC_Struct1.NVIC_IRQChannelPreemptionPriority = 0x00;
122.    NVIC_Struct1.NVIC_IRQChannelSubPriority = 0x00;
123.    NVIC_Init(&NVIC_Struct1);
124.
125.    NVIC_EnableIRQ(EXTI3_IRQn);
126.
127.    //Interrupt motor 1
128.    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
129.    //Conect pin PD4 to the interrupt EXTI Line 4
130.    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOC, EXTI_PinSource4);
131.
132.    EXTI_InitTypeDef EXTI_Struct2;
133.
134.    EXTI_Struct2.EXTI_Line = EXTI_Line4;
135.    EXTI_Struct2.EXTI_Mode = EXTI_Mode_Interrupt;
```

```
    EXTI_Struct2.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_Struct2.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_Struct2);

    NVIC_InitTypeDef NVIC_Struct2;
    NVIC_Struct2.NVIC_IRQChannel = EXTI4_IRQn;
    NVIC_Struct2.NVIC_IRQChannelPreemptionPriority = 0x01;
    NVIC_Struct2.NVIC_IRQChannelSubPriority = 0x01;
    NVIC_Init(&NVIC_Struct2);

    NVIC_EnableIRQ(EXTI4_IRQn);

    //Interrupt motor 2
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
    //Conect pin PD5 to the interrupt EXTI Line 5
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOC, EXTI_PinSource5);

    EXTI_InitTypeDef EXTI_Struct3;

    EXTI_Struct3.EXTI_Line = EXTI_Line5;
    EXTI_Struct3.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_Struct3.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_Struct3.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_Struct3);

    NVIC_InitTypeDef NVIC_Struct3;
    NVIC_Struct3.NVIC_IRQChannel = EXTI9_5_IRQn;
    NVIC_Struct3.NVIC_IRQChannelPreemptionPriority = 0x02;
    NVIC_Struct3.NVIC_IRQChannelSubPriority = 0x02;
    NVIC_Init(&NVIC_Struct3);

    NVIC_EnableIRQ(EXTI9_5_IRQn);

}



int16_t angle_directa_j1=0;
int angle_bl=0, angle_ant_J1=0, angle_ant_J2=0, angle_ant_J3=0;
int return_angle(char text_to_convert[],int16_t X_val, int16_t Y_val, int16_t Z_val){
//esta funcion lee las dos coordenadas X e Y del texto recibido por bluetooth y devuelve el ángulo
    int num_arrayX=0, num_arrayY=0, num_arrayZ=0;
    double angle=0.0, X=0.0, Y=0.0, Z=0.0;
    uint8_t cont_i=0;
    char letras[2], letras2[2], letras3[2];
    sscanf(text_to_convert,"%s %d %s %d %s %d",letras,&num_arrayX, letras2, &num_arrayY,
  letras3, &num_arrayZ);
    printf("%s\n", text_to_convert);
    printf("%d %d %d\n", num_arrayX, num_arrayY, num_arrayZ);
    X=num_arrayX;
    X_val=num_arrayX;
    Y=num_arrayY;
    Y_val=num_arrayY;
    Z=num_arrayZ;
    Z_val=num_arrayZ;
    angle=atan2(Y,X);
    angle=angle*180/Pi;
    return angle;
}

int16_t *return_angle_directa(char text_to_convert[]){
//esta funcion lee los valores recibidos de la cinematica directa
    char j1[3],j2[2],j3[2],j4[2];
    static int16_t angles[4];
    int ang_j1,ang_j2, ang_j3, ang_j4;
    sscanf(text_to_convert,"%s %d %s %d %s %d %s %d", j1, &ang_j1, j2, &ang_j2, j3, &ang_j3,
  j4, &ang_j4);
    printf("%s\n",text_to_convert);
    angles[0]=ang_j1;
    angles[1]=ang_j2;
```

```
    angles[2]=ang_j3;
    angles[3]=ang_j4;
    return angles;
}
int16_t *return_angle_inversa(char text_to_convert[]){
//esta funcion lee los valores recibidos de la cinematica directa y devuelve el de j1
    char j1[3],j2[2],j3[2],j4[2];
    static int16_t angles[3];
    int coord_x, coord_y, coord_z, ang_j4;
    sscanf(text_to_convert,"%s %d %s %d %s %d %s %d", j1, &coord_x, j2, &coord_y, j3, &coord_z,
    j4, &ang_j4);
//  printf("%s\n",text_to_convert);
//  printf("%d %d %d %d \n",ang_j1, ang_j2, ang_j3, ang_j4);
    angles[0]=coord_x;
    angles[1]=coord_y;
    angles[2]=coord_z;

    return angles;
}
void USARTSend(const unsigned char *pucBuffer, unsigned long ulCount)
{
    //
    // Loop while there are more characters to send.
    //
    while(ulCount--)
    {
        USART_SendData(USART3, (uint16_t) *pucBuffer++);
        /* Loop until the end of transmission */
        while(USART_GetFlagStatus(USART3, USART_FLAG_TC) == RESET)
        {
        }
    }
}
int i, n=0, m=0, c_cont=0;
uint8_t modo=0;
int16_t X_value=0, Y_value=0, Z_value=0;
char text;
char text_array_n[23], text_array_m[34], text_array_c[26];

void USART3_IRQHandler(void){
    if ((USART3->SR & USART_FLAG_RXNE) != (u16)RESET)
    {
        text = USART_ReceiveData(USART3);

       if(text == 't'){
                    n=0;
                    modo=0; } //Joystick
             else if(text == 'd'){
                    m=0;
                    modo=1; } //Cinematica directa
             else if(text == 'i'){
                    c_cont=0;
                    modo=2; } //Cinematica inversa
             else if(text == 'A'){  modo=3; } //Abrir pinza
             else if(text == 'C'){  modo=4; } //Cerrar pinza
             else if(text == 'O'){  modo=5; } //Apaga la pinza
             else if(text == 'R'){  modo=6; } //Puesta a cero

            if(modo == 0){ //Joystick
                if(n<24){
                            text_array_n[n]=text;
                            n++;
                printf("%s\n",text_array_n);
                }

                angle_bl=return_angle(text_array_n, X_value, Y_value, Z_value);
                printf("%d\n",angle_bl);

            }else if (modo == 1){ //Cinemática directa
                        if(m<35){
                            text_array_m[m]=text;
                            m++;
```

```
                                }
            }else if (modo == 2){ //Cinemática inversa
                        if(c_cont<38){
                            text_array_c[c_cont]=text;
                            c_cont++;
                            }

            }else if (modo == 3){
                printf("Abrir pinza");//Abrir pinza
                         TM_PWM_Init(-90);

            }else if (modo == 4){
                printf("Cerrar pinza");
                            TM_PWM_Init(90);//Cerrar pinza


            }else if (modo == 5){
                printf("OFF pinza");
                            TM_PWM_Init(0);//OFF

            }else if (modo == 6){

            }else{modo=10;}

}}
void bluetooth_init(void){

//Inicialización de estructuras
    GPIO_InitTypeDef     GPIO_InitStruct1;

    GPIO_InitTypeDef     GPIO_InitStruct2;

// Habilitar reloj para GPIOD
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

// Determinar que puerto de comunicación se usa
GPIO_PinAFConfig(GPIOD, GPIO_PinSource8, GPIO_AF_USART3);
GPIO_PinAFConfig(GPIOD, GPIO_PinSource9, GPIO_AF_USART3);
// Configuración de los pines
GPIO_InitStruct1.GPIO_Pin = GPIO_Pin_8;
GPIO_InitStruct1.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct1.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStruct1.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOD, &GPIO_InitStruct1);

GPIO_InitStruct2.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStruct2.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct2.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct2.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStruct2.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOD, &GPIO_InitStruct2);

//Inicialización del puerto de comunicación y de la interrupción
    USART_InitTypeDef USART_InitStruct;
    NVIC_InitTypeDef NVIC_InitStruct;

//Habilitar reloj para el puerto de comunicación
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);

/**
 * Set Baudrate to 9600
 * Disable Hardware Flow control
 * Set Mode To TX and RX, so USART will work in full-duplex mode
 * Disable parity bit
 * Set 1 stop bit
 * Set Data bits to 8
 *
 * Initialize USART3
 * Activate USART3
 */
USART_InitStruct.USART_BaudRate = baudrate;
```

```
USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStruct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_InitStruct.USART_Parity = USART_Parity_No;
USART_InitStruct.USART_StopBits = USART_StopBits_1;
USART_InitStruct.USART_WordLength = USART_WordLength_8b;
USART_Init(USART3, &USART_InitStruct);
USART_Cmd(USART3, ENABLE);

//Habilitar la interrupción para el puerto de comunicación USART3
USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);

//Definir el canal y la prioridad de la interrupción
NVIC_InitStruct.NVIC_IRQChannel = USART3_IRQn;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
NVIC_Init(&NVIC_InitStruct);

}
void usart_rxtx(void){
    const unsigned char welcome_str[] = "Welcome to Bluetooth!\r\n";

    /* Enable USART1 and GPIOA clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);

    /* NVIC Configuration */

    /* Configure the GPIOs */

    /* Configure the USART1 */
        bluetooth_init();

    /* Enable the USART1 Receive interrupt: this interrupt is generated when the
         USART1 receive data register is not empty */
    USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);

    /* print welcome information */
   USARTSend(welcome_str, sizeof(welcome_str));
}
volatile uint8_t fc1=1;
volatile uint8_t fc2=1;
volatile uint8_t fc3=1;

//declaración de mensajes de vuelta cuando se pulsa FC
const unsigned char puesta_cero1[] = "A 1 puesta a 0\r\n";
const unsigned char puesta_cero2[] = "B 2 puesta a 0\r\n";
const unsigned char puesta_cero3[] = "C 3 puesta a 0\r\n";

int16_t stepper1(int16_t angulo, int16_t angulo_ant, int8_t vel_0_100){
    //Funcion para la joint 2: stepper gordo y driver drv8825
    fc1=1;

    double steps=__fabs(3.2*(angulo-angulo_ant))/STEP_ANGLE1;
    int16_t i=steps;

    int16_t steps_vuelta=0;
    uint16_t cycle=vel_0_100*38+200;
    int16_t ang_devuelto=0;

    //Sentido
    if((angulo-angulo_ant)<0){GPIO_SetBits(GPIOD,GPIO_Pin_5);} //Eje Z dirección hacia dentro
        else {GPIO_ResetBits(GPIOD,GPIO_Pin_5);} //Eje Z dirección del palito


    while(i>0 && fc1==1){
        if(i<100){
            GPIO_SetBits(GPIOD,GPIO_Pin_7);
            delay_micros(1.3*cycle);
            GPIO_ResetBits(GPIOD,GPIO_Pin_7);
            delay_micros(1.3*cycle);

        } else{ GPIO_SetBits(GPIOD,GPIO_Pin_7);
```

```
                        delay_micros(cycle);
                        GPIO_ResetBits(GPIOD,GPIO_Pin_7);
                        delay_micros(cycle);}
    i--;
    }
    //Si se pulsa el FC
    if(fc1==0){
        USARTSend(puesta_cero1, sizeof(puesta_cero1));
        ang_devuelto=-85;
        GPIO_ResetBits(GPIOD,GPIO_Pin_5);
        while(steps_vuelta<44){
            GPIO_SetBits(GPIOD,GPIO_Pin_7);
            delay_micros(cycle);
            GPIO_ResetBits(GPIOD,GPIO_Pin_7);
            delay_micros(cycle);
            steps_vuelta++;
        }
        //Envia mensaje de vuelta de puesta a 0

        fc1=1;

        return ang_devuelto;

    }else if(angulo-angulo_ant==0){
        return angulo;

    }else if(angulo-angulo_ant>0){
        ang_devuelto=(0.3125*(steps)*STEP_ANGLE2+angulo_ant);
        return ang_devuelto;

    }else if(angulo-angulo_ant<0){
        ang_devuelto=(angulo_ant-(0.3125*steps)*STEP_ANGLE2);
        return ang_devuelto;

}
}
int16_t stepper2(int16_t angulo, int16_t angulo_ant, int8_t vel_0_100){
    //Funcion para la joint 2: stepper Nema 24 y driver drv8825
    fc2=1;
    // ángulo máximo permitido
    if(angulo>210){angulo=210;}
    //cáculo del número de pasos
    double steps=__fabs(angulo-angulo_ant)/STEP_ANGLE2;
    int16_t i=steps;

    int16_t steps_vuelta=0;
    //cálculo del periodo según la velocidad
    uint16_t cycle=vel_0_100*38+200;
    int16_t ang_devuelto=0;

    //Sentido
    if((angulo-angulo_ant)<0){GPIO_SetBits(GPIOD,GPIO_Pin_1);} //Giro hacia atrás
        else {GPIO_ResetBits(GPIOD,GPIO_Pin_1);} //Giro en sentido normal

    //Bucle para mover
    while(i>0 && fc2==1){
        if(i<100){
            GPIO_SetBits(GPIOD,GPIO_Pin_3);
            delay_micros(1.3*cycle);
            GPIO_ResetBits(GPIOD,GPIO_Pin_3);
            delay_micros(1.3*cycle);

        } else{ GPIO_SetBits(GPIOD,GPIO_Pin_3);
                        delay_micros(cycle);
                        GPIO_ResetBits(GPIOD,GPIO_Pin_3);
                        delay_micros(cycle);}
    i--;
    }

    //Si se pulsa el FC
    if(fc2==0){
        //Envia mensaje de vuelta que esta a 0
```

```
493.        USARTSend(puesta_cero2, sizeof(puesta_cero2));
494.        ang_devuelto=29;//angulo real 29º
495.        //Realiza una vuelta de 5º
496.        GPIO_ResetBits(GPIOD,GPIO_Pin_1);
497.        while(steps_vuelta<44){
498.            GPIO_SetBits(GPIOD,GPIO_Pin_3);
499.            delay_micros(cycle);
500.            GPIO_ResetBits(GPIOD,GPIO_Pin_3);
501.            delay_micros(cycle);
502.            steps_vuelta++;
503.        }
504.
505.
506.        fc2=1;
507.        return ang_devuelto;
508.
509.    }else if(angulo-angulo_ant==0){
510.        //si se pide el mismo ángulo en el que se encuentra devuelve el angulo
511.        return angulo;
512.
513.    }else if(angulo-angulo_ant>0){
514.        //si el sentido de giro es normal: cálculo del angulo devuelto
515.        ang_devuelto=(steps)*STEP_ANGLE2+angulo_ant;
516.        return ang_devuelto;
517.
518.    }else if(angulo-angulo_ant<0){
519.        //si el sentido de giro es contrario
520.        ang_devuelto=angulo_ant-(steps)*STEP_ANGLE2;
521.        return ang_devuelto;
522.
523.}
524.}
525.int16_t stepper3(int16_t angulo, int16_t angulo_ant, int8_t vel_0_100){
526.    //Funcion para la joint 2: stepper Nema 24 y driver drv8825
527.    fc3=1;
528.    // ángulo máximo permitido
529.    if(angulo>310){angulo=310;}
530.    //cáculo del número de pasos
531.    double steps=__fabs(angulo-angulo_ant)/STEP_ANGLE2;
532.    int16_t i=steps;
533.
534.    int16_t steps_vuelta=0;
535.    //cálculo del periodo según la velocidad
536.    uint16_t cycle=vel_0_100*38+200;
537.    int16_t ang_devuelto=0;
538.
539.    //Sentido
540.    if((angulo-angulo_ant)<0){GPIO_SetBits(GPIOD,GPIO_Pin_2);} //Giro hacia atrás
541.        else {GPIO_ResetBits(GPIOD,GPIO_Pin_2);;} //Giro en sentido normal
542.
543.    //Bucle para mover
544.    while(i>0 && fc3==1){
545.        if(i<100){
546.            GPIO_SetBits(GPIOD,GPIO_Pin_4);
547.            delay_micros(1.3*cycle);
548.            GPIO_ResetBits(GPIOD,GPIO_Pin_4);
549.            delay_micros(1.3*cycle);
550.
551.        } else{ GPIO_SetBits(GPIOD,GPIO_Pin_4);
552.                        delay_micros(cycle);
553.                        GPIO_ResetBits(GPIOD,GPIO_Pin_4);
554.                        delay_micros(cycle);;}
555.    i--;
556.    }
557.
558.    //Si se pulsa el FC
559.    if(fc3==0){
560.        //Envia mensaje de vuelta que esta a 0
561.        USARTSend(puesta_cero3, sizeof(puesta_cero3));
562.        ang_devuelto=29;//angulo real 29º
563.        //Realiza una vuelta de 5º
564.        GPIO_ResetBits(GPIOD,GPIO_Pin_2);
```

```
565.        while(steps_vuelta<44){
566.            GPIO_SetBits(GPIOD,GPIO_Pin_4);
567.            delay_micros(cycle);
568.            GPIO_ResetBits(GPIOD,GPIO_Pin_4);
569.            delay_micros(cycle);
570.            steps_vuelta++;
571.        }
572.
573.
574.        fc3=1;
575.        return ang_devuelto;
576.
577.    }else if(angulo-angulo_ant==0){
578.        //si se pide el mismo ángulo en el que se encuentra devuelve el angulo
579.        return angulo;
580.
581.    }else if(angulo-angulo_ant>0){
582.        //si el sentido de giro es normal: cálculo del angulo devuelto
583.        ang_devuelto=(steps)*STEP_ANGLE2+angulo_ant;
584.        return ang_devuelto;
585.
586.    }else if(angulo-angulo_ant<0){
587.        //si el sentido de giro es contrario
588.        ang_devuelto=angulo_ant-(steps)*STEP_ANGLE2;
589.        return ang_devuelto;
590.
591.}
592.}
593.uint8_t pulsador_PA0=0;
594.int16_t *c;
595.double q1=0.0, q2=0.0, q3=0.0;
596.int i=0;
597.
598.int main(void){
599.      pin_inicializar();
600.        interrupt_config();
601.
602.
603.    usart_rxtx();
604.    angle_ant_J1=stepper1(-360,0,50);
605.    angle_ant_J3=stepper3(-360,0,50);
606.    angle_ant_J2=stepper2(-360,0,50);
607.
608.
609.TM_TIMER_Init();
610.    while (1) {
611.
612.        if(modo==0){
613.             if(pulsador_PA0==1){
614.
615.             GPIO_ResetBits(GPIOD, GPIO_Pin_12);
616.             GPIO_ResetBits(GPIOD, GPIO_Pin_13);
617.             GPIO_ResetBits(GPIOD, GPIO_Pin_14);
618.
619.                        angle_ant_J1=stepper1(angle_bl,angle_ant_J1,50);
620.                        angle_ant_J3=stepper3(angle_bl,angle_ant_J3,50);
621.                        angle_ant_J2=stepper2(angle_bl,angle_ant_J2,50);
622.
623.
624.             delay_ms(200);
625.         }
626.
627.
628.         }else if(modo==1){
629.                        //enviamos las coordenadas directamente
630.            c=return_angle_directa(text_array_m);
631.             printf("%d %d %d %d \n", c[0], c[1], c[2], c[3]);
632.             angle_ant_J1=stepper1(c[0],angle_ant_J1,vel_stepper1);
633.             angle_ant_J2=stepper2(c[1],angle_ant_J2,vel_stepper2);
634.             angle_ant_J3=stepper3(c[2],angle_ant_J3,vel_stepper3);
635.
636.             printf("ang2= %d ang3= %d\n", angle_ant_J2, angle_ant_J3);
```

```
637.             delay_ms(200);
638.
639.          }else if(modo==2){
640.
641.             if(pulsador_PA0==1){
642.                                                  //PRUEBA CIN INVERSA
643.                 printf("%s\n", text_array_c);
644.             c=return_angle_inversa(text_array_c);
645.          printf("%d %d %d %d \n", c[0], c[1], c[2], c[3]);
646.//              printf("%f %f %f\n", pra[0],pra[1],pra[2]);
647.
648.         q1=q1_inversa(c[0],c[1],c[2]);
649.
650.         q2=q2_inversa(c[0],c[1],c[2],q3);
651.                 q3=q3_inversa(c[0],c[1],c[2]);
652.
653.         printf("ang1= %f ang2 = %f  ang3 = %f\n",q1,q2,q3);
654.         delay_ms(1000);
655.
656.
657.             angle_ant_J1=stepper1(q1,angle_ant_J1,25);
658.
659.              angle_ant_J2=stepper2(q2,angle_ant_J2,50);
660.                   angle_ant_J3=stepper3(180-q3,angle_ant_J3,50);
661.
662.              delay_ms(200);
663.
664.
665.          }}else if(modo==6){
666.
667.          angle_ant_J1=stepper1(-360,0,25);
668.         angle_ant_J2=stepper2(-360,0,100);
669.         angle_ant_J3=stepper3(-360,0,100);
670.          }
671.
672.     }
673.
674.   return(0);
675.}
```

MATRICES.C

```
1.  #include <stdio.h>
2.  #include <stm32f4xx.h>
3.  #include "stm32f4xx_rcc.h"
4.  #include "stm32f4xx_gpio.h"
5.  #include "stm32f4xx_adc.h"
6.  #include "RTE_Components.h"
7.  #include <math.h>
8.
9.  #define COL 1
10. #define FIL 3
11. #define L1 131
12. #define L2 160
13. #define L3 275
14. #define Lx 130
15. #define Pi 3.14159
16.
17.
18. double q1_inversa(int16_t px, int16_t py, int16_t pz){
19.     double x=px, z=pz-L1, y=py;
20.     double q1=0, sum=(x*x)+(y*y)-(Lx*Lx);
21.     double r=sqrt(sum);
22.     q1=atan2(y,x)+atan2(Lx,r);
23.     q1=180*(q1)/Pi;
24.     return q1;
25.
26. }
27.
28. double q2_inversa(int16_t px, int16_t py, int16_t pz, double q3){
29.
```

```
30.     double x=px, z=pz-L1, y=py;
31.     double alpha=0, q2=0,sum2=0, sum=(x*x)+(y*y)-(Lx*Lx);
32.     double r=sqrt(sum);
33.     double beta=atan2(z,r);
34.     q3=-acos((r*r+z*z-L3*L3-L2*L2)/(2*L3*L2));
35.
36.
37.     alpha=atan2((L3*sin(q3)),(L2+L3*cos(q3)));
38.     printf("%f",alpha);
39.
40.     q2=beta-alpha;
41.
42.     q2=180*q2/Pi;
43. //  if(px<0){q2=180-q2;}
44.     return q2;
45. }
46. double q3_inversa(int16_t ppx, int16_t ppy, int16_t ppz){
47.
48.     double z=ppz-L1, y=ppy, x=ppx;
49.     double sum=0, q3=0;
50.
51.     sum=(x*x)+(y*y)-(Lx*Lx);
52.     double r=sqrt(sum);
53.
54.     q3=-acos((r*r+z*z-L3*L3-L2*L2)/(2*L3*L2));
55.     q3=180*q3/Pi;
56. //  if(px<0){q3=-q3;}
57.
58.     return q3;
59. }
60. void printf_matriz(float A[][3], uint8_t f, uint8_t c){
61.
62.     int8_t i, j;
63.     for(i=0; i<f; i++){
64.         for(j=0; j<c; j++){
65.             printf("%.2f ", A[i][j]);
66.         }
67.         printf("\n");
68.     }
69.
70.
71. }
```

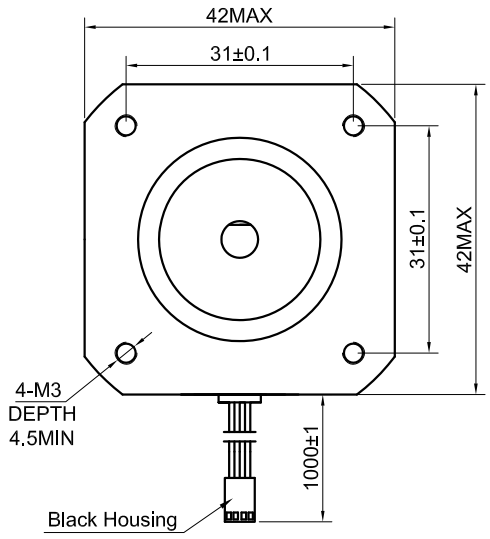
Anexo II: Datasheet motor paso a paso Nema 17

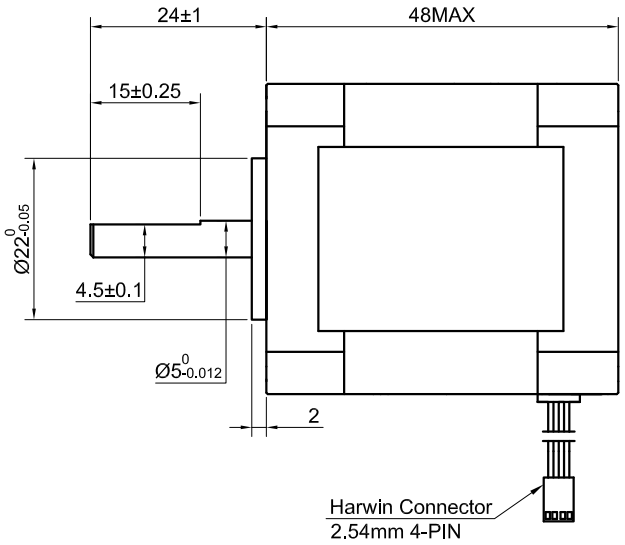
Anexo III: Datasheet motor paso a paso Nema 24

Anexo IV: Datasheet fuente alimentación

Anexo V: Datasheet servo

Anexo VI: Datasheet driver DRV8825



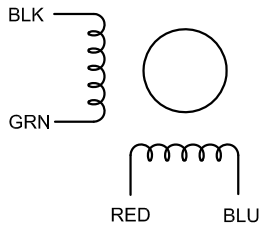


SPECIFICATION / CONNECTION	BIPOLAR
AMPS/PHASE	2.00
RESISTANCE/PHASE(Ohms)@25°C	1.40±10%
INDUCTANCE/PHASE(mH)@1KHz	3.00±20%
HOLDING TORQUE(Nm)[lb-in]	0.59[5.22]
STEP ANGLE(°)	1.80
STEP ACCURACY(NON-ACCUM)	±5.00%
ROTOR INERTIA(g-cm²)	82.00
WEIGHT(Kg)[lb]	0.40[0.88]
TEMPERATURE RISE:MAX.80°C（MOTOR STANDSTILL;FOR 2PHASE ENERGIZED）	
AMBIENT TEMPERATURE -10°C~50°C[14°F~122°F]	
INSULATION RESISTANCE 100 Mohm（UNDER NORMAL TEMPERATURE AND HUMIDITY）	
INSULATION CLASS B 130°C[266°F]	
DIELECTRIC STRENGTH 500VAC FOR 1MIN.(BETWEEN THE MOTOR COILS AND THE MOTOR CASE)	
AMBIENT HUMIDITY MAX.85%(NO CONDENSATION)	

TYPE OF CONNECTION (EXTERN)		MOTOR	
PIN NO	BIPOLAR	LEADS	WINDING
1	A —	BLK	A
2	A\ —	GRN	A\
3	B —	RED	B
4	B\ —	BLU	B\

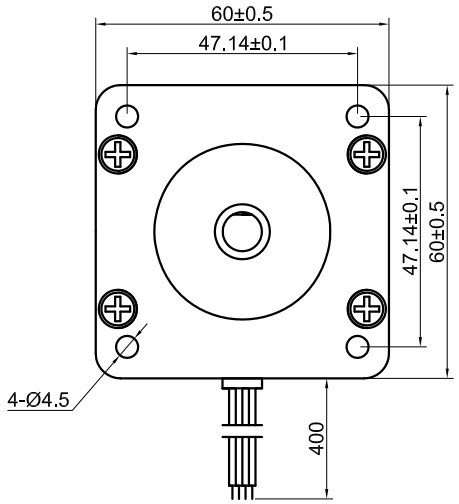
FULL STEP 2 PHASE-Ex. ,
WHEN FACING MOUNTING END (X)

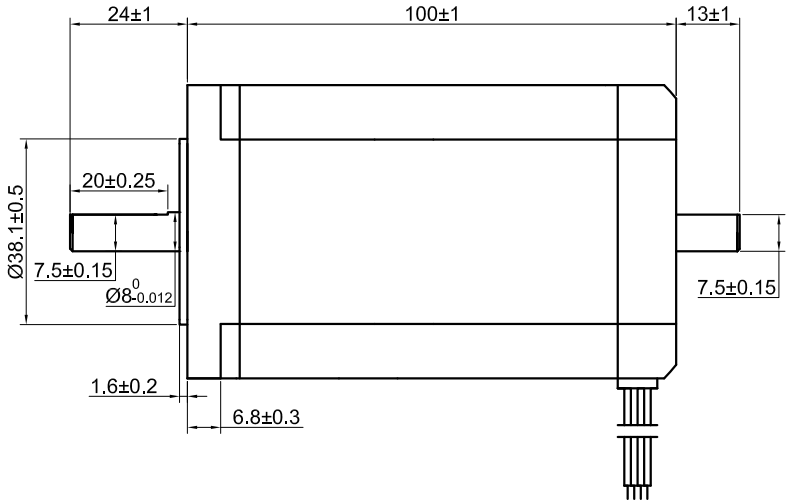
STEP	A	B	A\	B\		
1	+	+	-	-	↓	CCW ↑
2	-	+	+	-		
3	-	-	+	+		
4	+	-	-	+	CW	



STEPPERONLINE®

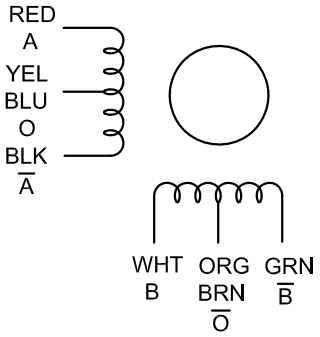
	APVD		8.18.2018	STEPPER MOTOR
	CHKD			
1:1	DRN			17HS19-2004S1
SCALE	SIGNATURE		DATE	



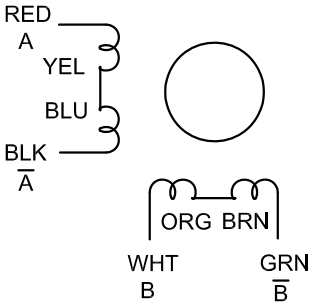


SPECIFICATION / CONNECTION	UNIPOLAR	BIPOLAR (SERIAL)	BIPOLAR (PARALLEL)
AMPS/PHASE	3.00	2.12	4.24
RESISTANCE/PHASE(Ohms)@25°C	1.40±10%	2.80±10%	0.70±10%
INDUCTANCE/PHASE(mH)@1KHz	3.00±20%	12.00±20%	3.00±20%
HOLDING TORQUE(Nm)[lb-in]	2.80[24.78]	4.00[35.40]	4.00[35.40]
STEP ANGLE(°)	1.80		
STEP ACCURACY(NON-ACCUM)	±5.00%		
ROTOR INERTIA(g-cm²)	980.00		
WEIGHT(Kg)[lb]	1.60[3.53]		
TEMPERATURE RISE:MAX.80°C（MOTOR STANDSTILL;FOR 2PHASE ENERGIZED）			
AMBIENT TEMPERATURE -10°C~50°C[14°F~122°F]			
INSULATION RESISTANCE 100 Mohm（UNDER NORMAL TEMPERATURE AND HUMIDITY）			
INSULATION CLASS B 130°C[266°F]			
DIELECTRIC STRENGTH 500VAC FOR 1MIN.(BETWEEN THE MOTOR COILS AND THE MOTOR CASE)			
AMBIENT HUMIDITY MAX.85%(NO CONDENSATION)			

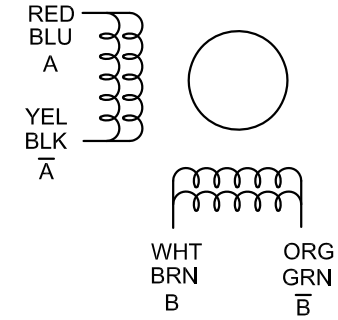
UNIPOLAR



BIPOLAR SERIES



BIPOLAR PARALLEL



	APVD		8.18.2018	STEPPER MOTOR
	CHKD			
1:1.5	DRN			24HS39-3008D
SCALE	SIGNATURE		DATE	

350W Power Supply S-350-36

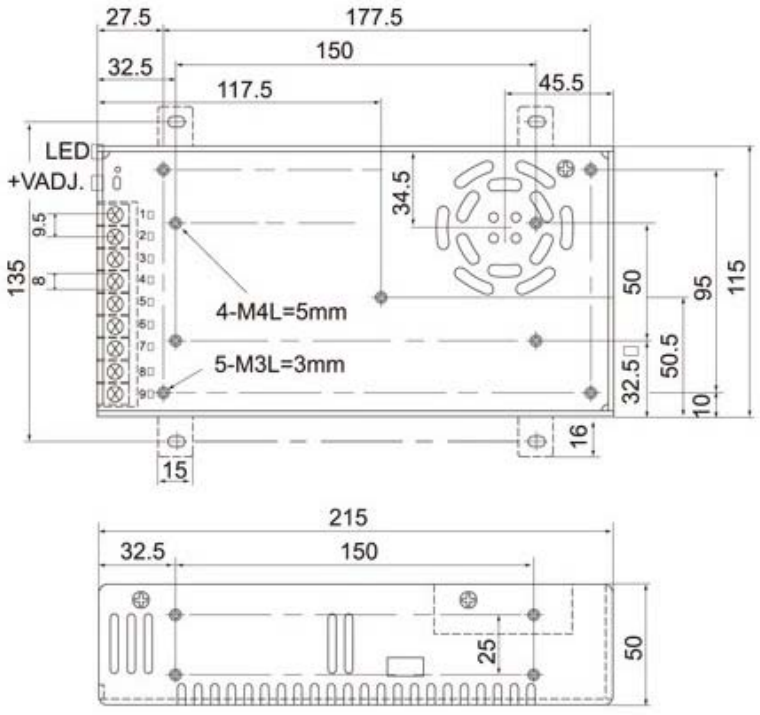
Features:

- 36V DC 9.8A output
- AC input voltage range: 93~132V/176~264VAC
- 115V/230V AC selected by switch
- High efficiency low cost
- Forced air cooling by built-in DC fan
- Low output ripple and yawp
- Over current, over voltage, short circuit and overheat protections
- 215*115*50mm（L*W*H）

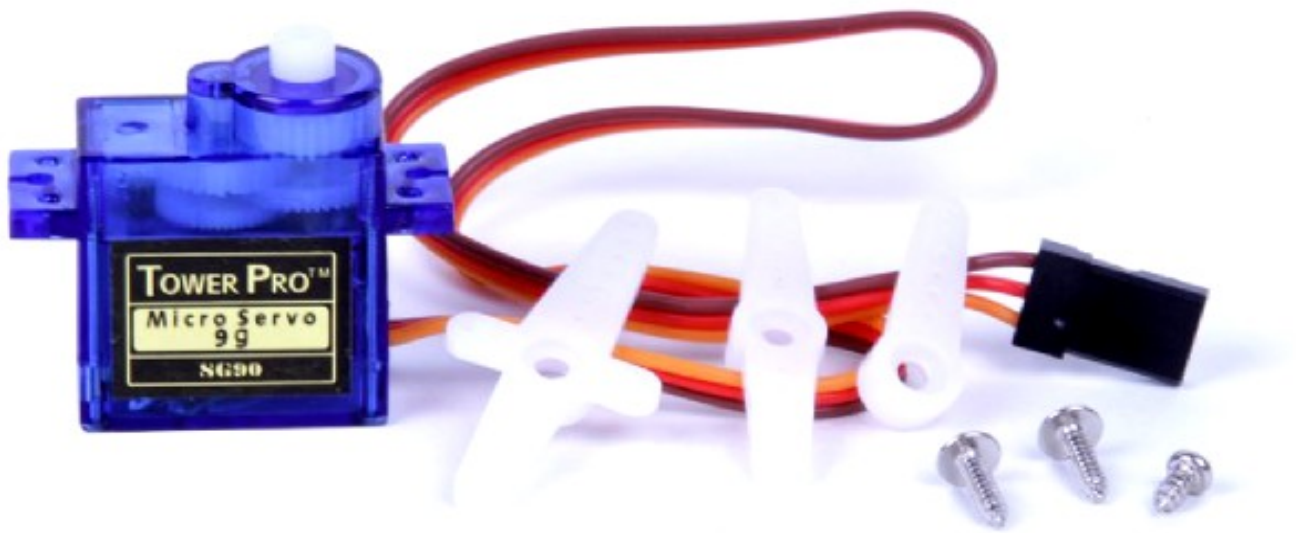
General Specification:

Model	S-350-36
DC Output	36V 9.8A
Wave and Noise	120mVp-p
Inlet Stability	±0.1%
Load Stability	±0.3%
Efficiency	85%
Adjustable range for DC Voltage	32V~39V
AC Input Voltage	93~132V/176~264VAC Slected by Switch
AC Input Current	3.54A/115VAC 1.73A/230VAC
Working Temperature	-10~50°C
Safety Standards	GB4943, UL60950, EN60950
EMC Standards	GB9254, 55022, ClassB
Weight	1.1kg

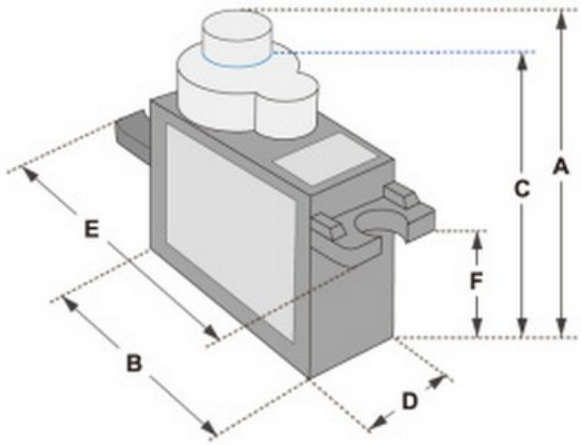
Dimensions:



SERVO MOTOR SG90 DATA SHEET

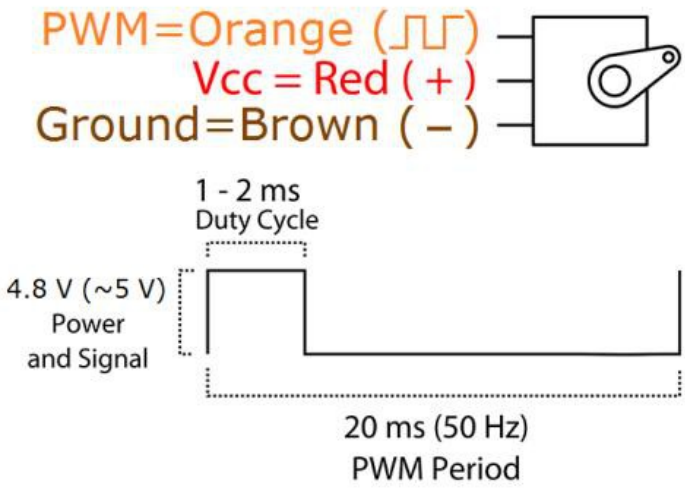


Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

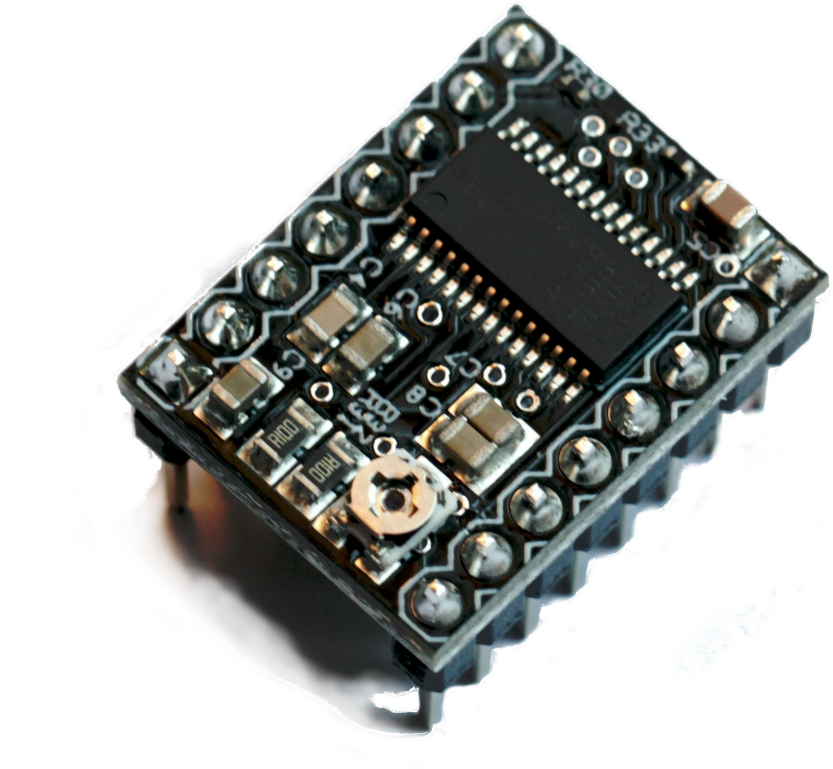


Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.



STEPSTICK DRV8825 v1.0 DATASHEET



Author	Bart Meijer
Date	21st of October 2013
Document version	1.0

ReprapWorld.com

PRODUCT OVERVIEW

The stepstick DRV8825 is a breakout board for the Texas Instruments DRV8825 stepper motor controller. You can use this board to act as interface between your microcontroller and stepper motor. The DRV8825 is able to deliver up to 2.5A and can be controlled with a simple step/direction interface. The controller has a resolution of min. 1/32 step and protective features for over-current, short circuit and over-temperature. See the DRV8825 Datasheet for details on the DRV8825 controller.

The stepstick DRV8825 supersedes the stepstick A4988, which has been discontinued. The aim is for the stepstick DRV8825 to be a drop-in replacement for Stepstick A4988.

SAFETY WARNINGS

Always disconnect the power source from the board before unplugging the stepper motor and/or adjusting the current. Failure to do so may result in permanent damage to the board and/or injuries due to high voltage spikes.

The stepper driver may get **HOT**, do not touch the device until it had a few minutes to cool down after operation.

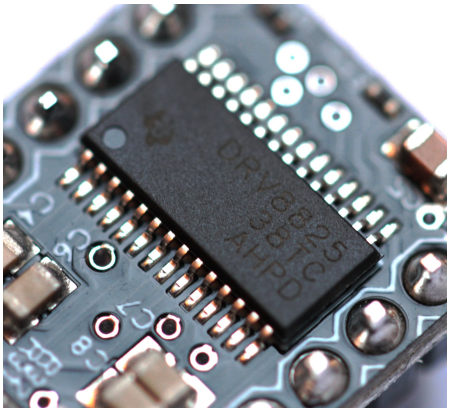
It is recommended to drive the stepper motor on current as low as possible to reduce power consumption and increase lifespan.

It is **NOT** recommended to turn the stepper motor while connected to the electronics. While turning the stepper motor, large voltages may be emitted through the VMOT pin, which can damage the electronics.

TECHNICAL SPECIFICATION

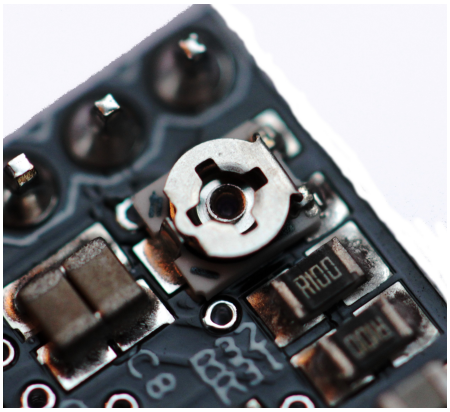
Controller	DRV8825
Operating Voltage (logic)	3-5.25V
Operating Voltage (vmot)	12-24V
Max current	2.5A
Dimensions	20.4x15.6mm

MAJOR FEATURES



DRV8825
Powerful DRV8825 with

- High current driver capable up to 2.5A
- Six different step resolutions: full-step, half-step, 1/4-step, 1/8-step, 1/16-step, and 1/32-step
- Protection against over-temperature and over-current
- No logic voltage required



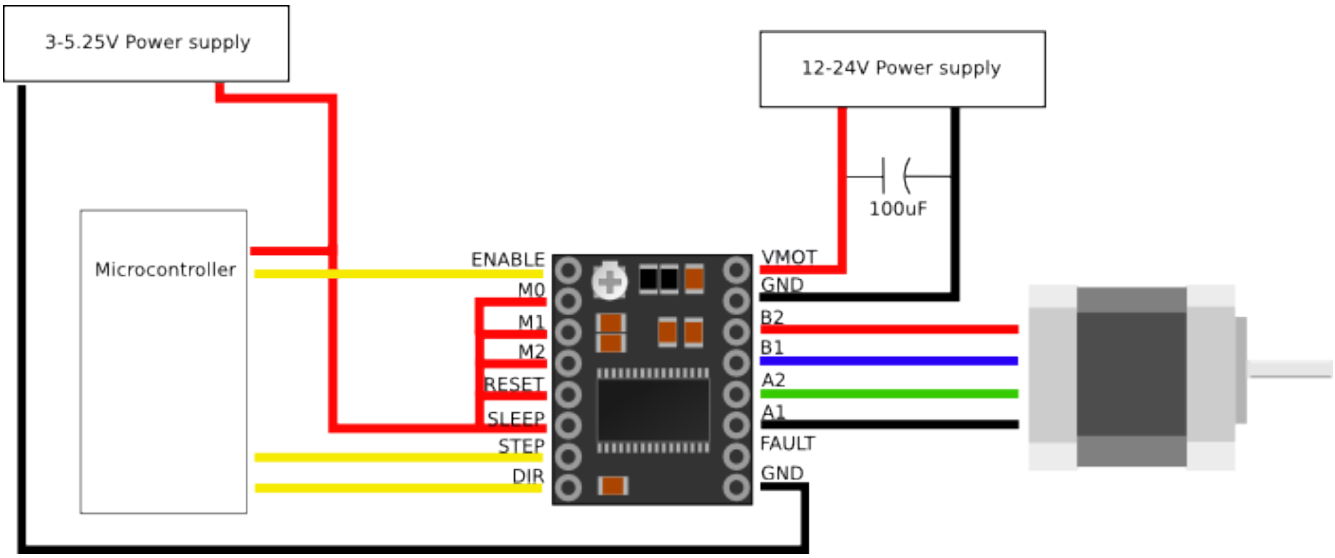
Adjustable current
Using the trimpot on the board you can easily turn the current up or down. Turn left to lower the current, right to output a higher current.

OTHER FEATURES

- Four layer high quality PCB board
- Pre-soldered, no need to solder the headers

EXAMPLE CIRCUIT

The following diagram lists the pins and show an example circuit.



Name	Description
Enable	Enable/disable the stepper driver HIGH – Disable LOW – Enable *
M0 – M2	Step resolution setting, see chapter 'step resolution configuration'
RESET	Enable/disable the H-bridge output * LOW – Disable * HIGH – Enable
SLEEP	Enable/disable low-power sleep mode LOW – Sleep * HIGH – Active
STEP	LOW → HIGH, move one step
DIR	LOW / HIGH switches direction
VMOT	Motor power (12-24V)
GND	System ground
FAULT	LOW when the stepper driver is in fault condition. You can provide 5V on this pin for compatibility with stepstick A4988

* this is the default state when the pin is not connected

STEP RESOLUTION CONFIGURATION

The DRV8825 had six step resolution modes, which can be configured using the M0-M2 pins on the stepstick DRV8825. The following table lists the step resolution settings:

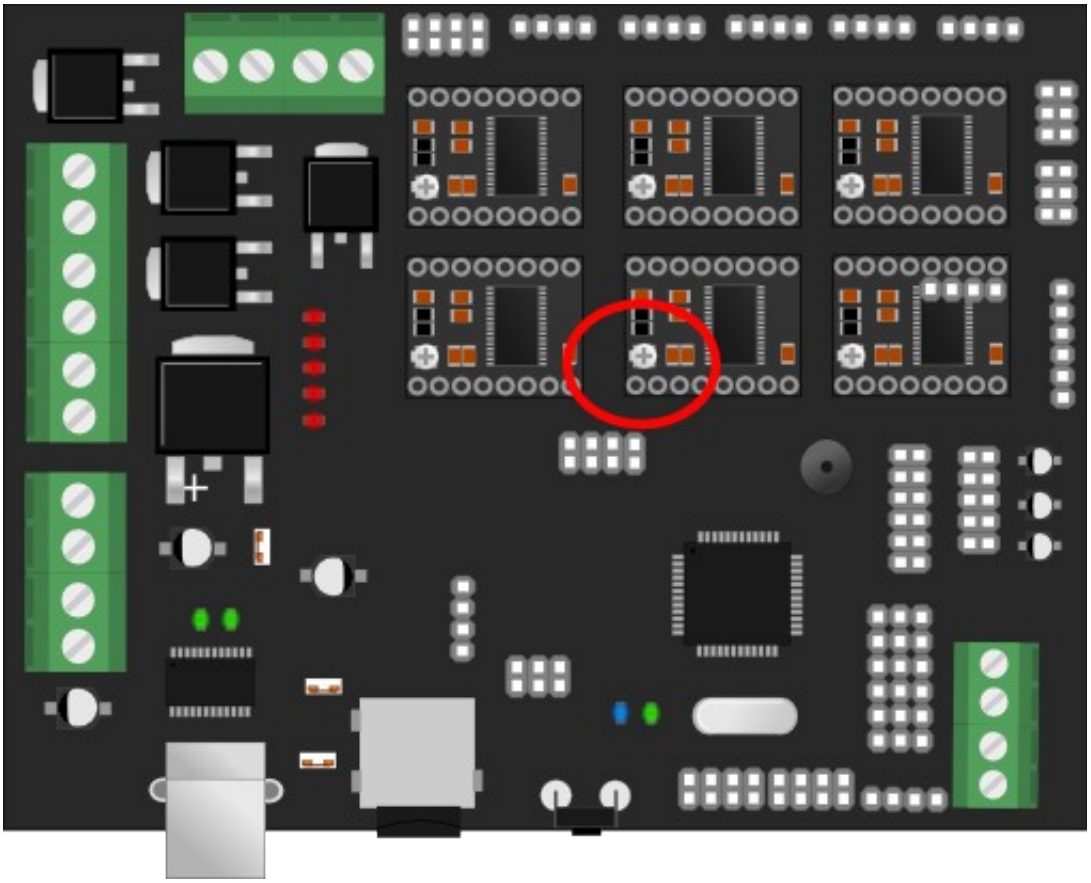
M0	M1	M2	Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	1/4 step
High	High	Low	1/8 step
Low	Low	High	1/16 step
High	Low	High	1/32 step
Low	High	High	1/32 step
High	High	High	1/32 step

COMPATIBILITY WITH REPRAP HARDWARE

The stepstick DRV8825 should be compatible with most RepRap hardware available. The following images show how to insert the stepstick DRV8825 in the most populair RepRap hardware.

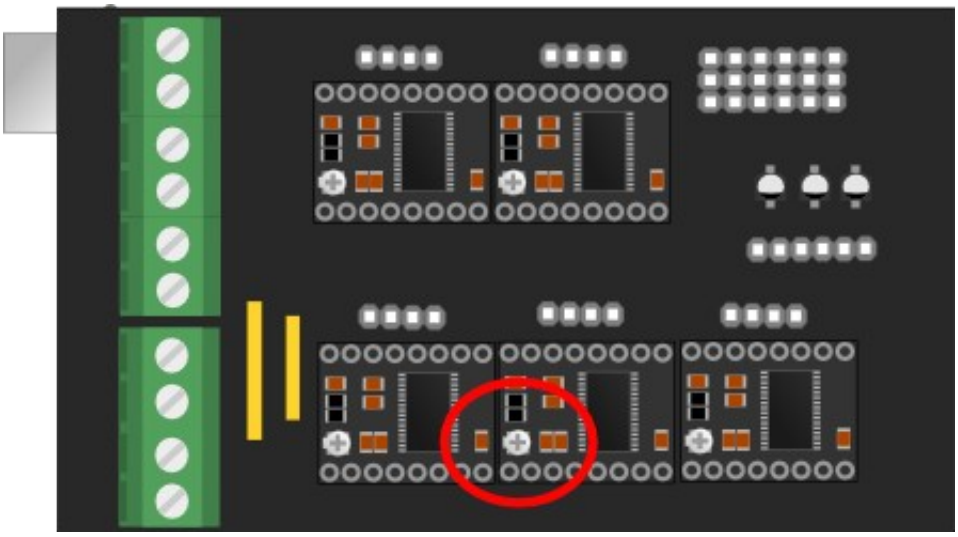
Megatronics

The orientation of the stepstick is indicated with the trimpot, marked with a red circle.



RAMPS

The orientation of the stepstick is indicated with the trimpot, marked with a red circle.



TROUBLESHOOTING

The stepper motor does old torque (you can rotate the stepper motor by hand)	- Check if the power is enabled - Is the ENABLE pin LOW? - Are SLEEP and RESET put HIGH?
The motor is losing steps while turning	- When the chip overheats the thermal protection will disable the device. Turn down the trimpot. - You may have not enough torque, turn the trimpot to the right to increase current.