UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Rediseño, implementación y control de un brazo robot portátil

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

AUTOR/A: Walter Rodríguez, Mathias Camill

Tutor/a: Zotovic Stanisic, Ranko

CURSO ACADÉMICO: 2022/2023

Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería del Diseño

REDISEÑO, IMPLEMENTACIÓN Y CONTROL DE UN BRAZO ROBOT PORTÁTIL

Trabajo de Fin de Master

Master en Ingeniería Mecatrónica

Autor: Mathias Camill Walter Rodríguez

Tutor: Ranko Zotovic Stanisic

Curso Académico: 2022/2023

Agradecimientos

A Ranko, por su ayuda, apoyo y exporto conocimiento a la hora de realizar este trabajo, y por proporcionarme la oportunidad de aportar mi granito de arena a su elaboración.

A Roland y Jose, por asentar las bases sobre las que se apoya todo lo aquí dearrollado.

A mi familia.

Resumen

El trabajo a continuación presentado aborda el proceso de mejora de diseño y control de un brazo robot portátil de bajo coste cuya función es asistir a usuarios discapacitados en tareas cotidianas, como comer, beber etc.

Es ya la tercera iteración del diseño de este brazo. En cuanto a rediseño, se trata de modificar y mejorar algunas cosas de las versiones previas. Se ha visto que en algunas articulaciones aparecen holguras. No están previstos los sitios para los fines de carrera, etc. Se deberán rediseñar y volver a fabricar eslabones enteros.

Por otra parte, en la versión actual están hecho solo el control elemental, punto a punto con un regulador PD. Se deberán implementar el movimiento coordinado, control con compensación de gravedad, generadores de trayectoria más complejos, posibilidad de comunicación con un nivel superior, interrupciones fin de carrera, homing, etc.

Abstract

The work presented below addresses the process of improving the design and control of a low-cost portable robot arm whose function is to assist disabled users in daily tasks, such as eating, drinking, etc.

It is already the third iteration of this arm's design. As for redesign, it is about modifying and improving some things from previous versions. It has been seen that looseness appears in some joints. Sites are not planned for race ends, etc. Entire links will have to be redesigned and remanufactured.

Furthermore, in the current version only elementary control is done, point-to-point with a PD regulator. Coordinated movement, control with gravity compensation, more complex trajectory generators, possibility of communication with a higher level, end-of-stroke interruptions, homing, etc. should be implemented.

Tabla de contenido

Índice de figuras

Índice de tablas

1. Introducción

1.1. Planteamiento

En este trabajo se desarrollará el proceso de diseño e implementación de mejoras aplicadas sobre el prototipo ya existente de un brazo robótico portátil, conceptualizado con el objetivo de ser utilizado por personas que sufran algún tipo de discapacidad en un brazo (o ambos).

El brazo robótico en sí se trata de un robot de 5 grados de libertad, diseñado para colocarse sobre el hombro del usuario, con piezas principalmente elaboradas mediante técnicas de manufacturación aditiva, perteneciendo estas al conjunto de procesos de manufactura conocidas como impresión 3D, tanto FDM (modelado por deposición de filamento) como SLA (sinterizado selectivo por láser); siendo capaz de levantar cargas de hasta 2kg mediante 5 motores DC sin escobillas.

De las mejoras desarrolladas a lo largo de este trabajo, las más destacables son el rediseño de prácticamente todas las piezas de manufacturado aditivo; la fabricación de placas de circuito para facilitar y asegurar la conexión y el cableado de todos los componentes electrónicos del brazo; la introducción de finales de carrera, tanto por motivos de seguridad como para añadir una función de homing (o retorno a 0); reescritura completa de la arquitectura de programa del microcontrolador utilizado, añadiendo funciones como el homing ya mencionado anteriormente; trayectorias lineales en espacio cartesiano o compensación de la gravedad en cada una de las articulaciones del brazo.

1.2. Alcance

Resulta obvio que el producto aquí desarrollado está orientado a usuarios que presenten algún tipo de discapacidad motriz en uno o ambos brazos (o manos), de forma que, a la hora de obtener información acerca de la posible

viabilidad que tenga el brazo, convendrá hacer un estudio demográfico a diversas escalas (regional, estatal, internacional…) mediante el cual se podría saber cuántas personas podrían beneficiarse de su uso.

En cuanto al nivel estatal, según un informe del Instituto Nacional de Estadística realizado en 1999 [1]; en España hay alrededor de un millón de personas que presentan algún tipo de discapacidad a la hora de utilizar brazos o manos de las cuales 447,985 se encuentran en el rango de edad de entre 6 y 64 años; y 644,887 son mayores de 65. Así pues, eso significa que un 2,7% de la población español en 1999 se beneficiaría del producto desarrollado a lo largo de este trabajo.

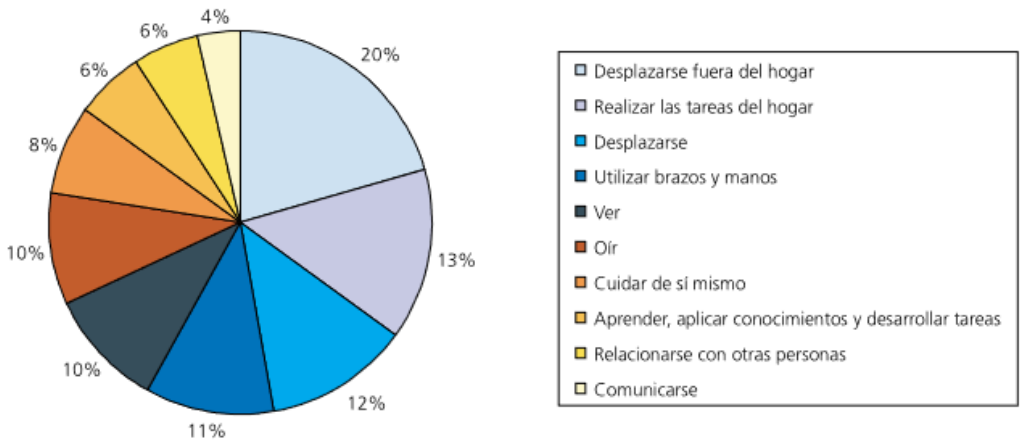


Ilustración 1: Distribución de discapacidades de la población española en 1999 por tipo de discapacidad

En cuanto al nivel internacional, se pueden encontrar facilmente datos relevantes en estudios realizados por organismos pertenecientes a la unión europea, destacando entre ellos la oficina estadística de la propia Unión Europea, conocida como Eurostat [2]. Así pues, según los datos recopilados por la propia Eurostat, aproximadamente el 3% de la población de la unión europea de entre 15 y 64 presenta problemas de salud relacionados con brazos y manos.

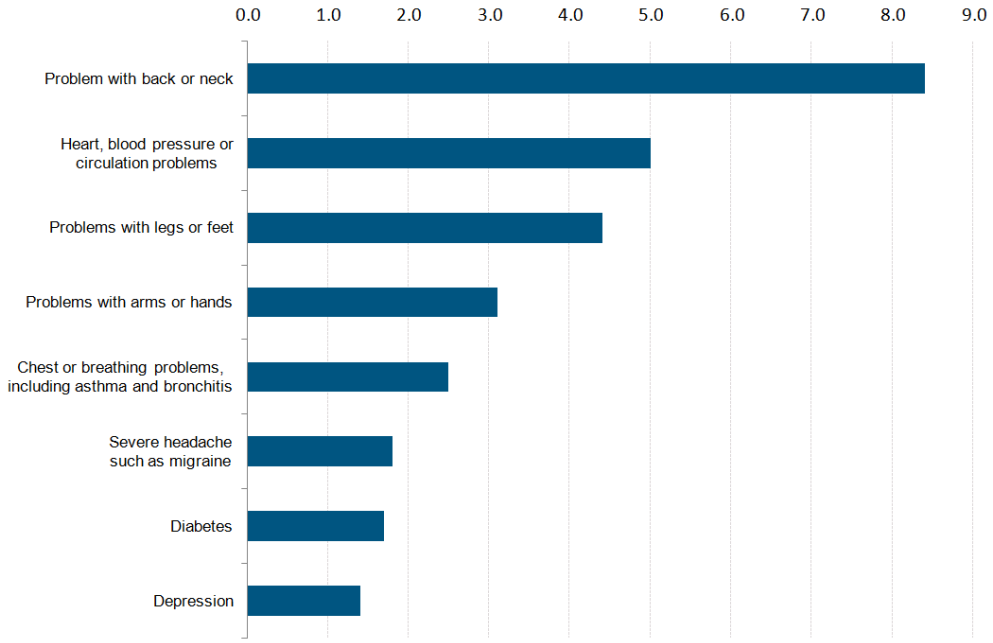


Ilustración 2: Prevalencia de problemas de salud de larga duración en personas de 15 a 64 años en la UE (2011)

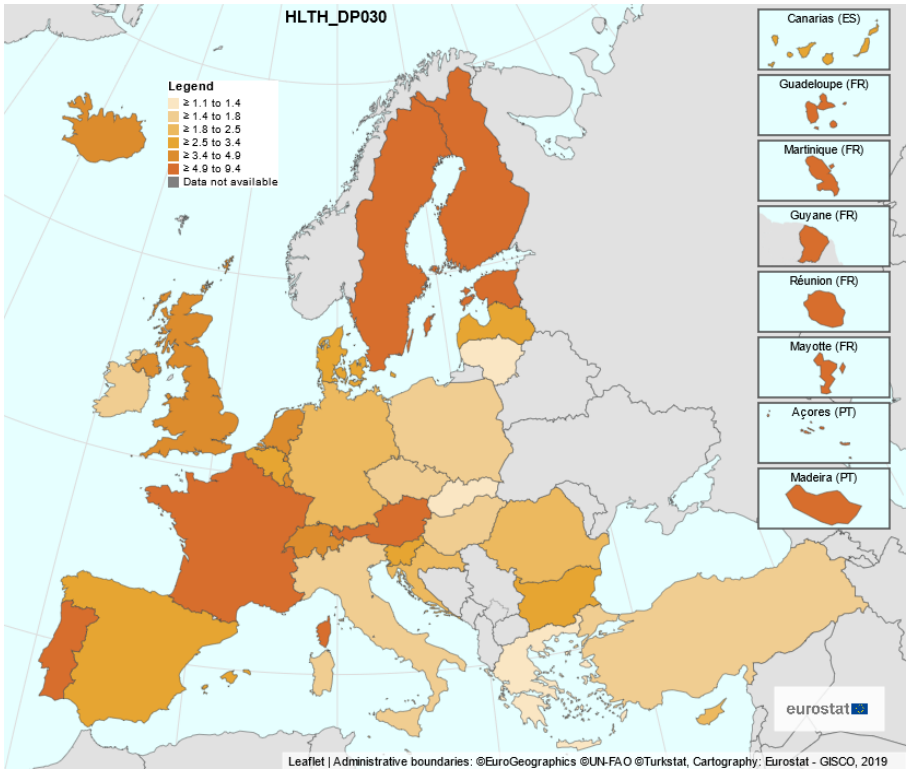


Ilustración 3: Prevalencia de problemas de salud de larga duración relacionados con brazos o manos en personas de 15 a 64 años en la UE por país (2011)

Así pues, se puede extrapolar que hay una gran cantidad de posibles usuarios que podrían beneficiarse de un mecanismo como el desarrollado en

este trabajo. Además, la naturaleza de las técnicas de manufacturado aditivo permite una gran flexibilidad a la hora de adaptar el diseño del brazo robot a un mayor abanico de usuarios y capacidades distintas.

1.3. Estado del arte

No hay demasiados robots portátiles desarrollados para miembros superiores, y la mayoría de ellos son exoesqueletos. Se pueden encontrar compilaciones recientes en [3] y [4]. Ambos están dedicados únicamente a exoesqueletos. Un prototipo inicial se describe en [5]. Consiste en un sistema que mueve la muñeca y el codo del usuario mediante el uso de tres cuerdas de longitud variable conectadas a cada t, todas ellas ancladas a una placa colgante fijada gracias a un tubo.

Otra órtosis de miembro superior que no requiere un punto de anclaje fijo es ABLE [6], que mediante el uso de diferentes sistemas de transmisión en cada articulación permite alojar los motores de la articulación del hombro en un módulo que se puede acoplar a la espalda del usuario como una mochila, que, a pesar de aportar mucha autonomía a usuarios que no tienen dificultades para caminar o ponerse de pie, puede resultar incómoda a la hora de sentarse.

Ilustración 4: Prótesis ABLE

Algunos se utilizan para neurorrehabilitación, como ARMin, [7], [8] y Armeo Spring [9]. Están fijados sobre un marco sólido para liberar al usuario de su peso. Por este motivo su autonomía es muy baja. Aunque el campo de los miembros robóticos supernumerarios se encuentra en una etapa muy temprana, algunos ya han sido implementados.

Respecto a los miembros superiores, en [Parietti16] se describen extremidades artificiales que reemplazan/mejoran los brazos humanos para la perforación en la industria aeroespacial.

1.4. Antecedentes

Como ya se ha mencionado anteriormente, este trabajo abarca el proceso de implementación de mejoras a un prototipo ya existente, cuyos principios y decisiones de diseño se detallan en los trabajos de fin de master de José Mullet Arbelos [10] y Roland Rojas Moreno [11], en los que figuran consideraciones importantes en lo que al diseño del brazo robot se refiere, como son los requisitos de diseño, la elección de número de grados de libertad del brazo, el análisis cinemático del mismo o los procesos de diseño de las partes mecánicas.

A continuación se presenta un resumen abreviado de las más relevantes.

1.4.1. Requisitos

Las principales propiedades a tener en cuenta a la hora del diseño del brazo son las siguientes:

- Forma ergonómica: asegurar que el brazo robótico se ajuste cómodamente y de manera segura a la anatomía del usuario. Esto es crucial para garantizar que el usuario pueda mover el brazo robótico de manera natural y sin esfuerzo, lo que contribuye a su comodidad y capacidad de uso diario.
- Facilidad de fabricación: Esto garantiza que el brazo robótico sea asequible y accesible para una amplia gama de personas con

discapacidades. Si el diseño es complejo y costoso de producir, puede limitar su disponibilidad y accesibilidad.

- Masa reducida: Reducir la masa del brazo robótico es importante para evitar la fatiga del usuario y garantizar que pueda moverlo con facilidad. Una masa reducida también puede facilitar el transporte del dispositivo, lo que es especialmente relevante para usuarios que necesitan llevarlo consigo.
- Resistencia: La resistencia es esencial para garantizar que el brazo robótico pueda realizar tareas útiles de manera eficaz y sin riesgo de daños. Debe ser lo suficientemente robusto para realizar acciones como agarrar objetos, manipular objetos y realizar otras actividades cotidianas.
- Facilidad de mantenimiento: Un diseño que permita un acceso sencillo a componentes y una reparación eficiente puede mejorar la durabilidad y la vida útil del brazo.

1.4.2. Estudio cinemático

En su TFM, Mullet [10] realiza el cinemático de un brazo humano, comparándolo con el que se podría conseguir con un brazo robot con 5 grados de libertad, obteniendo así las medidas y el rango de movimiento de cada articulación, así como el área de trabajo aproximada de los mismos:

Tabla 1: Parámetros cinemáticos de brazo humano promedio

Rango hombro (°)	Longitud Brazo (mm)	Rango codo (°)	Longitud Antebrazo (mm)	Rango Antebrazo (°)	Rango Muñeca (°)	Longitud Mano (mm)	Longitud total brazo (mm)
259	300	145	260	14	285	200	760

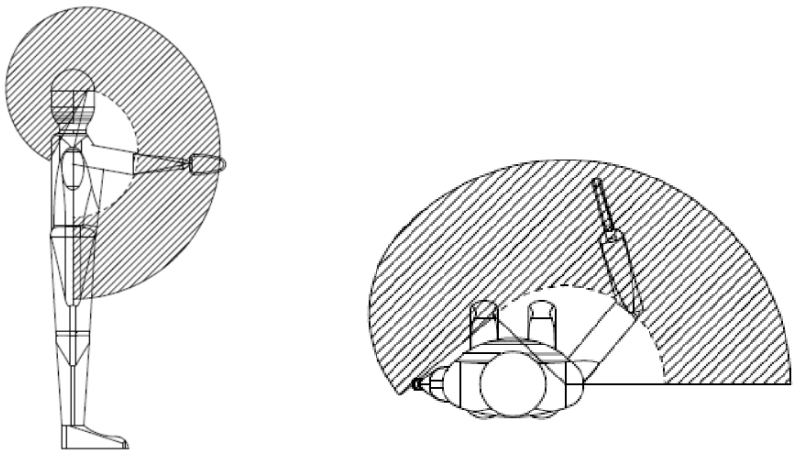


Ilustración 5: Volumen de trabajo de brazo humano promedio

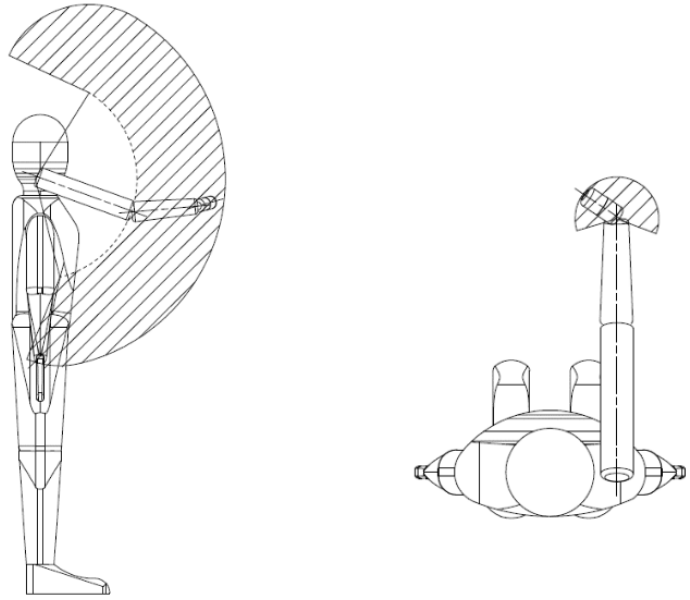


Ilustración 6: Volumen de trabajo de brazo de 5 grados de libertad

Como se puede observar, el brazo humano presenta un volumen de trabajo mucho mayor que el brazo de 5 grados de libertad, ya que este utiliza dos de ellos para posicionar la muñeca, y los restantes para orientar el efector final, de forma que no preseta ninguna forma de moverse hacia de lado a lado, lo cual el usuario puede solventar moviendo el tronco a la orientación deseada.

Así pues, el brazo dispone de 5 articulaciones rotacionales, siendo la primera el hombro, seguida del codo, y finalmente 3 articulaciones que forman una muñeca esférica.

Cabe mencionar además, que tanto la 2ª y 3ª como la 4ª y 5ª articulación están actuadas mediante un sistema de engranajes diferencial con dos entradas, lo que permite que los motores se coloquen lo más cerca de la base posible, de forma que se reduzca los efectos de la fuerza gravitatoria sobre el par que deban ejercer los motores de cada articulación.

1.4.3. Diseño generativo

El diseño generativo es un enfoque de diseño que utiliza algoritmos y software para generar automáticamente múltiples opciones de diseño basadas en parámetros específicos y objetivos definidos por el diseñador. En lugar de crear manualmente un diseño desde cero, el diseño generativo aprovecha la potencia de la computación y los algoritmos para explorar una amplia variedad de soluciones potenciales.

Como ya se ha mencionado, esta técnica está basada se basa en algoritmos y software especializados que pueden realizar cálculos y evaluaciones para generar diseños. Estos algoritmos pueden utilizar técnicas de optimización, simulación y aprendizaje automático para crear y refinar las opciones de diseño, y se guían mediante restricciones y parámetros específicos impuestos por el usuario, como pueden ser el tamaño, la forma, el rendimiento, el costo o cualquier otro criterio relevante para el proyecto.

Una vez que se establecen los parámetros y restricciones, el software de diseño generativo explora automáticamente una amplia gama de posibles soluciones. Esto puede resultar en diseños innovadores y eficientes que pueden no haber sido concebidos fácilmente de manera manual. Además, este proceso es iterativo. El diseñador puede revisar las opciones

generadas por el software y ajustar los parámetros y restricciones según sea necesario para refinar el diseño final.

El diseño generativo a menudo se utiliza para optimizar un diseño en función de ciertos criterios, como minimizar el peso, maximizar la resistencia o mejorar la eficiencia energética. Los algoritmos ajustan continuamente el diseño para alcanzar los objetivos definidos.

1.5. Material utilizado

1.5.1. Software

1.5.1.1. MATLAB

MATLAB (abreviatura de "Matrix Laboratory") es un lenguaje de programación y un entorno de desarrollo utilizado principalmente en ingeniería, matemáticas y ciencias relacionadas. Desarrollado por Mathworks y se utiliza para realizar una variedad de tareas relacionadas con el análisis numérico, la modelización matemática, la simulación y la visualización de datos.

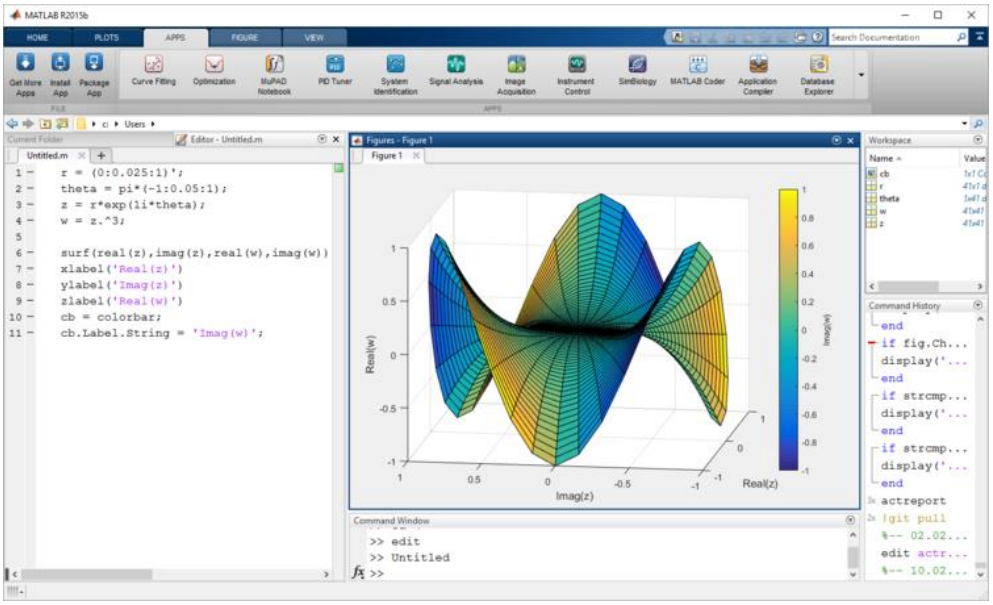


Ilustración 7: Entorno gráfico de MATLAB

En lo que concierne a este trabajo, se ha usado principalmente para desarrollar las funciones cinemáticas y dinámicas necesarias para el control del robot.

1.5.1.2. Fusion360

Fusion 360 es una aplicación de diseño asistido por computadora (CAD, por sus siglas en inglés) y de diseño de productos desarrollada por Autodesk diseñada para crear, modelar, simular y fabricar productos en 3D. Fusion 360 se ha convertido en una de las herramientas de diseño digital más popular en campos como la ingeniería mecánica, la arquitectura y el diseño de productos debido a sus capacidades integrales como la integración de simulaciones de esfuerzos y fluidos o funciones de diseño generativo y su enfoque en la colaboración y la nube.

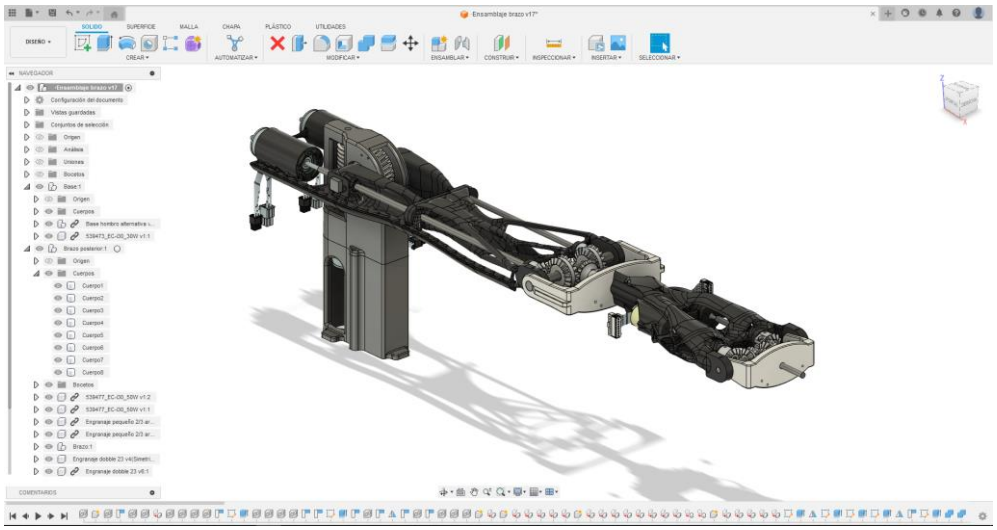


Ilustración 8: Interfaz Fusion360

En el ámbito de este trabajo, se ha utilizado MATLAB principalmente para el diseño tridimensional de piezas y por sus capacidades de diseño generativo.

1.5.1.3. Arduino IDE

El IDE de Arduino (Entorno de Desarrollo Integrado) de Arduinoes un software que se utiliza para programar y cargar código en placas de

desarrollo y microcontroladores Arduino (y compatibles). Arduino es una plataforma de hardware de código abierto ampliamente utilizada en proyectos de electrónica y robótica, y su IDE es una herramienta esencial para desarrolladores y entusiastas que trabajan con estos dispositivos, permitiendo la escritura, compilación y carga de programas de código a tarjetas microcontroladoras compatibles.

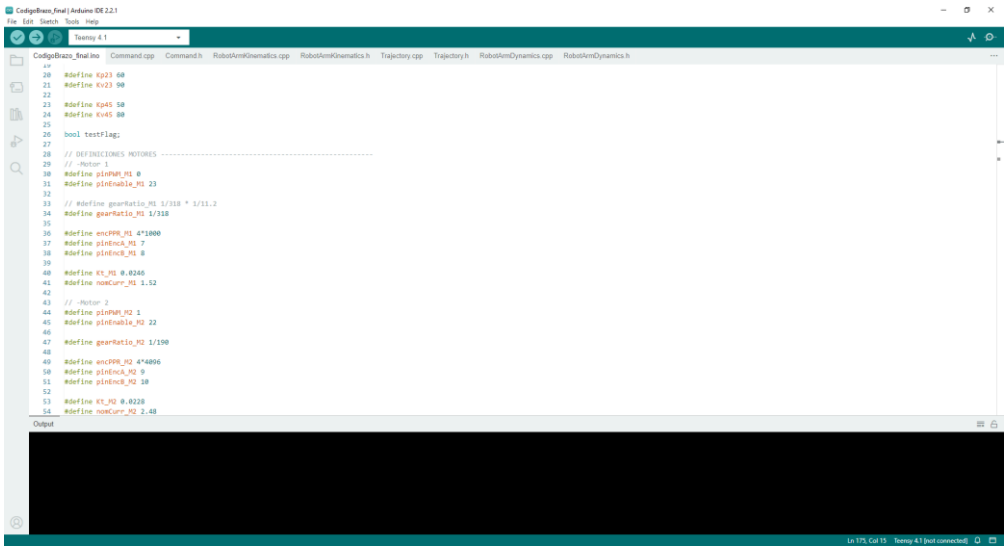


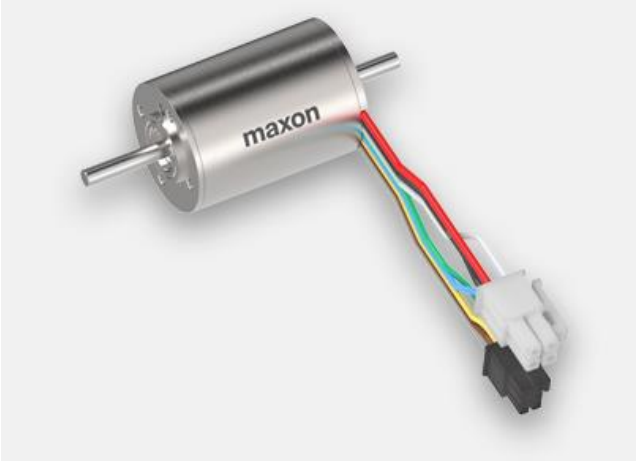
Ilustración 9: Interfaz de entorno de desarrollo de Arduino

1.5.2. Hardware

1.5.2.1. Motores

- 1ª articulación

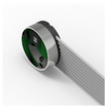
Para mover la primera articulación se utiliza un conjunto motor/reductor con encoder, con número de combinación 675639, compuesto por un motor EC-i 30 30W, un reductor planetario GP 32 C, con una relación de transmisión de 318:1, y un encoder ENC 16 EASY, que da una señal de salida de 1000 pulsos por vuelta del motor.



GEAR 	**Planetary Gearhead GP 32 C Ø32 mm, 1.0 - 6.0 Nm, Ceramic Version** Part number: 166951 **Price per unit:**	**€272.31**
MOTOR 	**EC-i 30 Ø30 mm, brushless, 30 W, with Hall sensors** Part number: 539473 **Price per unit:**	**€218.20**
SENSOR 	**Encoder ENC 16 EASY, 1000pulses** Part number: 499360 **Price per unit:**	**€104.10**

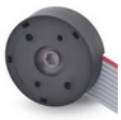
- ## 2ª y 3ª articulación

Para mover la segunda y tercera articulación se utiliza dos conjuntos motor/reductor con encoder, con número de combinación 715581, compuestos por un motor EC-i 30 50W, un reductor planetario GP 32 C, con una relación de transmisión de 190:1, y un encoder ENC 16 RIO, con una salida de 4096 pulsos por vuelta del motor.

GEAR 	**Planetary Gearhead GP 32 C Ø32 mm, 1.0 - 6.0 Nm, Ceramic Version** Part number: 166948 **Price per unit:**	**€250.83**
MOTOR 	**EC-i 30 Ø30 mm, brushless, 50 W, with Hall sensors** Part number: 539477 **Price per unit:**	**€250.52**
SENSOR 	**Encoder ENC 16 RIO, 4096 counts per turn, 3-channel, with RS 422 line driver** Part number: 575827 **Price per unit:**	**€212.31**

- ## 4ª y 5ª articulación

Para mover la segunda y tercera articulación se utiliza dos conjuntos motor/reductor con encoder, con número de combinación 675639, compuesto por un motor EC-max 22 26W, un reductor planetario GP 22 HP, con una relación de transmisión de 104:1, y un encoder MR Type ML, de 512 pulsos por vuelta del motor.

GEAR 	**Planetary Gearhead GP 22 HP Ø22 mm, 2.0 - 3.4 Nm, High Power** Part number: 370783	
	Price per unit:	**€268.10**
MOTOR 	**EC-max 22 Ø22 mm, brushless, 25 Watt, with Hall sensors** Part number: 283858	
	Price per unit:	**€257.15**
SENSOR 	**Encoder MR, Type ML, 512 CPT, 3 Channels, with Line Driver** Part number: 201940	
	Price per unit:	**€128.42**

1.5.3. Etapas de potencia

Todos los motores son alimentados por etapas de potencia en modo de control de corriente, que actúa sobre el par que ejercen en función de la constante de par del mismo motor, que establece una relación proporcional entre la corriente consumida y el par ejercido por el mismo.

1.5.3.1. 1ª, 2ª y 3ª articulación

Para alimentar y controlar los actuadores de las treas primeras articulaciones del robot, se utilizan etapas de potencia ESCON module 50/5

Es aparente a primera vista que el único tipo de conexión de la que dispone esta etapa (aparte del puerto microUSB mediante el que configura) de potencia son tres filas de pines estándar, de modo que para conectarlas a los motores o a la placa microcontroladora será necesaria una placa de circuito o conectores especiales.

1.5.3.2. 4ª y 5ª articulación

Los motores de estas articulaciones se controlan mediante etapas de potencia ESCON 36/6.

A diferencia de las anteriores, estas etapas de potencia sí que cuentan con una carcasa de protección y conectores que facilitan el conexionado entre la ESCON y los motores y controladora

1.5.3.3. Configuración y control

Para configurar las ESCON al tipo de control y motor al que estén conectadas, se conectan a un PC mediante el puerto micro USB del que disponen; posteriormente, será posible cambiar la configuración de la misma mediante el programa ESCON Studio. Para ello, la ESCON deberá estar completamente conectada al motor que se pretenda que controle, tanto los devanados del motor como los sensores de efecto Hall del mismo.

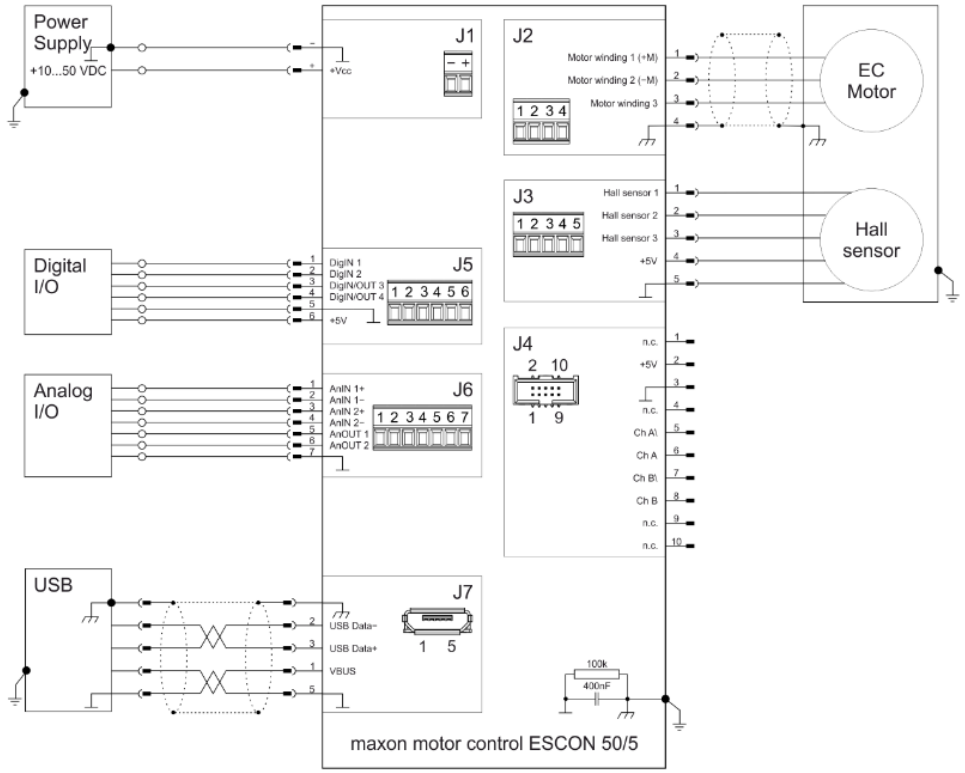
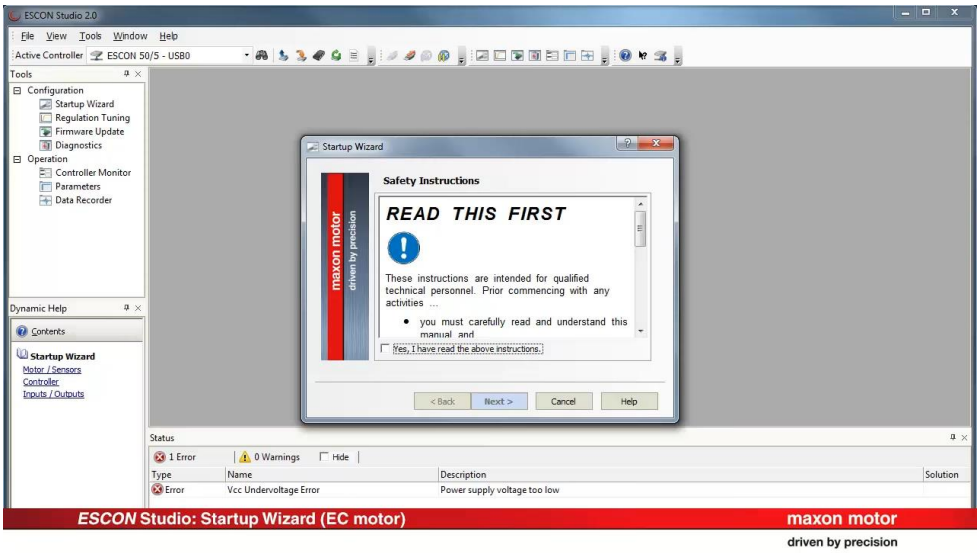
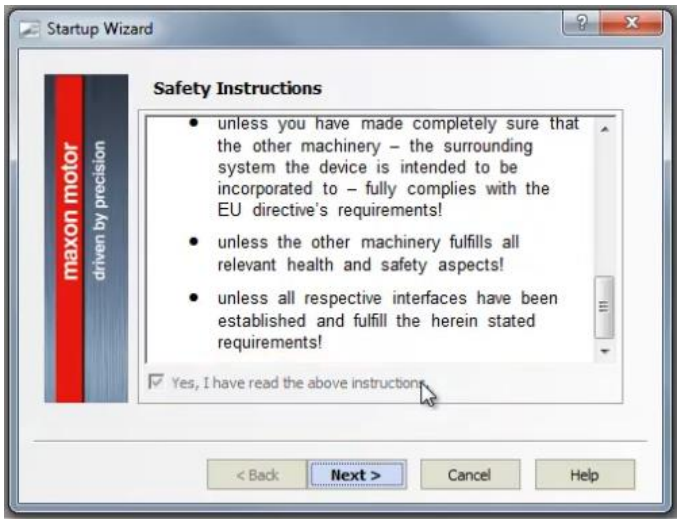


Ilustración 10: Esquema de conexionado de una ESCON 50/5 con un motor EC

Así pues, el proceso de configuración de una ESCON es relativamente sencillo, al disponer el programa ESON studio de un asistente de arranque en el que se introducen los datos del motor utilizado, el tipo de control que se quiera aplicar sobre el motor (en nuestro caso control de corriente), y la configuración de las entradas y salidas de la ESCON.

Es gracias a estas entradas y salidas por lo que nos será posible comunicar las etapas de potencia con el microcontrolador, usando, por ejemplo, una entrada digital para habilitar o deshabilitar la alimentación y conmutación del motor, o una entrada analógica para controlar la corriente suministrada al motor.





La anteriormente mencionada consigna analógica se consigue realizando un filtro paso bajo sobre señales PWM (modulación de ancho de pulso). Este filtro es implementado mediante circuitos RC simples, con resistencias de 10 kΩ y 1 µF, lo cual da una frecuencia de corte de 15 Hz, lo cual filtra las componentes de alta frecuencia de

la señal PWM permitiendo a su vez que la señal filtrada tenga un tiempo de respuesta aceptable.

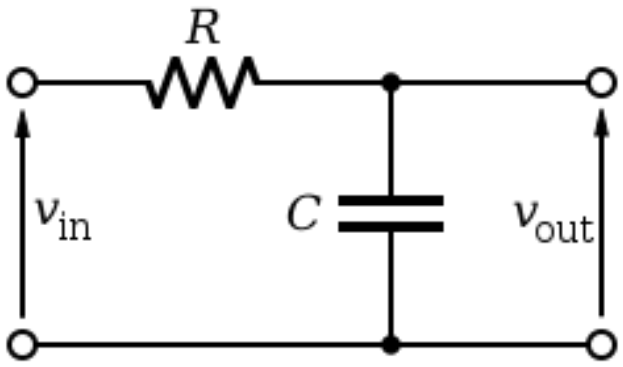


Ilustración 11: Esquema de circuito RC de paso bajo

1.5.4. Teensy 4.1

Se conoce como Teensy 4.1 a un modelo placas de desarrollo de microcontroladores diseñadas y fabricadas por PJRC (Paul Stoffregen's Teensy LLC). Estas placas se utilizan comúnmente en proyectos de electrónica y programación, y son conocidas por su tamaño compacto, potencia y versatilidad. A lo largo del tiempo, se han lanzado varias versiones de Teensy, cada una con diferentes características y capacidades. La placa cuenta con un procesador ARM Cortex-M7 con una frecuencia de 600 MHz; una unidad lógica para la realización de operaciones de punto flotante; gran cantidad de memoria RAM y FLASH, con buses de alta velocidad; y 55 pines de entrada y salida, todos con funcionalidad de interrupciones, y de los cuales 35 pueden dar una señal de tipo PWM.

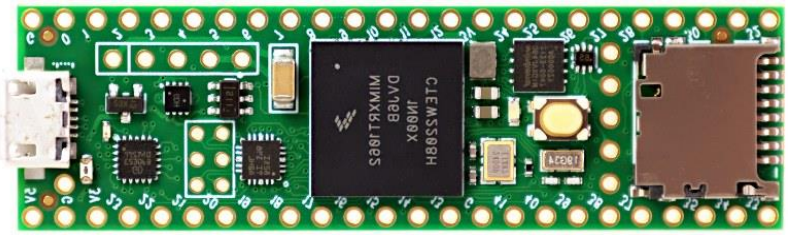


Ilustración 12: Teensy 4.1

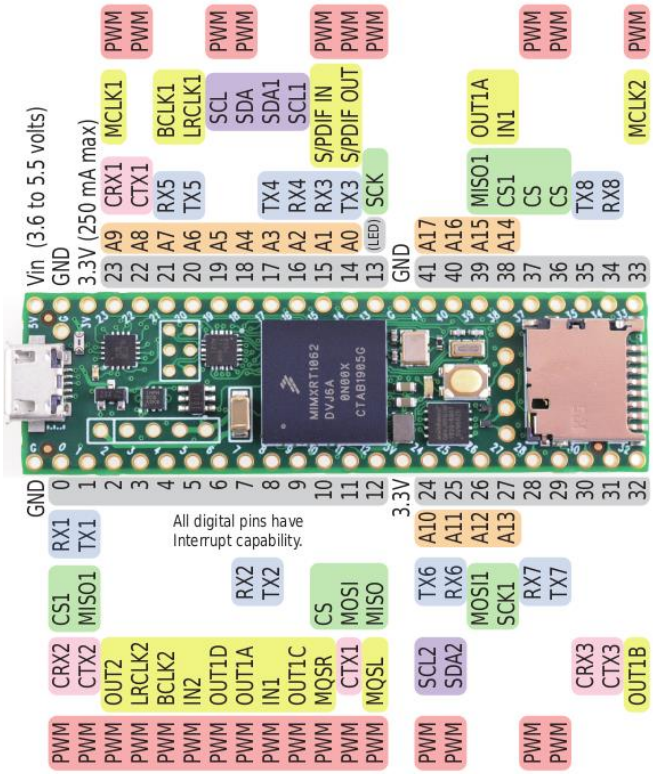


Ilustración 13: Pines de la Teensy 4.1

Para cargar el programa de control el robot en la placa, se usa el IDE de Arduino junto con la extensión Teensyduino, que permite dicha carga, conectado a través del puerto miniUSB, que también se usará para establecer comunicación serie con la placa.

2. Conexionado y electrónica

2.1. Conceptos previos

Como ya se ha mencionado, para mover el brazo robot se usa como actuadores 5 motores de corriente continua sin escobillas, todos de la marca Maxon, alimentados mediante etapas de potencia conocidos como ESCON, también de Maxon, que permiten controlar su corriente y velocidad.

A continuación se exponen brevemente los componentes y características de los motores utilizados en cada articulación y de las etapas de potencia.

2.2. Finales de carrera

Una modificación destacable realizada sobre el brazo es la adición de finales de carrera físicos, mediante los cuales es posible evitar colisiones entre articulaciones, haciendo que se paren los motores que los accionan antes de que una de dichas articulaciones salga de su rango de movimiento admisible.

Además, estos finales de carrera permiten que el robot realice una rutina de "homing", o retorno a cero, lo que permite que el robot conozca la posición de cada una de sus articulaciones sin importar la pose en la que se encontraba al inicializarse, lo cual no sería posible sin los finales de carrera, al no disponer las articulaciones de sensores de posición absolutos, siendo la única información disponible la obtenida de los encoders relativos de los motores.

Los finales de carrera escogidos para este propósito son los Microinterruptor, Palanca de Rodillo Simulado SP-CO.

Ilustración 14: Sensor de final de carrera SP-CO

Resulta obvio que se necesitan como mínimo 10 sensores de fin de carrera, puesto que cada uno de los 5 ejes de articulación del robot necesita dos de ellos, uno para el límite positivo de su rango de movimiento, y otro para el límite negativo. Una práctica común en el ámbito industrial, realizada para operar con mayor seguridad, es añadir un sensor extra a cada eje de

articulación para la rutina de homing, aunque también es posible utilizar uno de los sensores de límite ya mencionados.

15 sensores de final de carrera requeriría del uso de 15 entradas digitales del microcontrolador si se conectasen de forma directa, se ha abogado por realizar una conexión similar a la vista en los teclados de ordenador, denominada matriz de teclado, que reduce el número de contactos directos desde 15 hasta 8, de las cuales 5 de ellas son entradas digitales, una para cada eje, y 3 de ellas, salidas digitales, una para cada tipo de sensor de final de carrera (límite positivo, límite negativo y homing).

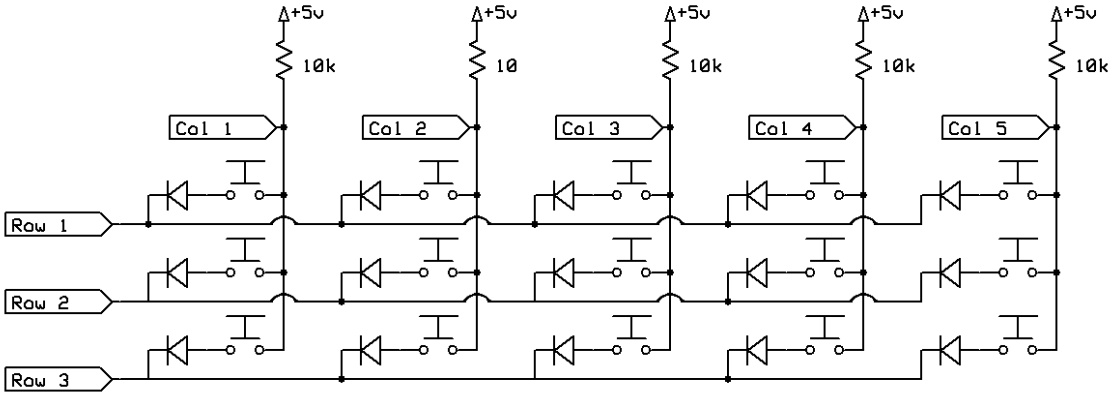


Ilustración 15: Circuito de matriz de teclado

En la figura anterior (Ilustración 15), las filas y columnas representan las salidas y entradas digitales del microcontrolador, respectivamente, utilizadas para la conexión de los finales de carrera. En principio, las salidas del microcontrolador serán iguales a tierra durante la operación del robot, a no ser que se quiera conocer qué finales de carrera están activados y desactivados, en cuyo caso se sondearán las entradas (que incidentemente, en nuestro caso se trata de entradas de tipo pull-up) haciendo que solo una de las salidas digitales esté a tensión nula. La matriz también consta de diodos que aseguran que no haya un falso positivo durante dicho sondeo.

El único inconveniente que presenta este sistema de conexión frente a una conexión directa es que una vez se active uno de los finales de carrera no es posible averiguar de cual se trata de forma directa, y habrá que realizar un sondeo de todos los sensores para averiguarlo. Sin embargo, esto no resulta tan importante como para los motores una vez se detecte que una articulación haya salido fuera de rango, de forma que será posible realizar dicho sondeo una vez estén todos los motores parados.

2.3. Placas de circuitos

Para reducir el espacio ocupado por los componentes electrónicos, facilitar su transporte y aumentar la seguridad de la conexión, se han soldado gran parte de los componentes a placas de circuito junto con puertos terminales y zócalos, que facilitan la conexión de elementos externos, como son los motores, las ESCON, los encoders o la placa microcontroladora.

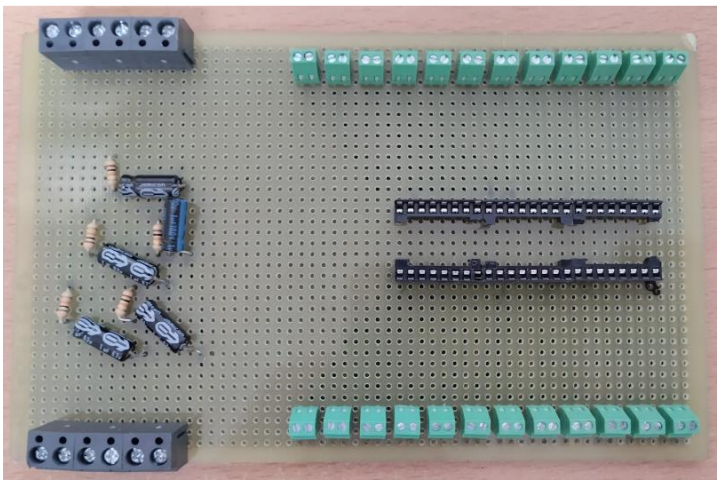


Ilustración 16: Placa de conexión de la Teensy y filtros paso-bajo, lado superior

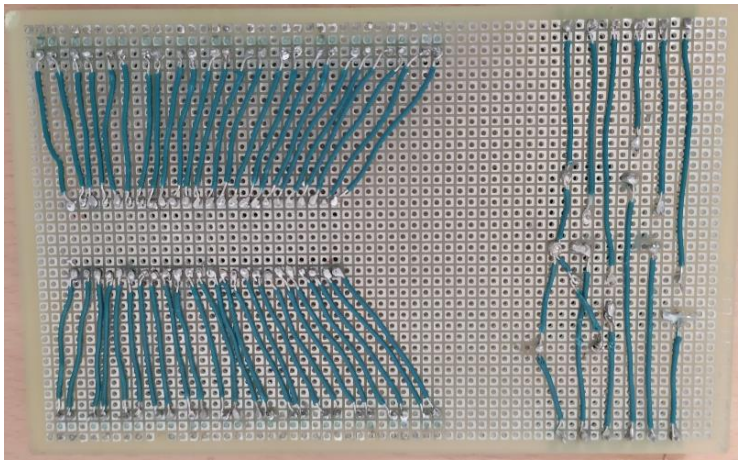


Ilustración 17: Placa de conexión de la Teensy y filtros paso-bajo, lado inferior

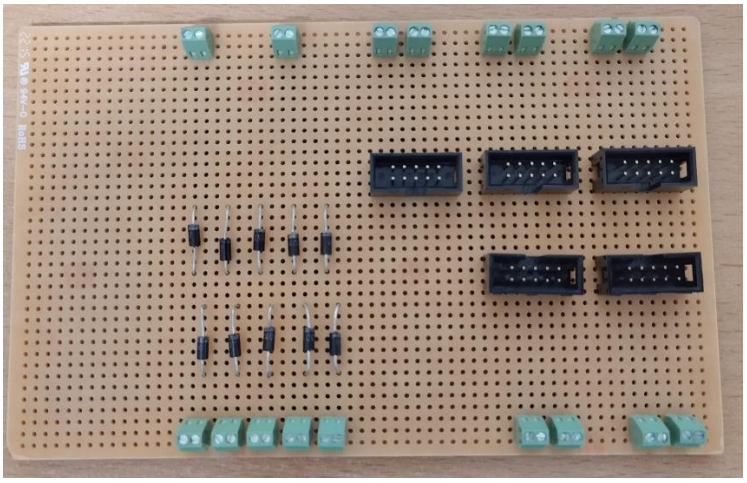


Ilustración 18: Placa de conexión de los encoders y diodos de matriz de teclado, lado superior

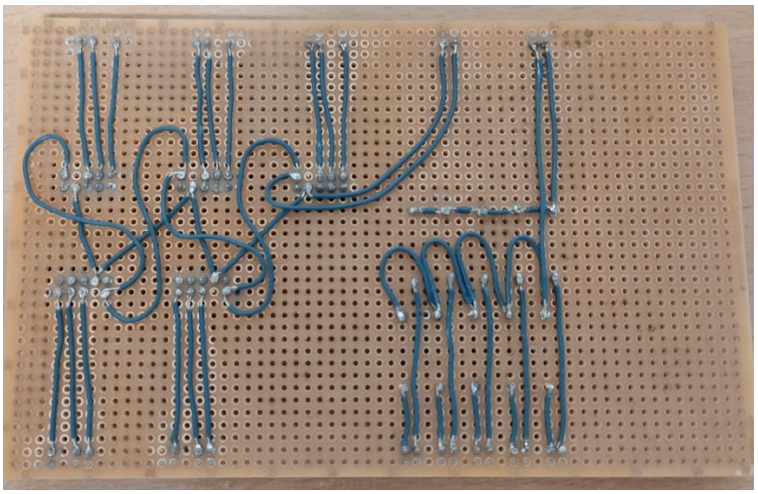


Ilustración 19: Placa de conexión de los encoders y diodos de matriz de teclado, ladoinferior

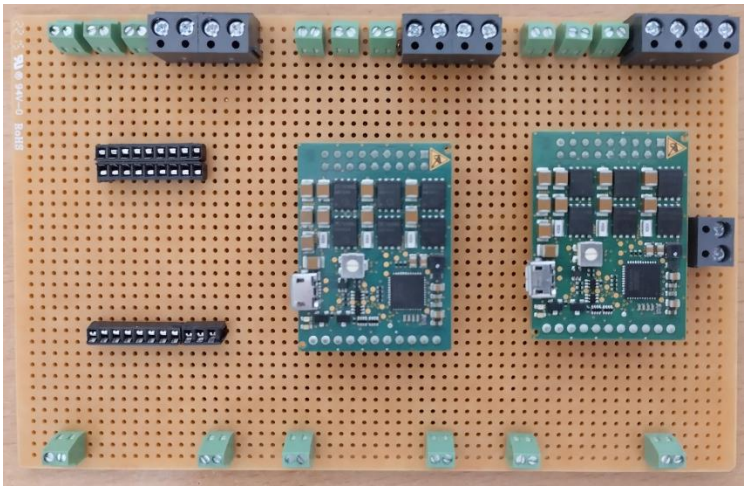


Ilustración 20: Placa de conexión de etapas de potencia de motores, lado inferior

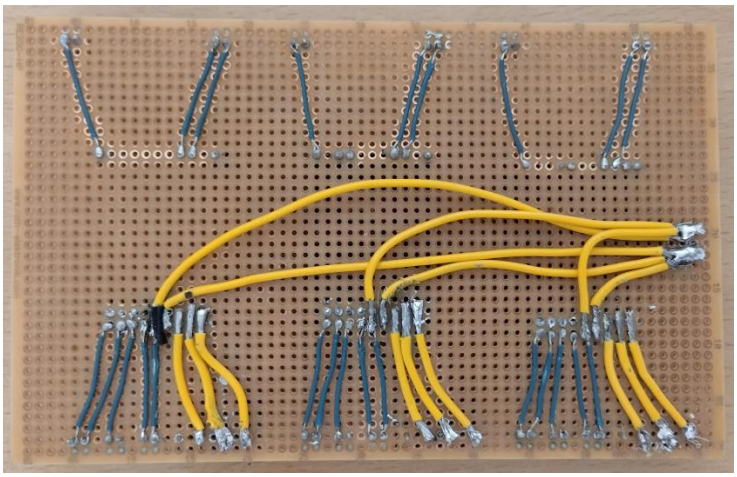


Ilustración 21: Placa de conexión de etapas de potencia de motores, lado inferior

2.4. Cableado

También se ha mejorado el cableado del sistema, sobre todo en lo referente a los motores de la 1ª, 2ª y 3ª articulación, añadiendo extensiones al cableado del devanado de los motores mediante cables de 20 AVG; a la conexiones de los encoders mediante cable plano para conectores IDC; y para los sensores de efecto Hall de los motores mediante cable de 28 AWG, que también se ha utilizado para el conexionado de los finales de carrera.

3. Diseño mecánico

3.1. Conceptos previos

En lo que respecta al diseño mecánico del brazo en sí, hay algunos conceptos pertinentes al que conviene tener claros:

3.1.1. Mecanismo primera articulación

El movimiento y transmisión de potencia de la primera articulación del brazo, que podemos considerar el 'hombro' del mismo, funciona mediante un mecanismo de tornillo sin fin, moviendo el motor el tonillo dentado, de forma que la corona esté acoplada al eje de la articulación del hombro. El procedimiento de diseño de los engranajes usados en este mecanismo (y en los vistos en el apartado siguiente) se describe detalladamente en el trabajo de master de Roland [11].

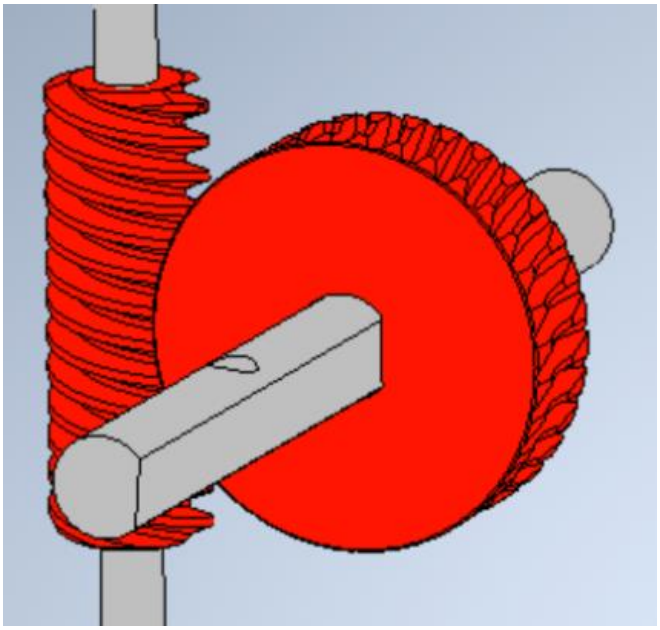


Ilustración 22: Tornillo sin fin utilizado en la primera articulasción del brazo

3.1.2. Sistema de transmisión diferencial

Tal y como ya se ha mencionado en el apartado 1.4.2, en el diseño del brazo se usa un tipo de sistema de engranajes con dos entradas para accionar dos conjuntos de articulaciones distintos (conformados por la 2ª y la 3ª; y la 4ª y la 5ª articulación, respectivamente), de forma que se permita la posibilidad de instalar los motores de dichas articulaciones lo más cerca de la base del robot como sea posible (o en el caso de los que accionan la 2ª y 3ª articulación, detrás de la misma), para contrarrestar los efectos adversos de las fuerzas gravitatorias que actúen sobre el robot sobre el comportamiento del mismo.

Éste sistema de engrane está formado por dos piñones de entrada, conectados a los motores mediante ejes de acero y acopladores flexibles, cada uno de los cuales está en contacto con una corona intermedia. El ángulo de giro de la primera articulación del conjunto se tratará del promedio de los ángulos de giro de ambas coronas. A su vez, estas coronas intermedias están en contacto a su vez con un piñón de salida, cuyo ángulo de giro se corresponderá con el de la 2ª articulación del conjunto

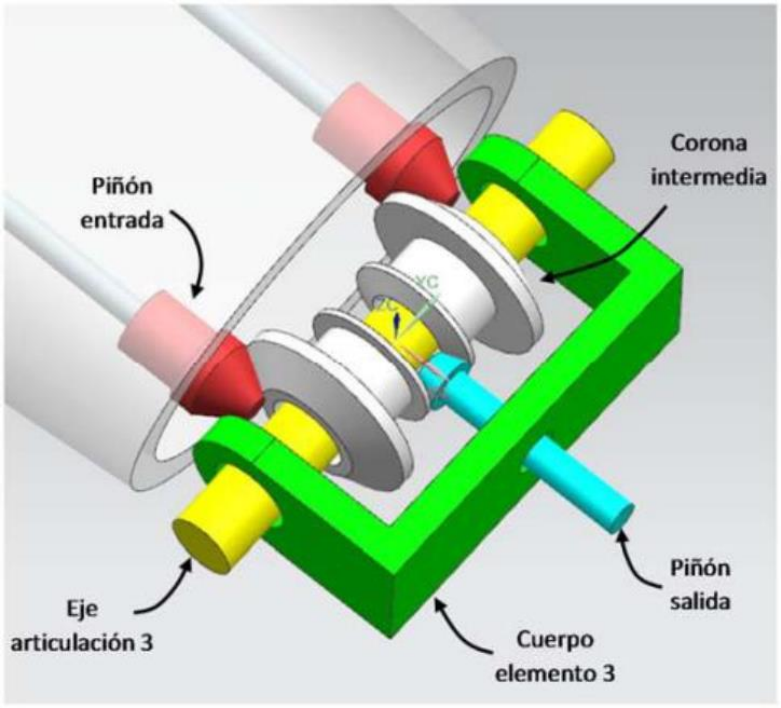


Ilustración 23: Diagrama sistema de engrane diferencial

Así pues, el accionamiento de la primera articulación del sistema, que realizaría un movimiento de flexión, se producirá cuando los piñones de entrada giren de la misma forma en sentidos opuestos, mientras que la segunda articulación, que realizará un movimiento de alabeo, se producirá cuando los piñones de entrada giren en el mismo sentido.

Así pues, el movimiento de las articulaciones que comprenden el sistema se podrá expresar mediante las siguientes ecuaciones:

$$\begin{cases} \theta_{flexion} = \dfrac{n_{corona_{outer}}}{n_{piñon_{in}}} \dfrac{(\theta_{in_1} - \theta_{in_2})}{2} \\ \theta_{alabeo} = \dfrac{n_{corona_{outer}}}{n_{piñon_{in}}} \dfrac{n_{piñon_{out}}}{n_{corona_{inner}}} \dfrac{(\theta_{in_1} + \theta_{in_2})}{2} \end{cases}$$

$$\begin{cases} \theta_{flexion} = \dfrac{n_{corona_{outer}}}{n_{piñon_{in}}} \dfrac{(\theta_{in_1} - \theta_{in_2})}{2} \\ \theta_{alabeo} = \dfrac{n_{corona_{outer}}}{n_{piñon_{in}}} \dfrac{n_{piñon_{out}}}{n_{corona_{inner}}} \dfrac{(\theta_{in_1} + \theta_{in_2})}{2} \end{cases}$$

Siendo θ y ω el ángulo y velocidad de giro del engranaje correspondiente, y n el número de dientes del mismo. A su vez, el par transmitido desde los piñones de entrada responderá a la siguiente exuación:

$$\begin{cases} \tau_{flexion} = \dfrac{n_{piñon_{in}}}{n_{corona_{outer}}}(\tau_{in_1} - \tau_{in_2}) \\ \tau_{alabeo} = \dfrac{n_{piñon_{in}}}{n_{corona_{outer}}}\dfrac{n_{corona_{inner}}}{n_{piñon_{out}}}(\tau_{in_1} + \tau_{in_2}) \end{cases}$$

Resulta obvia la posibilidad de invertir estas ecuaciones para calcular el par motor que se debería aplicar sobre los piñones de entrada para obtener un par determinado a la salida, lo cual resultará útil a la hora de controlar el brazo.

3.1.3. Impresión 3D

La impresión 3D, también conocida como fabricación aditiva, es un proceso de fabricación que crea objetos tridimensionales sólidos a partir de un modelo digital utilizando capas sucesivas de material. A diferencia de los métodos de fabricación tradicionales, que suelen ser sustractivos (donde se quita material de una pieza sólida), la impresión 3D es un proceso aditivo en el que se agrega material capa por capa para construir el objeto deseado. Este proceso consta de varias partes, como son el modelado u obtención de modelos de las piezas que se pretenda imprimir; el procesado de las mismas en un tipo programa de software especializado conocido como slicer, que traduce la forma de la pieza a un conjunto de órdenes de movimiento y control de la impresora, pudiendo modificar también parámetros como el relleno del interior de la pieza, la velocidad de movimiento de la impresión o los soportes que necesite la pieza para ser impresa (si es que acaso los necesita); posteriormente se imprime la pieza mediante el proceso que se haya elegido (FDM, SLA,

SLS…), y finalmente se puede posprocesar la pieza, para, por ejemplo, lograr que tenga una mayor exactitud dimensional o suavizar la superficie para eliminar los artefactos superficiales surgidos de la naturaleza de este tipo de fabricación capa a capa.

Las piezas del brazo impresas en 3D se han manufacturado usando dos técnicas distintas, que se expondrán a continuación.

3.1.3.1. Impresión 3D FDM

La impresión 3D FDM (Fused Deposition Modelling), o de Modelado por Deposición Fundida, es la forma más extendida de fabricación aditiva, consistente en derretir un filamento de material termoplástico y depositarlo capa por capa en el área de impresión hasta conseguir la pieza deseada.

Ilustración 24: Ejemplo genérico de impresora FDM

Comparada con otras técnicas de fabricación aditiva, es la más barata y la que cuenta con menor resolución, presentando también problemas

en cuanto a la resistencia mecánica de las piezas en relación a la orientación en las que han sido impresas, o la impresión de salientes y/o voladizos presentes en las mismas, requiriendo de material extra de soporte para su impresión correcta.

Sin embrago, cuenta con un abanico muy amplio de posibles materiales de impresión (PLA, PETG, ABS, TPU, Nylon…) con propiedades y usos distintos, y, dependiendo del diseño de la pieza a imprimir, puede dar unos resultados excelentes para su precio.

3.1.3.2. Impresión 3D SLS

La impresión 3D de Sinterizado Selectivo por Laser, o SLS es un proceso de fabricación aditiva que utiliza un láser para fusionar partículas de polvo de material termoplástico, cerámico o metálico para crear objetos tridimensionales. Este método de impresión 3D es conocido por su capacidad para producir piezas de alta calidad con detalles finos y una amplia gama de materiales.

Ilustración 25: Impresora SLS Formlabs Fuse 1

Este proceso funciona depositando una fina capa de material (en nuestro caso Nylon) en forma de polvo, y calentándola con un láser de alta potencia a una temperatura ligeramente mayor que la temperatura de fusión del material, fusionando las partículas del material en los lugares donde se necesita crear la pieza. Posteriormente, se procede a depositar otra fina capa de material encima de la anterior, y se vuelve a repetir todo el proceso; esto continúa hasta que se haya formado toda la pieza. Posteriormente, se deja enfriar la pieza en el lecho del polvo, ayudando a mantener la estructura y rigidez de la misma, y por último, se retira la pieza del lecho de polvo y se somete a un proceso de limpieza, para eliminar excesos de material no fundido.

Ilustración 26: Ejemplo de piezas manufacturadas mediante un proceso SLS

La impresión 3D SLS es apreciada por su capacidad para producir piezas funcionales y complejas con una amplia variedad de materiales, lo que la hace adecuada para aplicaciones en la industria aeroespacial, automotriz, médica y muchas otras. Además, dado que no se requieren estructuras de soporte como en otros métodos de impresión, es posible imprimir objetos con geometrías intrincadas y sin necesidad de eliminar soportes después de la impresión. Sin embargo, tanto la maquinaria como el material necesarios resultan más costosos que los utilizados en impresión FDM, y el polvo utilizado para una tirada de impresión que no se haya fundido no puede ser reutilizado tal cual, sino que habrá que mezclarlo con polvo no utilizado, de forma que no se puede decir que realmente se trate de un método de fabricación aditiva que no desperdicie material.

3.1.4. Diseño generativo

Como ya se ha expuesto en el apartado de introducción, el diseño generativo se trata de una herramienta extremada útil a la hora de diseñar piezas tan resistentes y ligeras como sea posible, lo cual obviamente resulta ventajoso en lo que respecta a este proyecto, asegurando que el

diseño mecánico del brazo cumpla los requisitos establecidos de la forma más holgada posible.

Así pues, para llevar a cabo el proceso de diseño generativo, se ha utilizado el software Fusion360 de Autodesk, que incorpora funciones y características que permite realizar este tipo de estudio.

A continuación, se tocará brevemente el proceso de diseño de una pieza, mediante una pieza de ejemplo dada por Autodesk. [12]

Antes de nada, habrá que entrar en el apartado de programa de diseño generativo y, o bien cargar o elaborar un diseño geométrico a partir del que se obtendrá el resultado final. Para ello, habrá que indicar los elementos geométricos que se desee preservar, o, opuestamente, actuar como geometría de obstáculo, haciendo que no haya material en el diseño generado por el proceso.

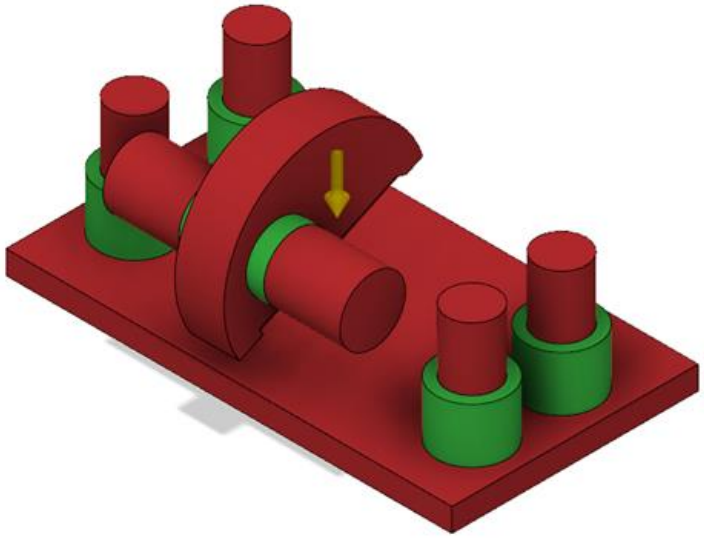


Ilustración 27: Geometría a preservar (en verde) y de obstáculo (en rojo) de para un ejemplo de diseño generativo

A continuación se procede a establecer las restricciones estructurales (como por ejemplo, de movimiento en un eje de coordenadas) y establecer diversos casos de carga con diferentes fuerzas o presiones que actúen sobre un elemento geométrico de la pieza, a los que se prevé que se vaya a ser sometida la pieza a lo largo de su vida útil.

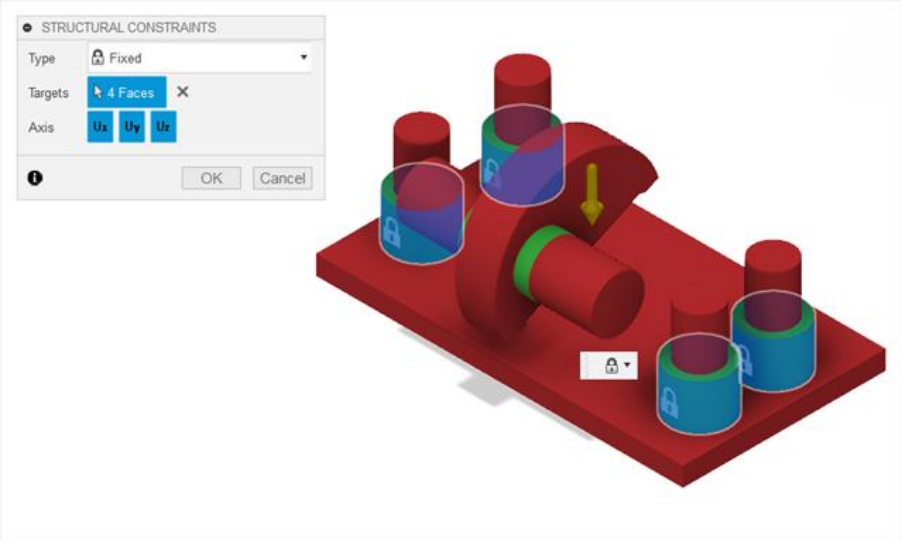


Ilustración 28: Ejemplo de restricciones estructurales de diseño generativo

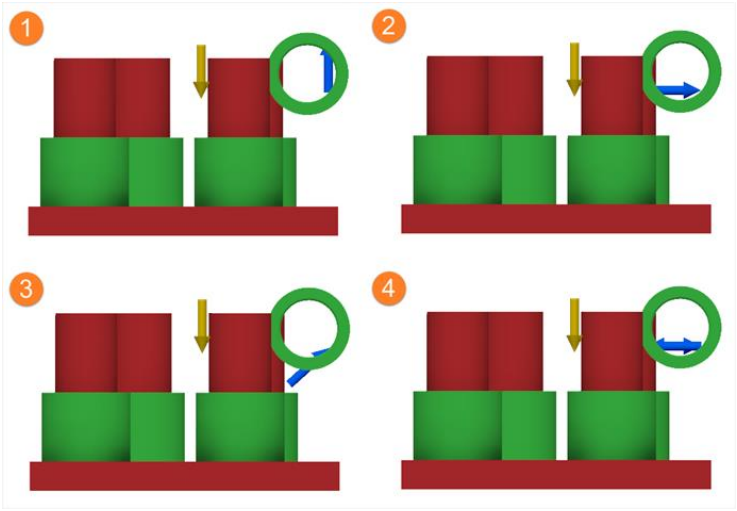


Ilustración 29: Ejemplo de casos de carga de diseño generativo

Posteriormente, se procede a especificar el método de fabricación de la pieza (fabricación aditiva, fresado, troquelado, etc), el material de la pieza, y los objetivos (minimizar masa o maximizar rigidez) y límites (coeficiente de seguridad mínimo de la pieza) de diseño; tras lo cual se puede realizar uno o más estudios generativos.

Una vez procesados computacionalmente, dichos estudios se pueden explorar, ordenar y filtrar en base a los parámetros que se consideren oportunos, pudiendo exportar los resultados de cada una de las iteraciones de cada estudio.

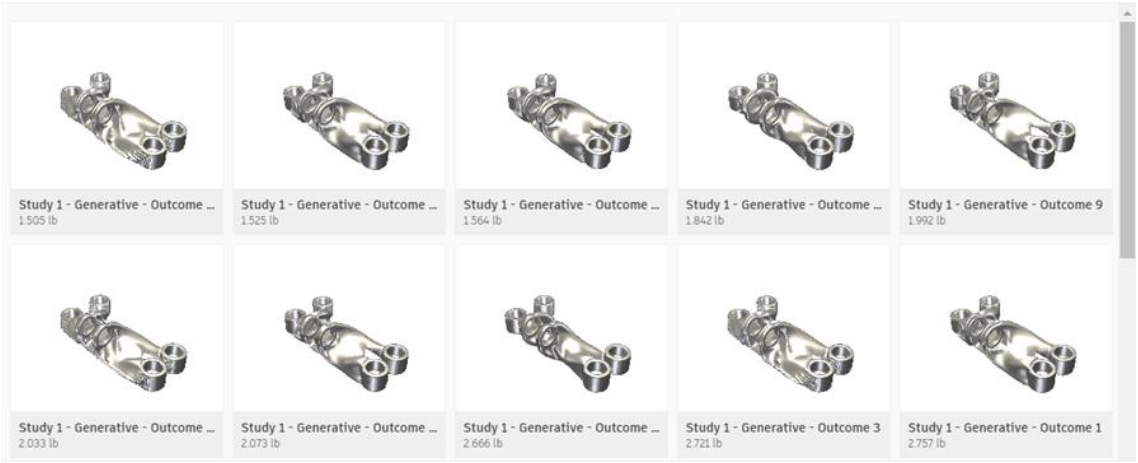


Ilustración 30: Ventana de exploración de estudios de diseño generativo

Ilustración 31: Ejemplo de resultado final de diseño generativo

3.2. Mejoras y nuevas aportaciones

3.2.1. Eje hombro y rodamientos lineales

Como se puede ver en la Ilustración 22, en la versión anterior del brazo, el eje de la primera articulación, fijada a la corona del tornillo sin fin, consistía en una barra redonda de aluminio, sometido a una operación de lamado para proporcionar agarre entre el engranaje y el eje.

Ilustración 32: Eje del hombro de diseño anterior del brazo robot

Sin embargo, este perfil no ha resultado ser muy adecuado, provocando un acoplamiento mecánico subóptimo entre el eje y el engranaje, y bastante desgaste en este último. Por tanto, se ha decidido sustituir este eje por un perfil cuadrado, añadiendo además rodamientos entre este y la base del brazo, montados mediante adaptadores fabricados mediante impresión 3D para facilitar el movimiento del mismo.

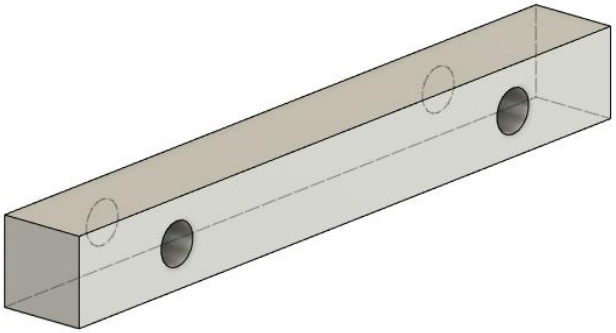


Ilustración 33: Nuevo diseño del eje de la primera articulación

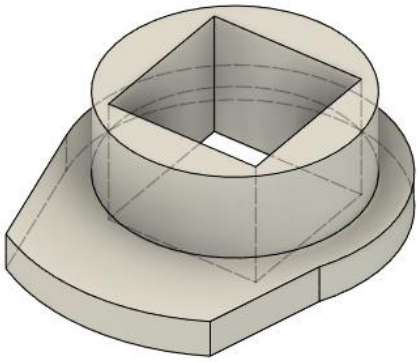


Ilustración 34: Adaptador eje/rodamiento

Ilustración 35: Rodamiento de bolas 6804-2RS

Otro elemento de diseño de la versión anterior del brazo que resultó en una reducción importante del rendimiento, sobre todo en cuanto al sistema comprendido por la 2ª y la 3ª articulación, es el hecho de que los ejes acoplados a los motores de dichas articulaciones fueran el único elemento mecánico del brazo que transmitía los esfuerzos provocados por las fuerzas gravitatorias en el robot al eje del hombro, produciendo unos esfuerzos de rozamiento considerables, lo cual se traduce en un mayor esfuerzo mecánico requerido por los motores de las articulaciones afectadas y, por tanto, un mayor consumo de energía.

Ilustración 36: Junta cillindrica formada por eje de la primera articulación y eje de transmisión de potencia de 2ª/3ª articulación

El nuevo perfil cuadrado del eje de la primera articulación ayuda a solventar buena parte de este problema, haciendo que los esfuerzos gravitatorios se vean transmitidos por las piezas del brazo posterior impresas mediante SLS; además se han añadido rodamientos lineales a caga lado del eje de la primera articulación para ambos ejes de motores, para prácticamente eliminar los restantes esfuerzos de rozamiento entre ejes que pudiera haber.

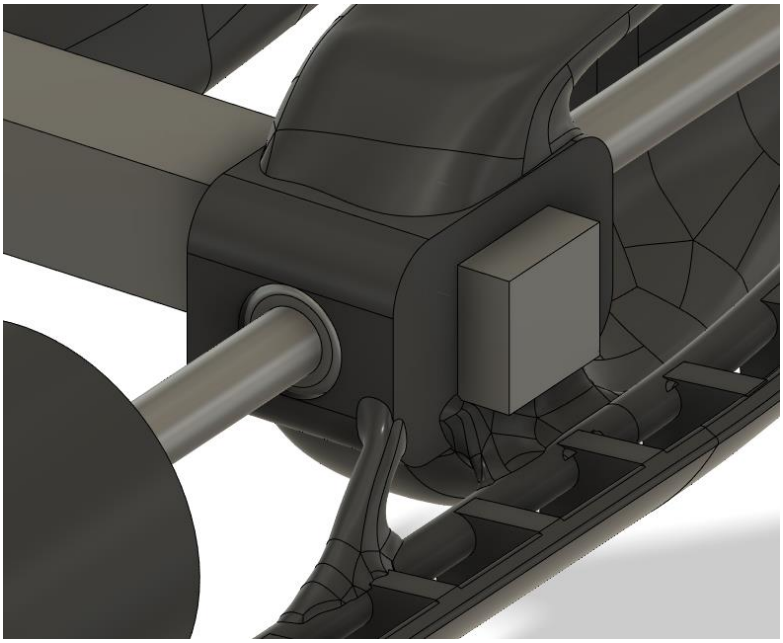


Ilustración 37: Colocación de los rodamientos lineales

Ilustración 38: Rodamiento lineal HK 609

3.2.2. Engranajes

Otra mejora de diseño implementada sobre esta versión del robot se concentra en los engranajes de los sistemas diferenciales, a los que se ha incorporado un elemento de abrazadera que, junto con un tornillo y una tuerca M3, hacen que haya mayor fricción, y por tanto, menos juego, entre los propios engranajes y los ejes que mueven estos.

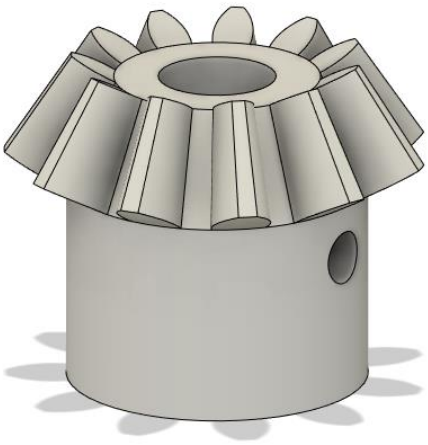


Ilustración 39: Diseño de engranaje de la versión anterior del brazo

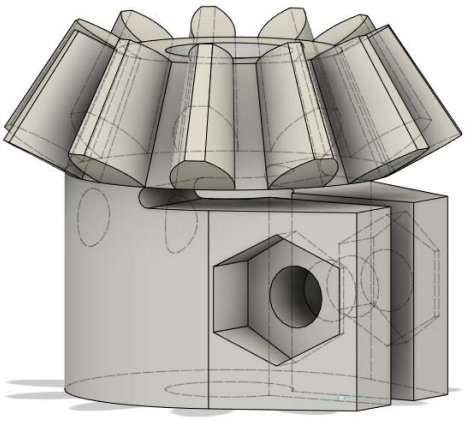


Ilustración 40: Nuevo diseño de engranaje

Además, debido a que sería difícil fabricar este nuevo diseño de engranaje mediante impresión 3D FDM, se ha optado por usar un proceso SLS.

3.2.3. Finales de carrera

También, para instalar los finales de carrera de forma que hay que cambiar el diseño de las piezas del brazo lo menos posible, se han añadido agujeros de atornillado en los topes del brazo posterior y el antebrazo, y se han diseñado dos piezas distintas para instalarlos en los ejes de la segunda y de la cuarta articulación.

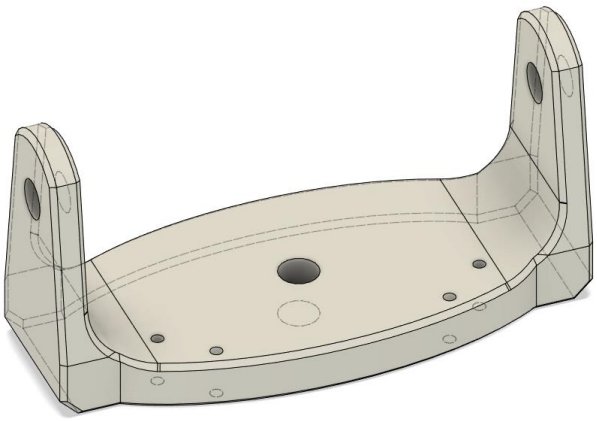


Ilustración 41: Tope brazo posterior

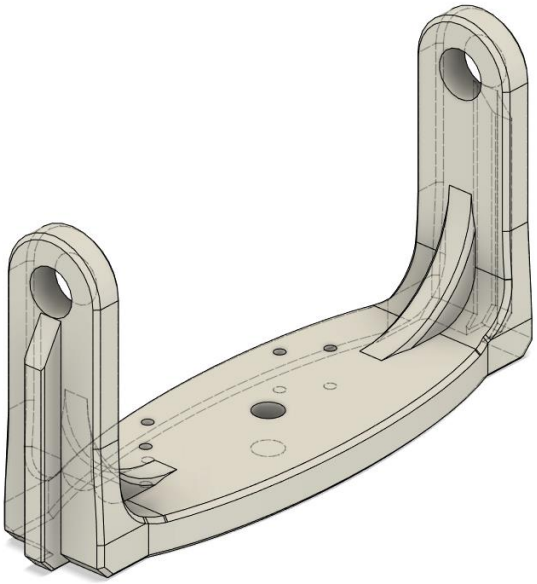


Ilustración 42:Tope antebrazo

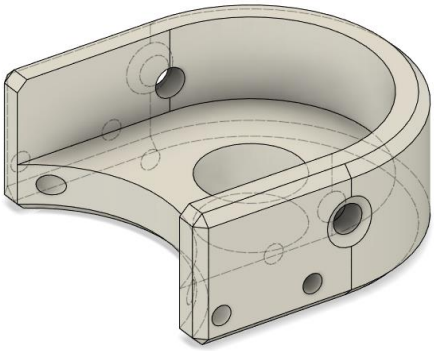


Ilustración 43: Montura finales de carrera 4ª articulación

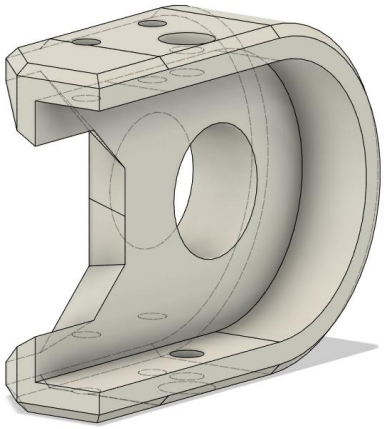


Ilustración 44: Montura finales de carrera 2ª articulación

3.2.4. Otras mejoras y aportaciones

Además de las incorporaciones anteriores, el diseño mecánico del brazo también ha recibido mejoras más pequeñas, como la sustitución de un tubo colocado a lo largo del brazo posterior para guiar el cableado de los motores de la 4ª y la 5ª articulación por una guía ranurada de forma que dichos cables, así como los necesarios para la instalación de los finales de carrera, se fijen al brazo mediante el uso de bridas.

Ilustración 45: Diseño antebrazo con eje ranurado

Otra característica nueva del diseño es la adición de un rack, diseñado para acomodar las placas de circuitos tocadas en el apartado 2.3, sirviendo también para permitir la portabilidad y, por medio de un recipiente adecuado, resistencia a los elementos, de las mismas.

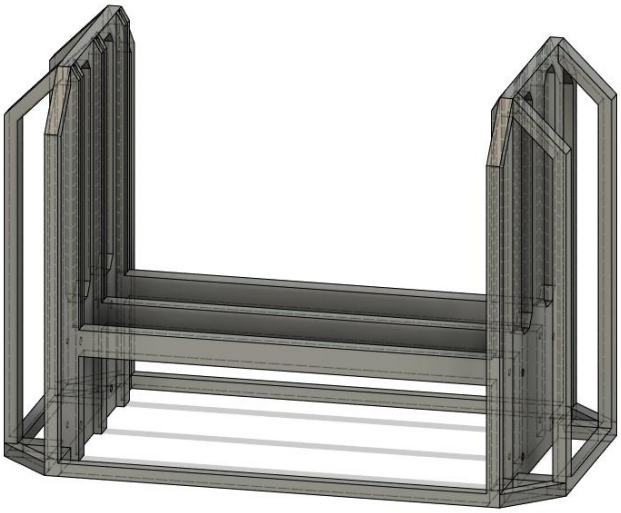


Ilustración 46: Rack para las placas

Un detalle que cabe destacar es que se ha tenido que rediseñar todas las piezas para impresión 3D (excepto el antebrazo, que no ha requerido ninguna modificación) desde cero, al no disponer de acceso a los archivos CAD de los diseños originales.

4. Software y control

4.1. Conceptos previos

Para comprender el trabajo desarrollado en este apartado, es necesario tener claros algunos conceptos relacionados con la cinemática de robots y su representación matemática estandarizada.

4.1.1. Convención de cinemática en robótica

Entendemos como matrices de transformación a matrices utilizadas para representar y realizar transformaciones geométricas en objetos bidimensionales o tridimensionales. Estas transformaciones incluyen traslaciones, rotaciones, escalas, sesgados y combinaciones de estas operaciones. Las matrices de transformación afines son especialmente útiles en la representación y manipulación de objetos en el espacio 2D y

3D, teniendo una estructura cuadrada, con un número de flas y columnas una unidad superior al número de dimensiones que el espacio en el que ese esté trabajando (de forma que una matriz afín para un espacio tridimensional tendría 4 filas y 4 columnas). Cuando una matriz afín solo representa una operación de rotación y una de traslación se conoce como transformación euclídea. Este tipo de matrices de transformación son las más usadas en el campo de la robótica, ya que fácilmente pueden representar posiciones relativas entre elementos rígidos, como son las articulaciones de un robot.

Incidentemente, los robots se suelen entender y representar como cadenas cinemáticas comprendidas por tantos eslabones como articulaciones tenga el robot. Podemos entender que cada uno de dichos eslabones está compuesto por tres elementos: el eslabón en sí, la articulación del mismo, y su sistema de referencia, colocado al final del mismo y cuyo eje Z coincide con el eje de movimiento (rotación o traslación) de la articulación del eslabón siguiente. Esto nos permite obtener la posición del efector final del robot aplicando las transformaciones relativas entre un eslabón y el siguiente una tras otra.

Así pues, para la mayoría de sistemas robóticos, estas trasformaciones relativas se pueden expresar mediante cuatro parámetros, llamados parámetros de Denavit-Hartenberg, a partir de los cuales se puede construir una matriz de transformación afín que represente dicha transformación relativa.

Estos cuatro parámetros son:

- d: Offset desde el eje z de la articulación anterior a la normal común entre ambas esta y la siguiente.
- θ: Diferencia de ángulo del eje X de la articulación anterior a la siguiente, sobre el ángulo Z
- a: Longitud de la normal común

- α: Diferencia de ángulo del eje Z de la articulación anterior a la siguiente, sobre la normal común

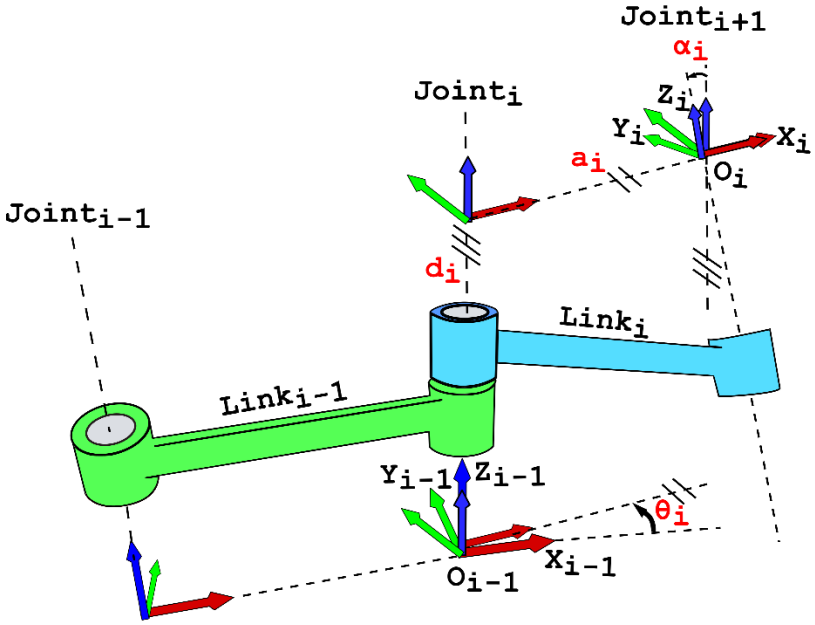


Ilustración 47: Representación cinemática de dos articulaciones de un robot, con diagrama ilustrativo de parámetros de Denavit-Hartenberg

Así pues, la transformación euclídea resultante responde a la siguiente ecuación:

$$A_{i-1}^{i} = \begin{pmatrix} \cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & a\cos\theta \\ \sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & a\sin\theta \\ 0 & \sin\alpha & \cos\alpha & d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Que se puede descomponer de la siguiente forma:

$$A_{i-1}^{i} = \begin{pmatrix} & R & & T \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Representando las submatrices R (de 3x3) y T (de 3x1) las partes rotacional y traslacional de la transformación.

Así pues, el brazo robot abordado en este trabajo tiene los siguientes parámetros DH:

Tabla 2: Parámetros Denavit-Hartenberg del brazo robot

Articulación	d	θ	a	α
1º	0	q_1	0.3	0
2ª	0	q_2 - π/2	0	π/2
3ª	0.26	q_3	0	-π/2
4ª	0	q_4	0	π/2
5ª	0.06	q_5	0	0

Siendo q_i la rotación de la articulación i con respecto a su posición neutral.

4.1.1. Cinemática directa

Como ya se ha dicho en el apartado anterior, calcular la posición y rotación del efector final resulta trivial gracias a las matrices de transformación afines obtenidas de los parámetros DH del robot, mediante una sucesión de multiplicaciones matriciales.

Cabe destacar que también hay que tener en cuenta una matriz de transformación extra, que representa la rotación y traslación del sistema de referencia de la base con respecto a lo que tomemos como origen de coordenadas.

4.1.1. Cinemática inversa

A la hora realizar el cálculo de la cinemática inversa, es decir, obtener las posición articular todas las articulaciones que debería tener el robot de forma que su efector final tenga una posición y rotación determinadas, se ha optado por escoger la posición de la muñeca en el eje XZ, puesto que el robot solo tiene 5 grados de libertas y la rotación del efector final restringe 3 de ellos.

4.1.1. Generación de trayectorias

Para controlar el movimiento de robots de forma precisa, es común utilizar un sistema de generación y seguimiento de trayectorias, en el

que el robot calcula que trayectoria tendría que seguir para realizar un movimiento determinado.

Estas trayectorias pueden verse clasificadas en diferentes tipos:

- En espacio articular o cartesiano: En las primeras se pasa de una posición inicial a una final en cada articulación de forma independiente, de forma que el estado de una de ellas no afecte a las demás, mientras que en las segundas se controla la posición y rotación del efector final, requiriendo un cálculo de cinemática inversa para obtener la posiciones articulares correspondientes, y permitiendo distintos tipos de movimiento del efector final, como puede ser un movimiento lineal.
- De posición o de velocidad: Tal y como el nombre indica, las primeras pasan de una posición inicial a una posición final determinada por el usuario o el sistema de control; mientras que las segundas hacen que el robot siga un perfil de velocidad determinado.
- Relativas: En las primeras, la posición final es relativa frente a la inicial, mientras que en las segundas ambas variables son independientes
- Por su tipo de perfil: polinomiales, trapezoidales.

4.1.2. Control de una articulación

Para realizar el control de una articulación del brazo, se usará un método de control PD+G, en el que se comparará la posición y velocidad dicha articulación frente a unas de referencia, obtenidas por ejemplo, de un generador de trayectoria, introduciendo además una componente de compensación de gravedad, que calculará el par que tendrá que ejercer el motor de cada articulación para compensar el producido por las fuerzas de gravedad,

Así pues, este sistema de control se puede expresar de la siguiente forma:

$$\tau_i = K_p\left(q_i^{ref} - q_i\right) + K_v\left(\dot{q}_i^{ref} - \dot{q}_i\right) + G(q)$$

Siendo *τ* el par motor de la articulación, *Kp* y *Kv* ganancias de control y *G(q)* el término de compensación de la gravedad.

4.1.1. Jacobiana

En robótica, se conoce como matriz jacobiana a la matriz que representa la relación de las velocidades del efector final del robot con respecto a las velocidades de sus articulaciones (por ejemplo, la velocidad de la muñeca o la herramienta que sostiene). Se representa generalmente como J y puede tener diferentes dimensiones según el número de grados de libertad del robot. Hay dos formas de calcular la matriz jacobiana de un robot, la forma analítica y la geométrica.

La matriz jacobiana analítica se calcula tomando las derivadas parciales de las coordenadas del extremo efector con respecto a las articulaciones del robot, mientras que la geométrica utiliza las matrices de transformación homogéneas de cada articulación, relacionando la velocidad de cada articulación con las velocidades de traslación y rotación del efector final.

La jacobiana inversa, como su nombre indica, es la matriz inversa de la jacobiana, y consecuentemente, representa la relación de las articulaciones del robot a partir de las velocidades del efector final.

4.1.2. Compensación de la gravedad

Para realizar el cálculo del par que ejercen las fuerzas de gravedad sobre cada una de las articulaciones se necesitará conocer el centro de masa de cada una de ellas, y la posición de dicho centro de masa con respecto al sistema de referencia de las mismas (que, recordemos, está situado en el eje de la siguiente articulación de la cadena cinemática), así como las matrices de transformación euclídeas de cada una de ellas.

Una vez conocidos los valores de las variables anteriores, se puede calcular el efecto de la gravedad sobre cada uno de los elementos de la cadena cinemática del robot mediante un algoritmo recursivo de orden descendente, empezado por la última articulación, lo cual se debe a que el par producido por la gravedad sobre un eslabón de la cadena se ve afectado por todos los eslabones posteriores. El algoritmo utilizado se ilustra mediante el siguiente diagrama:

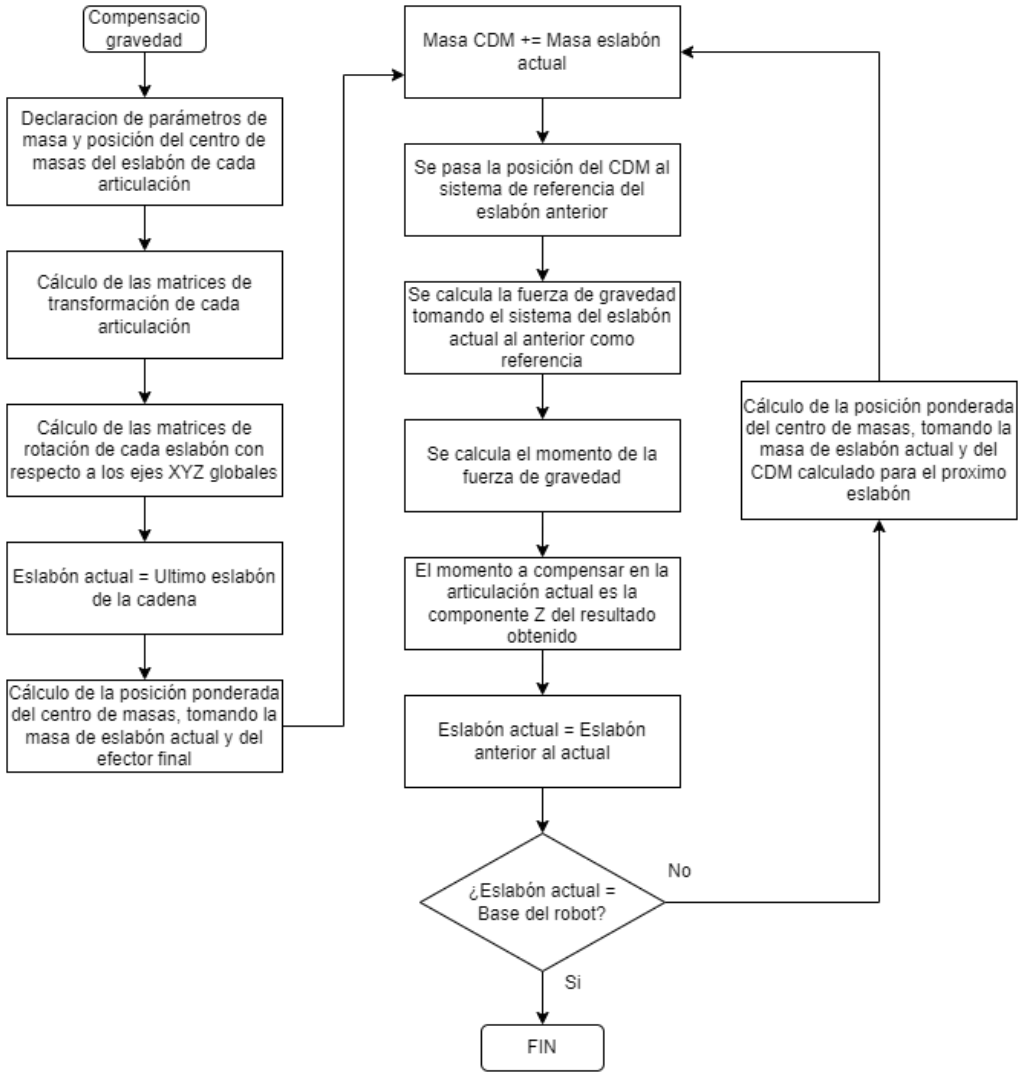


Ilustración 48: Diagrama de flujo de algoritmo de compensación de gravedad

4.2. Arquitectura del programa

Como ya se ha dicho, la paca microcontroladora se trata de una Teensy 4.1, que se programará utilizando el IDE de Arduino. Para el correcto

funcionamiento del programa, escrito en C++, utilizaremos las siguientes librerías:

- TeensyThreads, para la utilización de hilos en paralelo, de forma que la Teensy pueda ejecutar varias funciones distintas de forma paralela (solo puede ejecutar una a la vez, pero se ejecutan en particiones pequeñas)
- ArduinoEigen, que incluye una gran variedad de funciones de álgebra de vectores y matrices que resultan muy útiles a la hora de realizar los cálculos necesarios para controlar el robot
- CircularBuffer, para implementar de órdenes que ejecute el robot; Encoder, para la lectura de los encoders de los motores
- Serial, para la el envío de comando y comunicación con el brazo robot

Y se han escrito cuatro librerías distintas adicionales:

- RobotArmKinematics: para el cálculo de la cinemática directa, la cinemática inversa y la jacobiana.
- Trajectory: Para el cálculo de trayectorias que se pretende que siga el robot. Usa la librería anterior.
- Command: Implementa un tipo de variable, que podemos entender como una orden que se pretende que siga el robot, y que almacena el tipo de orden, el número de pasos de la trayectoria de la propia orden y el paso actual en el que se encuentra el programa de control, las posiciones articulares de referencia del robot para cada paso, y una variable que indica si la orden ha terminado.
- RobotArmDynamics: para el cálculo de los pares de compensación de gravedad del brazo.

Así pues, el programa principal del robot se ejecuta de forma discreta en intervalos de 10 milisegundos, y su estructura ve ilustrada en los siguientes diagramas:

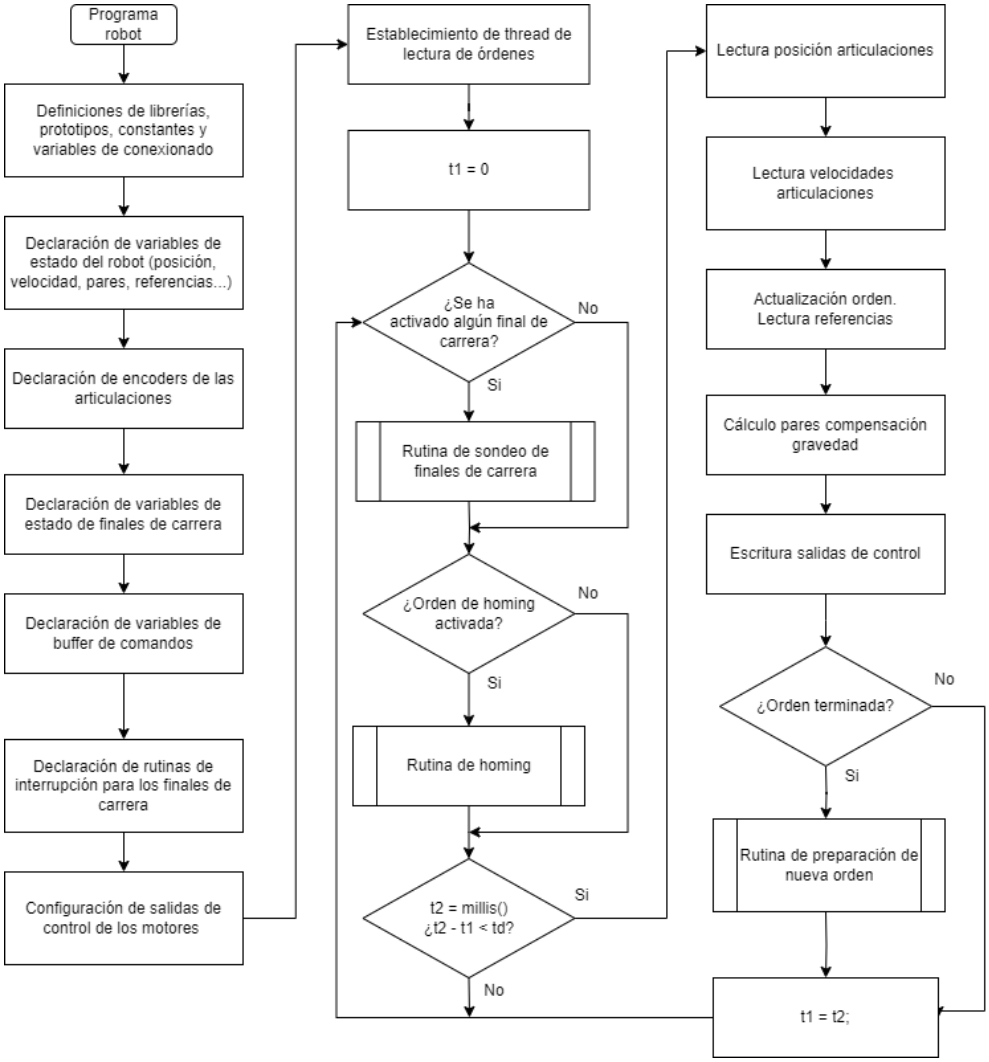


Ilustración 49: Diagrama de flujo del programa de control del brazo

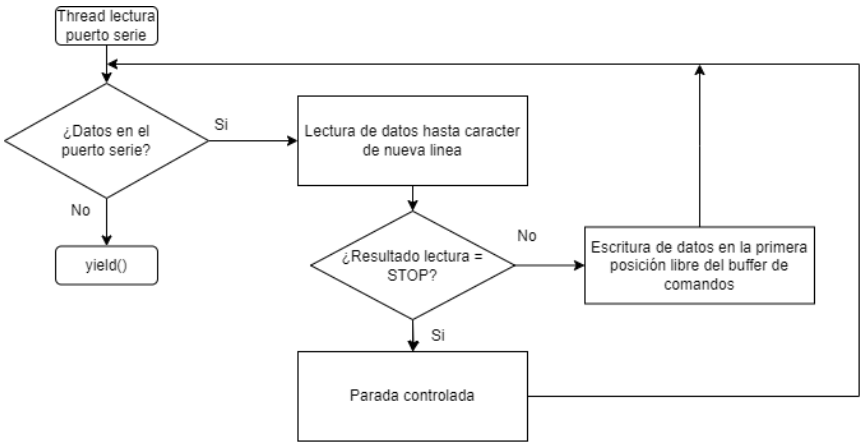


Ilustración 50: Diagrama de flujo del hilo de lectura del puerto serie

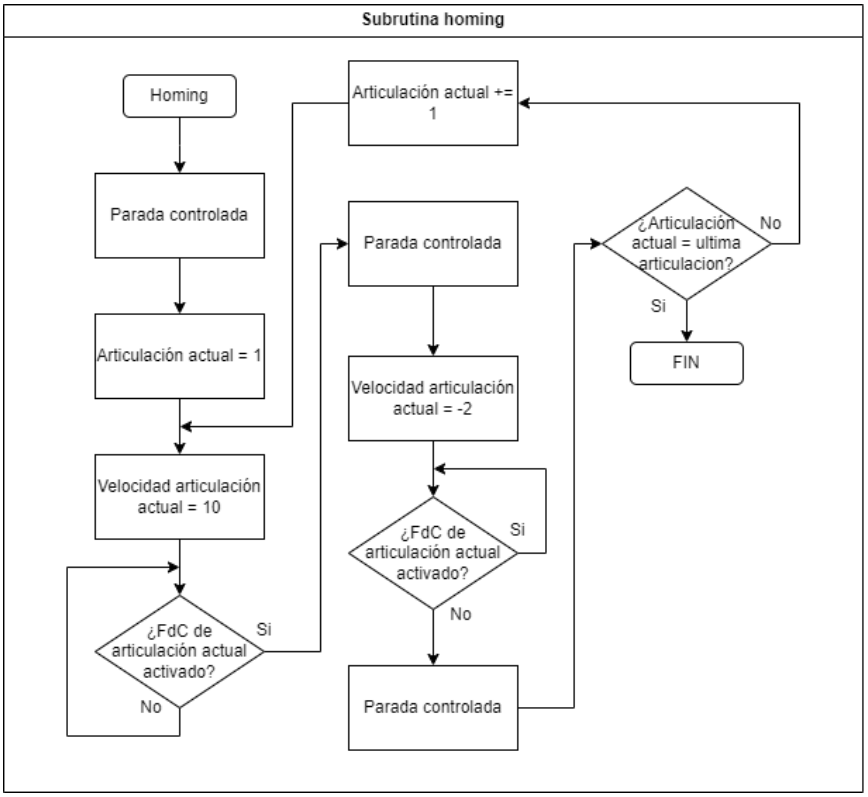


Ilustración 51: Diagrama de flujo de subrutrina de homing

4.3. Implementación de funciones

Para el desarrollo y la implementación de la mayoría de funciones cinemáticas se ha usado MATLAB, ya que con la extensión Robotic Toolbox [13] es posible caracterizar el robot y verificar que las funciones que se implementen tengan unos resultados correctos, comparando dichos resultados con los obtenidos por las funciones del toolbox para los mismos parámetros de entrada.

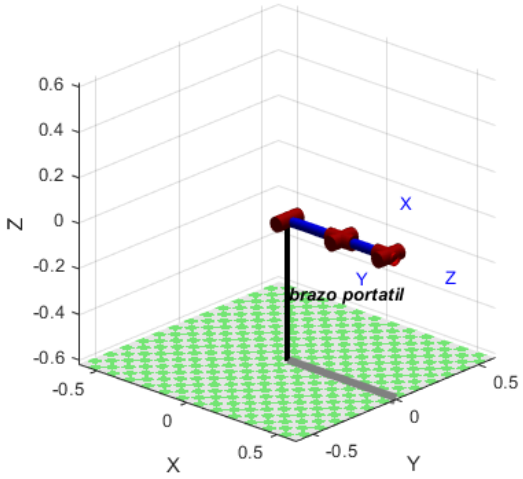


Ilustración 52: Representación gráfica del brazo robot obtenida mediante el Robotic Toolbox de Matlab mediante los parámetros DH del mismo

4.3.1. Cinemática directa

Para la implementación de la cinemática directa, se han usado las matrices de transformación homogéneas obtenidas de los parámetros de Denavit-Hartemberg del brazo, y se ha estudiado la forma más computacionalmente eficiente de realizar el cálculo cinemático. Así pues, finalmente, se ha optado por utilizar una única multiplicación de matrices 4x4, siendo la primera la matriz de transformación del origen de coordenadas a la articulación 3, y la segunda, la matriz de transformación de la tercera articulación al efector final.

$$T_{03} = \begin{pmatrix} -\sin(q_1+q_2)\cos q_3 & -\cos(q_1+q_2) & \sin(q_1+q_2)\sin q_3 & 0.3\cos q_1 + 0.26\cos(q_1+q_2) \\ -\sin q_3 & 0 & -\cos q_3 & 0 \\ \cos(q_1+q_2)\cos q_3 & -\sin(q_1+q_2) & -\cos(q_1+q_2)\sin q_3 & 0.3\cos q_1 + 0.26\cos(q_1+q_2) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_{45} = \begin{pmatrix} \cos q_4 \cos q_5 & -\cos q_4 \sin q_5 & \sin q_4 & 0.06\sin q_4 \\ \sin q_4 \cos q_5 & -\sin q_4 \sin q_5 & -\cos q_4 & 0.06\cos q_4 \\ \sin q_5 & \cos q_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_{05} = T_{03} \cdot T_{45}$$

4.3.1. Cinemática inversa

Para el cálculo de la cinemática inversa, se han utilizado los resultados del estudio analítico realizado en el apartado anterior, y se ve dividida en dos partes. La primera deriva los valores de posición de las dos primeras articulaciones para una posición determinada de la muñeca (que solo precisa de dos coordenadas para ser definida, al verse su movimiento restringido en un plano), y la segunda calcula los valores de posición de la tercera, cuarta y quinta articulación tomando en que las tres forman una muñeca esférica, de forma que se pueden usar los métodos de cinemática inversa utilizados para este tipo de mecanismos.

4.3.1. Jacobiana

Para la obtención de la jacobiana, se ha calculado de forma geométrica, de nuevo utilizando las matrices de transformación obtenidas para la implementación de la cinemática directa.

4.3.1. Trayectoria en espacio de articulaciones

Este tipo de trayectoria se trata de una polinomial de 5ª orden ya implementada anteriormente en el trabajo de Roland [11].

4.3.2. Trayectoria en espacio cartesiano

Para implementar este tipo de trayectoria, se ha implementado un perfil trapezoidal por partes unidimensional, adaptándolo posteriormente para el control coordinado de múltiples variables.

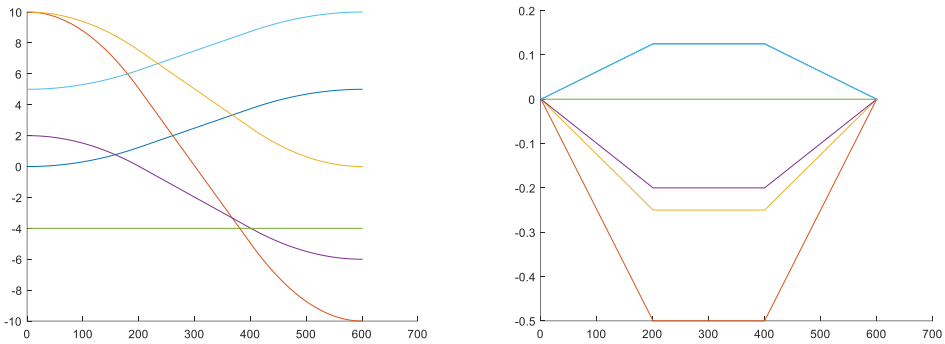


Ilustración 53: Gráfica de posiciones y velocidades para una trayectoria trapezoidal multidimensional

Así pues, se usa dicho perfil trapezoidal para generar una trayectoria en espacio cartesiano, pasando las posiciones y velocidades de cada uno de los pasos de la misma mediante cinemática inversa y la matriz inversa de la matriz jacobiana.

5. Resultados y conclusiones

5.1. Resultados

Desgraciadamente, debido al funcionamiento defectuoso de una de las etapas de potencia, no ha sido posible realizar una demostración del funcionamiento del brazo completo, aunque sí que se ha grabado una demostración en la que se controlan las dos últimas articulaciones del brazo. [https://www.youtube.com/watch?v=BLUiLHmrayM]

Se ha comprobado así el correcto funcionamiento de las funciones de control del brazo (aunque en el caso de funciones como el seguimiento de trayectorias cartesianas no se puede visualizar debido a la falta de movimiento en las articulaciones anteriores), los finales de carrera o las rutinas de homing.

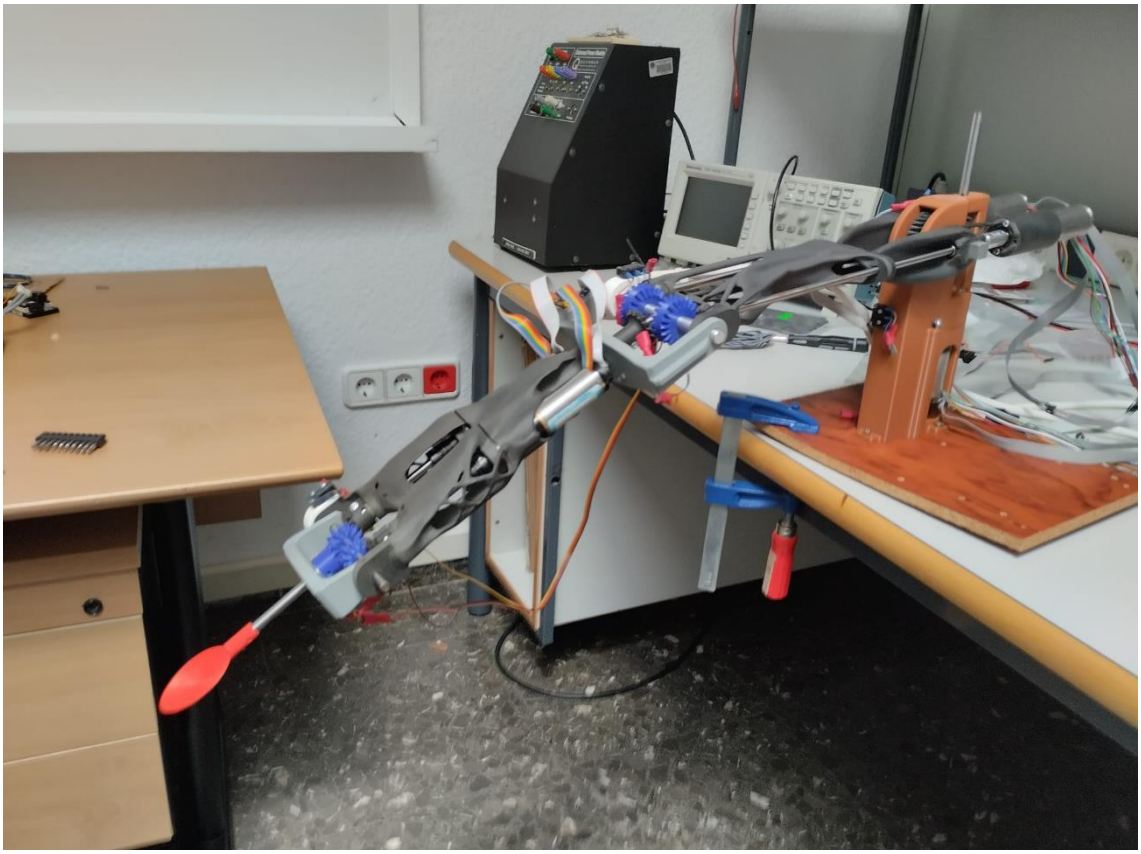


Ilustración 54: Barzo robot completo, con cableado y fines de carrera

5.2. Conclusiones

A lo largo del desarrollo de este trabajo se han introducido elementos para una mejora de la seguridad y el comportamiento del mismo, introduciendo una mejora no solo de las conexiones electrónicas y componentes mecánicos, para así reducir riesgos de seguridad, juego mecánico y esfuerzos adversos de rozamiento entre piezas del mismo, así como una reescritura prácticamente completa del código de programa usado para moverlo, sino que además se han introducido elementos nuevos que mejoran su comportamiento, como los finales de carrera, que mejoran aún más la seguridad del brazo al eliminar colisiones entre articulaciones y permiten la realización de una rutina de homing, que elimina la necesidad de hacer que arranque en una posición neutral (con todas las posiciones articulares puestas a 0); así como un algoritmo de compensación de la gravedad, que

elimina los efectos adversos que tienen los esfuerzos de gravedad sobre el comportamiento del mismo.

5.3. Posibles mejoras

- Falta poner el brazo entero en marcha, sustituyendo la etapa de potencia defectuosa.
- Estudio de la adaptación del diseño del brazo a un caso concreto de usuario con discapacidad que requiera de ella (por ejemplo, una reducción del diseño para que lo use un niño)
- Pasar el motor de la primera articulación de estar fijo a la base, a esta unido a dicha articulación, haciendo que la corona del tornillo sin fin que transmite la potencia a dicha articulación este fija

Referencias bibliográficas

[1] Instituto Nacional de Estadística - Encuesta sobre Discapacidades, Deficiencias y Estado de Salud, 1999. Avance de Resultados. https://www.ine.es/prodyser/pubweb/discapa/disctodo.pdf (accedido del 2 de Marzo de 2023).

[2] European Statistical Office, Disability statistics – health. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Archive:Disability_statistics_-_health/, 2003 (accedido del 2 de Marzo de 2023).

[3]: A. Kapsalyamov, S. Hussain, And P. K. Jamwal, State-of-the-Art Assistive Powered Upper Limb Exoskeletons for Elderly, IEEE Access, VOLUME 8, 2020, 178991- 179001.

[4], Yang Shen, Peter Walker Ferguson and Jacob Rosen, upper limb exoskeleton systems—overview, Wearable Robotics. DOI: https://doi.org/10.1016/B978-0-12-814659-0.00001-1, © 2020 Elsevier Inc. All rights reserved.

[5] K. Homma and T. Arai, Design of an Upper Limb Motion Assist System with Parallel Mechanism, Proceedings of 1995 IEEE International Conference on Robotics and Automation, Nagaoya, Japan, 1995

[6] P. Garrec, J.P. Friconneau, Y. Measson and Y. Perrot, ABLE, an innovative transparent exoskeleton for the upper-limb, 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France, 2008

[7], M. Mihelj, T. Nef, R. Riener, ARMin II – 7 DoF rehabilitation robot: mechanics and kinematics, 2007 IEEE International Conference on Robotics and Automation, Roma, Italy, 2007.

[8] Urs Keller, Hubertus J. A. van Hedel, Verena Klamroth-Marganska, and Robert Riener, ChARMin: The First Actuated Exoskeleton Robot for Pediatric Arm Rehabilitation, IEEE/ASME TRANSACTIONS ON MECHATRONICS, VOL. 21, NO. 5, OCTOBER 2016.

[9] V. Cimolin, A. Vagnini, C. Germiniasi, M. Galli, I. Pacifici, L. Negri, E. Beretta, L.Piccinini, The Armeo Spring as training tool to improve upper limbfunctionality in hemiplegic Cerebral Palsy: a pilot study, 2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), Bologna, Italy, 2016, pp. 1-4, doi: 10.1109/RTSI.2016.7740602.

[10] Mulet Alberola, J., 2017. Trabajo Fin de Master: Estudio y diseño de un brazo robótico de asistencia a discapacitados. Valencia: Universidad Politécnica de Valencia

[11] Rojar Moreno, R., 2022. Trabajo Fin de Master: Diseño e implementación de un brazo robot portátil para la ayuda a discapacitados. Valencia: Universidad Politécnica de Valencia

[12] Tutorial de ejemplo de Diseño Generativo en Fusion360 https://help.autodesk.com/view/fusion360/ENU/?guid=GD-TUT-GE-QS-ASSIGN-GEOM-TYPE

[13] Página del Toolox Robótico de MATLAB, https://petercorke.com/toolboxes/robotics-toolbox/

[14] Video de demostración de funcvionamiento del brazo robot, https://www.youtube.com/watch?v=BLUiLHmrayM

Anexo 1: Código de programa del brazo

• Programa principal

```
#include <Arduino.h>
#include <cmath>
#include <Encoder.h>
#include <TeensyThreads.h>
#include <ArduinoEigen.h>
#include <CircularBuffer.h>
#include "RobotArmKinematics.h"
#include "Command.h"
#include "RobotArmDynamics.h"

#define sign(x) ((x) < 0 ? -1 : ((x) > 0 ? 1 : 0))

#define Pi 3.14159265

#define td 0.01

#define Kp1 40
#define Kv1 60

#define Kp23 60
#define Kv23 90

#define Kp45 50
#define Kv45 80

bool testFlag;

// DEFINICIONES MOTORES ---------------------------------------------
------
// -Motor 1
#define pinPWM_M1 0
#define pinEnable_M1 23

// #define gearRatio_M1 1/318 * 1/11.2
#define gearRatio_M1 1/318

#define encPPR_M1 4*1000
#define pinEncA_M1 7
#define pinEncB_M1 8

#define Kt_M1 0.0246
```

```
#define nomCurr_M1 1.52

// -Motor 2
#define pinPWM_M2 1
#define pinEnable_M2 22

#define gearRatio_M2 1/190

#define encPPR_M2 4*4096
#define pinEncA_M2 9
#define pinEncB_M2 10

#define Kt_M2 0.0228
#define nomCurr_M2 2.48

// -Motor 3
#define pinPWM_M3 2
#define pinEnable_M3 21

#define gearRatio_M3 1/190

#define encPPR_M3 4*4096
#define pinEncA_M3 11
#define pinEncB_M3 12

#define Kt_M3 0.0228
#define nomCurr_M3 2.48

// Articulación 2/3
#define twistGearRatio_23 11/16
#define flexGearRatio_23 11/22

// -Motor 4
#define pinPWM_M4 3
#define pinEnable_M4 20

#define gearRatio_M4 1/104

#define encPPR_M4 4*512
#define pinEncA_M4 13
#define pinEncB_M4 14

#define Kt_M4 1.4
#define nomCurr_M4 0.0174

// -Motor 5
```

```
#define pinPWM_M5 4
#define pinEnable_M5 19

#define gearRatio_M5 1/104

#define encPPR_M5 4*512
#define pinEncA_M5 15
#define pinEncB_M5 16

#define Kt_M5 1.4
#define nomCurr_M5 0.0174

// Articulación 4/5
#define twistGearRatio_45 11/16
#define flexGearRatio_45 11/16

// DEFINICIONES FDC ----------------------------------------------------
------
#define pinLS_j1 24
#define pinLS_j2 25
#define pinLS_j3 26
#define pinLS_j4 27
#define pinLS_j5 28

#define pinLS_pos 31 //definición pines input
#define pinLS_neg 32

// Variables globales --------------------------------------------------
------

double q[5]; //Posicion de cada eje
double qv[5]; //Velocidad de cada eje

double q_m_23[2]; //Posiciones ejes diferenciales 2ª y 3ª articulación
double qv_m_23[2]; //Velocidades ejes diferenciales 2ª y 3ª
articulación

double q_m_45[2]; //Posiciones ejes diferenciales 4ª y 5ª articulación
double qv_m_45[2]; //Posiciones ejes diferenciales 4ª y 5ª articulación

double pos_final[2];
double vel_final[2];

double q_last[5];
```

```
// Variables globales de control ----------------------------------------
-----

unsigned long tiempo1 = 0;
unsigned long tiempo2 = 0;

double tau[5];
double tau_m_23[2];
double tau_m_45[2];

double tau_g[5];

double q_ref[5] = {0, 0, 0, 0, 0}; //Posicion de cada eje
double qv_ref[5] = {0, 0, 0, 0, 0}; //Velocidad de cada eje

double q_ref_m_23[2]; //Posiciones ejes diferenciales 2ª y 3ª
articulación
double qv_ref_m_23[2]; //Velocidades ejes diferenciales 2ª y 3ª
articulación

double q_ref_m_45[2]; //Posiciones ejes diferenciales 2ª y 3ª
articulación
double qv_ref_m_45[2]; //Posiciones ejes diferenciales 2ª y 3ª
articulación

// Variables seguridad ------------------------------------------------
------
bool LimitSwitch_Eje1[2];
bool LimitSwitch_Eje2[2];
bool LimitSwitch_Eje3[2];
bool LimitSwitch_Eje4[2];
bool LimitSwitch_Eje5[2];

bool LS_flag = 0;

// Encoders de cada motor
Encoder encMotor1(pinEncA_M1, pinEncB_M1);
Encoder encMotor2(pinEncA_M2, pinEncB_M2);
Encoder encMotor3(pinEncA_M3, pinEncB_M3);
Encoder encMotor4(pinEncA_M4, pinEncB_M4);
Encoder encMotor5(pinEncA_M5, pinEncB_M5);

// Variables de conexion serie para ordenes ---------------------------
------
CircularBuffer<String,50> commandBuffer;
CommandTrajectory currentCommand = CommandTrajectory();
```

```
Threads::Mutex mylock;

bool avisoComandoTerminado = false;

bool home_flag;

void setup() {

  // Rutinas de interrupcion
  attachInterrupt(digitalPinToInterrupt(pinLS_j1), stopLimitSwitch,
RISING);

  attachInterrupt(digitalPinToInterrupt(pinLS_j2), stopLimitSwitch,
RISING);
  attachInterrupt(digitalPinToInterrupt(pinLS_j3), stopLimitSwitch,
RISING);
  attachInterrupt(digitalPinToInterrupt(pinLS_j4), stopLimitSwitch,
RISING);
  attachInterrupt(digitalPinToInterrupt(pinLS_j5), stopLimitSwitch,
RISING);

  pinMode(pinLS_pos, OUTPUT);
  digitalWrite(pinLS_pos, LOW);

  pinMode(pinLS_neg, OUTPUT);
  digitalWrite(pinLS_neg, LOW);

  // Se preparan las salidas de control de los motores
  analogWrite(pinPWM_M1, 127);

  pinMode(pinEnable_M1, OUTPUT);
  digitalWrite(pinEnable_M1, HIGH);

  analogWrite(pinPWM_M1, 127);

  pinMode(pinEnable_M1, OUTPUT);
  digitalWrite(pinEnable_M1, HIGH);

  analogWrite(pinPWM_M1, 127);

  pinMode(pinEnable_M1, OUTPUT);
  digitalWrite(pinEnable_M1, HIGH);

  // interrupts();
  Serial.begin(9600);
```

```
  threads.setSliceMicros(20);


  // threads.addThread(mainThread);
  threads.addThread(serial_comm);
}

void loop() {
  //Threads::Scope m(mylock);
  tiempo1 = millis();
  if (LS_flag) {
    probeLimitSwitches();
  }
  if (home_flag) {
    homing();
  }

  if (tiempo1 > tiempo2 + td * 1000) {
    // Lectura posicion
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();

    // Obtención referencia
    getReferences();

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));

    if(currentCommand.isFinished()) {
      if (!avisoComandoTerminado) {
        Serial.println("Comando terminado");
        avisoComandoTerminado = true;
```

```
      }


      if(!commandBuffer.isEmpty()) {
        Threads::Scope m(mylock);
        getNextCommand()
      }
    }


    tiempo2 = millis();
  }
}

void serial_comm() {
  while(1) {
    while(Serial.available() > 0) {
      String cmd = "";

      Threads::Scope m(mylock);
      cmd = Serial.readStringUntil('\n');                // "cmd" keep
the recived byte


      if(cmd == "debug") {
        encMotor1.write(0);
      }
      else {
        commandBuffer.push(cmd);
      }
    }

    threads.yield();
  }
}


void stopLimitSwitch() {
  LS_flag = 1;
}

void readJointPositions() {
  q_m_23[0] =  encMotor2.read()*gearRatio_M2/encPPR_M2;
  q_m_23[1] =  encMotor3.read()*gearRatio_M3/encPPR_M3;
  q[3] = ( q_m_23[0] - q_m_23[1] ) / 2 * flexGearRatio_23;
  q[4] = ( q_m_23[0] + q_m_23[1] ) / 2 * flexGearRatio_23;
```

```
  q_m_45[0] =  encMotor4.read()*gearRatio_M4/encPPR_M4;
  q_m_45[1] =  encMotor5.read()*gearRatio_M5/encPPR_M5;
  q[3] = ( q_m_45[0] - q_m_45[1] ) / 2 * flexGearRatio_45;
  q[4] = ( q_m_45[0] + q_m_45[1] ) / 2 * flexGearRatio_45;
}

void getJointVelocities() {
  for (int i = 0; i < 5; i++) {
      qv[i] = (q[i] - q_last[i]) * (tiempo1 - tiempo2);
      q_last[i] = q[i];
  }
}

void getReferences() {
  currentCommand.update();
  currentCommand.get_q_ref(q_ref);
  currentCommand.get_qv_ref(qv_ref);
}

void calculateTorques() {
  tau[0] = Kp1*(q_ref[0]-q[0]) + Kv1*(qv_ref[0]-qv[0]) + tau_g[0];

  tau[1] = Kp45*(q_ref[1]-q[1]) + Kv45*(qv_ref[1]-qv[1]) + tau_g[1];
  tau[2] = Kp45*(q_ref[2]-q[2]) + Kv45*(qv_ref[2]-qv[2]) + tau_g[2];
  tau_m_23[0] = (tau[4] * twistGearRatio_45 + tau[3] *
flexGearRatio_45)/2;
  tau_m_23[1] = (tau[4] * twistGearRatio_45 - tau[3] *
flexGearRatio_45)/2;

  tau[3] = Kp45*(q_ref[3]-q[3]) + Kv45*(qv_ref[3]-qv[3]) + tau_g[3];
  tau[4] = Kp45*(q_ref[4]-q[4]) + Kv45*(qv_ref[4]-qv[4]) + tau_g[4];
  tau_m_45[0] = (tau[4] * twistGearRatio_45 + tau[3] *
flexGearRatio_45)/2;
  tau_m_45[1] = (tau[4] * twistGearRatio_45 - tau[3] *
flexGearRatio_45)/2;
}

void getNextCommand() {
  String cmd = commandBuffer.pop();

  bool badCmdFlag = false;

  String varSubstring =
cmd.substring(cmd.indexOf('(')+1,cmd.indexOf(')'));
```

```
  char buff[64];

  varSubstring.toCharArray(buff, varSubstring.length()+1);

  int nvars = 0;
  char *token;
  const char *delim = ",";

  token = strtok(buff, delim);

  while (token != NULL) {
    nvars++;
    token = strtok(NULL, delim);
  }

  double vars[nvars];

  varSubstring.toCharArray(buff, varSubstring.length()+1);

  if (nvars > 1) {
    token = strtok(buff, delim);

    for(int i = 0; i < nvars; i++) {
      vars[i] = strtod(token,NULL);

      token = strtok(NULL, delim);
    }
  }

  if( (cmd.startsWith("MoveAbsJ")) & (nvars == 5) ) {
    double qf[5] = {vars[0], vars[1], vars[2], vars[3], vars[4]};

    currentCommand.reset();
    currentCommand.generateJointPosTrajectory(q, qf, 15, td);
    avisoComandoTerminado = false;
  }
  else if( (cmd.startsWith("MoveJ")) & (nvars == 5) ) {
    double pos_rot_0[5];
    double pos_rot_f[5] = {vars[0], vars[1], vars[2], vars[3],
vars[4]};

    Eigen::Matrix4d T = cinDir(q[0],q[1],q[2],q[3],q[4]);
    tform2eul(T, pos_rot_0[0], pos_rot_0[1], pos_rot_0[2],
pos_rot_0[3], pos_rot_0[4]);

    currentCommand.reset();
```

```
    currentCommand.generateCartJointTrajectory(pos_rot_0, pos_rot_f,
15, td);
    avisoComandoTerminado = false;
  }
  else if( (cmd.startsWith("MoveL")) & (nvars == 5) ) {
    double pos_rot_0[5];
    double pos_rot_f[5] = {vars[0], vars[1], vars[2], vars[3],
vars[4]};

    Eigen::Matrix4d T = cinDir(q[0],q[1],q[2],q[3],q[4]);
    tform2eul(T, pos_rot_0[0], pos_rot_0[1], pos_rot_0[2],
pos_rot_0[3], pos_rot_0[4]);

    currentCommand.reset();
    currentCommand.generateCartPoseTrajectory(pos_rot_0, pos_rot_f,
200, 25, 170, 20, td);
    avisoComandoTerminado = false;
  }
  else if( (cmd.startsWith("MoveAbsJ_incr")) & (nvars == 5) ) {
    double qf[5] = {q[0] + vars[0], q[1] + vars[1], q[2] + vars[2],
q[3] + vars[3], q[4] + vars[4]};

    currentCommand.reset();
    currentCommand.generateJointPosTrajectory(q, qf, 15, td);
    avisoComandoTerminado = false;
  }
  else if( (cmd.startsWith("MoveJ_incr")) & (nvars == 5) ) {
    double pos_rot_0[5];
    Eigen::Matrix4d T = cinDir(q[0],q[1],q[2],q[3],q[4]);
    tform2eul(T, pos_rot_0[0], pos_rot_0[1], pos_rot_0[2],
pos_rot_0[3], pos_rot_0[4]);

    double pos_rot_f[5] = {pos_rot_0[0] + vars[0], pos_rot_0[1] +
vars[1], pos_rot_0[2] + vars[2], pos_rot_0[3] + vars[3], pos_rot_0[4] +
vars[4]};

    currentCommand.reset();
    currentCommand.generateCartJointTrajectory(pos_rot_0, pos_rot_f,
15, td);
    avisoComandoTerminado = false;
  }
  else if( (cmd.startsWith("MoveL_incr")) & (nvars == 5) ) {
    double pos_rot_0[5];
    Eigen::Matrix4d T = cinDir(q[0],q[1],q[2],q[3],q[4]);
    tform2eul(T, pos_rot_0[0], pos_rot_0[1], pos_rot_0[2],
pos_rot_0[3], pos_rot_0[4]);
```

```
    double pos_rot_f[5] = {pos_rot_0[0] + vars[0], pos_rot_0[1] +
vars[1], pos_rot_0[2] + vars[2], pos_rot_0[3] + vars[3], pos_rot_0[4] +
vars[4]};

    currentCommand.reset();
    currentCommand.generateCartPoseTrajectory(pos_rot_0, pos_rot_f,
200, 25, 170, 20, td);
    avisoComandoTerminado = false;
  }
  else if( (cmd.startsWith("SpeedJ")) & (nvars == 7) ) {
    double SPEED[5] = {vars[0], vars[1], vars[2], vars[3], vars[4]};
    double tv = vars[5];
    double t_accel = vars[6];

    currentCommand.reset();
    currentCommand.generateJointSpeedTrajectory(q, SPEED, tv, t_accel,
td);
    avisoComandoTerminado = false;
  }
  else if( (cmd.startsWith("SpeedL")) & (nvars == 7) ) {
    double pos_rot_0[5];
    Eigen::Matrix4d T = cinDir(q[0],q[1],q[2],q[3],q[4]);
    tform2eul(T, pos_rot_0[0], pos_rot_0[1], pos_rot_0[2],
pos_rot_0[3], pos_rot_0[4]);

    double SPEED[5] = {vars[0], vars[1], vars[2], vars[3], vars[4]};
    double tv = vars[5];
    double t_accel = vars[6];

    currentCommand.reset();
    currentCommand.generateCartSpeedTrajectory(pos_rot_0, SPEED, tv,
t_accel, td);
    avisoComandoTerminado = false;
  }
  else {Serial.print("Comando no reconocido, nº de variables: "),
Serial.println(nvars);}
}

void controlledStop() {
  // Lectura posicion
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();
```

```
    // Obtención referencia
    for (int i = 0; i < 5; i++) {
      qv_ref[i] = sign(qv[i])*max(abs(q[i])-STOP_ACCEL,0);
      q_ref[i] = q[i] + qv_ref[i]*td;
    }

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));
}

void probeLimitSwitches() {

  controlledStop();

  digitalWrite(pinLS_pos, LOW);
  digitalWrite(pinLS_neg, LOW);

  digitalWrite(pinLS_pos, HIGH);
  LimitSwitch_Eje5[0] = !digitalRead(pinLS_j1);
  LimitSwitch_Eje4[0] = !digitalRead(pinLS_j2);
  LimitSwitch_Eje5[0] = !digitalRead(pinLS_j3);
  LimitSwitch_Eje4[0] = !digitalRead(pinLS_j4);
  LimitSwitch_Eje5[0] = !digitalRead(pinLS_j5);
  digitalWrite(pinLS_pos, LOW);

  digitalWrite(pinLS_neg, HIGH);
  LimitSwitch_Eje5[1] = !digitalRead(pinLS_j1);
  LimitSwitch_Eje4[1] = !digitalRead(pinLS_j2);
  LimitSwitch_Eje5[1] = !digitalRead(pinLS_j3);
  LimitSwitch_Eje4[1] = !digitalRead(pinLS_j4);
  LimitSwitch_Eje5[1] = !digitalRead(pinLS_j5);
  digitalWrite(pinLS_neg, LOW);
}
```

```
void homing() {

  controlledStop();

  // Articulacion 1

  currentCommand.reset();

  double SPEED[5] = {10,0,0,0,0};
  currentCommand.generateJointSpeedTrajectory(q, SPEED, 30, 0.1, td)

  while(digitalRead(pinLS_j1)) {
    if (tiempo1 > tiempo2 + td * 1000) {
    // Lectura posicion
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();

    // Obtención referencia
    getReferences();

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));
  }

  currentCommand.reset();

  double SPEED[5] = {-2,0,0,0,0};
  currentCommand.generateJointSpeedTrajectory(q, SPEED, 30, 0.1, td)

  while(digitalRead(pinLS_j1)) {
    if (tiempo1 > tiempo2 + td * 1000) {
    // Lectura posicion
```

```
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();

    // Obtención referencia
    getReferences();

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));
  }

  // Articulacion 2

  currentCommand.reset();

  double SPEED[5] = {0,10,0,0,0};
  currentCommand.generateJointSpeedTrajectory(q, SPEED, 30, 0.1, td)

  while(digitalRead(pinLS_j2)) {
    if (tiempo1 > tiempo2 + td * 1000) {
    // Lectura posicion
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();

    // Obtención referencia
    getReferences();

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
```

```
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));
  }

  currentCommand.reset();

  double SPEED[5] = {0,-2,0,0,0};
  currentCommand.generateJointSpeedTrajectory(q, SPEED, 30, 0.1, td)

  while(digitalRead(pinLS_j2)) {
    if (tiempo1 > tiempo2 + td * 1000) {
    // Lectura posicion
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();

    // Obtención referencia
    getReferences();

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));
  }

  // Articulacion 3
```

```
  currentCommand.reset();

  double SPEED[5] = {0,0,10,0,0};
  currentCommand.generateJointSpeedTrajectory(q, SPEED, 30, 0.1, td)

  while(digitalRead(pinLS_j3)) {
    if (tiempo1 > tiempo2 + td * 1000) {
    // Lectura posicion
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();

    // Obtención referencia
    getReferences();

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));
  }

  currentCommand.reset();

  double SPEED[5] = {0,0,-2,0,0};
  currentCommand.generateJointSpeedTrajectory(q, SPEED, 30, 0.1, td)

  while(digitalRead(pinLS_j3)) {
    if (tiempo1 > tiempo2 + td * 1000) {
    // Lectura posicion
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();

    // Obtención referencia
```

```
    getReferences();

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));
  }

  // Articulacion 4

  currentCommand.reset();

  double SPEED[5] = {0,0,0,10,0};
  currentCommand.generateJointSpeedTrajectory(q, SPEED, 30, 0.1, td)

  while(digitalRead(pinLS_j4)) {
    if (tiempo1 > tiempo2 + td * 1000) {
    // Lectura posicion
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();

    // Obtención referencia
    getReferences();

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
```

```
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));
  }

  currentCommand.reset();

  double SPEED[5] = {0,0,0,-2,0};
  currentCommand.generateJointSpeedTrajectory(q, SPEED, 30, 0.1, td)

  while(digitalRead(pinLS_j4)) {
    if (tiempo1 > tiempo2 + td * 1000) {
    // Lectura posicion
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();

    // Obtención referencia
    getReferences();

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));
  }

  // Articulacion 5

  currentCommand.reset();

  double SPEED[5] = {0,0,0,0,10};
  currentCommand.generateJointSpeedTrajectory(q, SPEED, 30, 0.1, td)

  while(digitalRead(pinLS_j5)) {
```

```
    if (tiempo1 > tiempo2 + td * 1000) {
    // Lectura posicion
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();

    // Obtención referencia
    getReferences();

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));
  }

  currentCommand.reset();

  double SPEED[5] = {0,0,0,0,-2};
  currentCommand.generateJointSpeedTrajectory(q, SPEED, 30, 0.1, td)

  while(digitalRead(pinLS_j5)) {
    if (tiempo1 > tiempo2 + td * 1000) {
    // Lectura posicion
    readJointPositions();

    // Calculo velocidades
    getJointVelocities();

    // Obtención referencia
    getReferences();

    // Calculo par (PD)
    calculateTorques();

    // Escritura motores
```

```
    analogWrite(pinPWM_M1, constrain(map(tau[0] * gearRatio_M1 / Kt_M1,
-nomCurr_M1, nomCurr_M1, 0, 255), 0, 255));
    analogWrite(pinPWM_M2, constrain(map(tau[1] * gearRatio_M2 / Kt_M2,
-nomCurr_M2, nomCurr_M2, 0, 255), 0, 255));
    analogWrite(pinPWM_M3, constrain(map(tau[2] * gearRatio_M3 / Kt_M3,
-nomCurr_M3, nomCurr_M3, 0, 255), 0, 255));
    analogWrite(pinPWM_M4, constrain(map(tau[3] * gearRatio_M4 / Kt_M4,
-nomCurr_M4, nomCurr_M4, 0, 255), 0, 255));
    analogWrite(pinPWM_M5, constrain(map(tau[4] * gearRatio_M5 / Kt_M5,
-nomCurr_M5, nomCurr_M5, 0, 255), 0, 255));
  }

}
```

- Librería de cinemática del brazo

```
#ifndef RobotArmKinematics_h
#define RobotArmKinematics_h

#include <ArduinoEigen.h>

Eigen::Matrix4f cinDir(float q1, float q2, float q3, float q4, float
q5);
void cinInv(double x_wrist, double z_wrist, double phi, double theta,
double psi, double* q);
void tform2eul(Eigen::Matrix4d T, double &x_wrist, double &z_wrist,
double &phi, double &theta, double &psi);
Eigen::MatrixXd jacobian(double q1, double q2, double q3, double q4,
double q5);

#endif
```

```
#include <ArduinoEigen.h>
#include "RobotArmKinematics.h"

#define L1 300
#define L2 260
#define L3 100

#define PI 3.14159265358979323846264338327795
```

```
Eigen::Matrix4d cinDir(double q1, double q2, double q3, double q4,
double q5) {
  Eigen::Matrix4d T_03;
  Eigen::Matrix4d T_45;
  Eigen::Matrix4d T;

  T_03 << -sin(q1+q2)*cos(q3), -cos(q1+q2),  sin(q1+q2)*sin(q3),
L1*cos(q1)+L2*cos(q1+q2),
          -sin(q3)           ,  0         , -cos(q3)           , 0,
           cos(q1+q2)*cos(q3), -sin(q1+q2), -cos(q1+q2)*sin(q3),
L1*sin(q1)+L2*sin(q1+q2),
           0                 ,  0         ,  0                 , 1;

  T_45 << cos(q4)*cos(q5), -cos(q4)*sin(q5),  sin(q4),  L3*sin(q4),
          sin(q4)*cos(q5), -sin(q4)*sin(q5), -cos(q4), -L3*cos(q4),
          sin(q5)        ,  cos(q5)        ,  0      ,  0,
          0              ,  0              ,  0      ,  1;

  T = T_03*T_45;
  return T;
}

void cinInv(double x_wrist, double z_wrist, double phi, double theta,
double psi, double* q) {
  Eigen::Matrix3d R_02;
  Eigen::Matrix3d R_eul;
  Eigen::Matrix3d R_eul2eff;
  Eigen::Matrix3d R_eff;
  Eigen::Matrix3d R_35;

  double d = sqrt(pow(x_wrist,2) + pow(z_wrist,2));
  double gamma = atan2(z_wrist,x_wrist);
  double alpha = acos((pow(d,2) + pow(L1,2) - pow(L2,2))/(2*d*L1));
  double beta = acos((pow(L1,2) + pow(L2,2) - pow(d,2))/(2*L1*L2));

  double q1 = gamma - alpha;
  double q2 = PI - beta;

  R_02 << -sin(q1+q2), 0, cos(q1+q2),
           cos(q1+q2), 0, sin(q1+q2),
           0         , 1, 0;

  R_eul << cos(phi)*cos(theta), cos(phi)*sin(theta)*sin(psi) -
sin(phi)*cos(psi), sin(phi)*sin(psi) + cos(phi)*sin(theta)*cos(psi),
```

```
          sin(phi)*cos(theta), cos(phi)*cos(psi) +
sin(phi)*sin(theta)*sin(psi), sin(phi)*sin(theta)*cos(psi) -
cos(phi)*sin(psi),
         -sin(theta)         ,
cos(theta)*sin(psi)                              , cos(theta)*cos(psi);

  R_eul2eff << 0,  0, 1,
               0, -1, 0,
               1,  0, 0;

  R_eff << R_eul * R_eul2eff;

  R_35 = R_02.transpose()*R_eff;

  double q4 = -atan2(sqrt(1-pow(R_35(2,2),2)),R_35(2,2));

  double q3;
  double q5;

  if (abs(R_35(2,2)) != 1) {
    q3 = atan2(-R_35(1,2),-R_35(0,2));
    q5 = atan2(-R_35(2,1),R_35(2,0));
  }
  else {
    double diff_q3_q5 = atan2(R_35(1,0),R_35(0,0));
    q3 = 0;
    q5 = q3 + diff_q3_q5;
  }

  q[0] = q1;
  q[1] = q2;
  q[2] = q3;
  q[3] = q4;
  q[4] = q5;
}


void tform2eul(Eigen::Matrix4d T, double &x_wrist, double &z_wrist,
double &phi, double &theta, double &psi) {
  Eigen::Matrix3d R_eff;
  Eigen::Matrix3d R_eff2eul;
  Eigen::Matrix3d R_eul;
  double diff_phi_psi;

  x_wrist = T(0,3) - L3*T(0,2);
  z_wrist = T(2,3) - L3*T(2,2);
```

```
  R_eff << T.block<3,3>(0,0);

  R_eff2eul << 0,  0, 1,
               0, -1, 0,
               1,  0, 0;

  R_eff << R_eul * R_eff2eul;

  theta = asin(-R_eul(3,1));

  if (abs(R_eul(2,0)) != 1) {
    phi = atan2(R_eul(1,0),R_eul(0,0));
    psi = atan2(R_eul(2,1),R_eul(2,2));
  }
  else {
    diff_phi_psi = atan2(-R_eul(0,1),R_eul(1,1));
    phi = 0;
    psi = diff_phi_psi;
  }
}

Eigen::MatrixXd jacobian(double q1, double q2, double q3, double q4,
double q5){
  Eigen::MatrixXd J(5,5);

  J << -L1*sin(q1)-L2*sin(q1+q2), -
L2*sin(q1+q2),  0,           0,                   0,
        L1*cos(q1)+L2*cos(q1+q2),  L2*cos(q1+q2),  0,           0,
             0,
        0,                         0,              cos(q1+q2),  sin(q1+
q2)*sin(q3),  cos(q1+q2)*cos(q4)-sin(q1+q2)*cos(q3)*sin(q4),
       -1,                        -1,              0,          -
cos(q3),            -sin(q3)*sin(q4),
        0,                         0,              sin(q1+q2), -
cos(q1+q2)*sin(q3),  cos(q1+q2)*cos(q3)*sin(q4)+sin(q1+q2)*cos(q4);

  return J;
}
```

- Librería de generación de trayectorias

```
#ifndef Trajectory_h
#define Trajectory_h

void jointTrajectory( double q0[], double qf[], double tv, double td,
double qd0, double qd1, double q_traj[5][4096], double
qv_traj[5][4096]);
void trapezoidTrajectoryStep(double q0, double qf, double SPEED, double
ACCEL, double td, double t_accel, double t_const, int step, double
&q_ref, double &qv_ref);
void cartesianTrajectory(double pos_rot_0[5], double pos_rot_f[5],
double POS_SPEED, double POS_ACCEL, double ANG_SPEED, double ANG_ACCEL,
double td, double q_traj[5][4096], double qv_traj[5][4096],  int
&steps);
void jointVelTrajectroy(double q0[5], double SPEED[5], double tv,
double t_accel, double td, double q_traj[5][4096], double
qv_traj[5][4096], int &steps);
void cartesianVelTrajectroy(double pos_rot_0[5], double SPEED[5],
double tv, double t_accel, double td, double q_traj[5][4096], double
qv_traj[5][4096], int &steps);

#endif
```

```
#include "Arduino.h"
#include "Trajectory.h"
#include "RobotArmKinematics.h"

#define sign(x) ((x) < 0 ? -1 : ((x) > 0 ? 1 : 0))

void jointTrajectory( double q0[5], double qf[5], double tv, double td,
double qd0, double qd1, double q_traj[5][4096], double
qv_traj[5][4096]){
  float t = 0;
  int i = 0;
  int j = 0;
  // e=e+1;
  for (i = 0; i <= 4; ++i) {
    // Serial.print("----------------i=");
    // Serial.println(i);
    double A = 6 * (qf[i] - q0[i]) * (1 / pow(tv, 5)) - 3 * (qd1 + qd0)
* (1 / pow(tv, 4));
```

```
    double B = -15 * (qf[i] - q0[i]) * (1 / pow(tv, 4)) + (8 * qd0 * (1
/ pow(tv, 3)) + 7 * qd1 * (1 / pow(tv, 3)));
    double C = 10 * (qf[i] - q0[i]) * (1 / pow(tv, 3)) - (6 * qd0 * (1
/ pow(tv, 2)) - 4 * qd1 * (1 / pow(tv, 2)));
    double D = 0;
    double E = qd0;
    double F = q0[i];
    //Serial.println(A,8);Serial.println(B,6);Serial.println(C,6);Seria
l.println(F,6);

    for (t = 0; t <= tv; t = t + td) {
      q_traj[i][j] = F + E * t + D * pow(t, 2) + C * pow(t, 3) + B *
pow(t, 4) + A * pow(t, 5);
      qv_traj[i][j] = E + 2* D * t + 3 *C * pow(t, 2) + 4* B * pow(t,
3) + 5 * A * pow(t, 4);

      // Serial.print("q[");
      // Serial.print(i); Serial.print("]"); Serial.print("[");
      // Serial.print(j);
      // Serial.println("]");
      // Serial.println(q_traj[i][j]);
      j = j + 1 ;
      //Q(i,l)=q;ok

    }
    j = 0;
  }
}

void trapezoidTrajectoryStep(double q0, double qf, double SPEED, double
ACCEL, double td, double t_accel, double t_const, int step, double
&q_ref, double &qv_ref) {
  double speed = SPEED * sign(qf-q0);
  double accel = ACCEL * sign(qf-q0);

  if (step <= 0) {
    q_ref = q0;
    qv_ref = 0;
  }
  else if (step <= t_accel/td) {
    q_ref = q0 + accel/2 * pow((step*td),2);
    qv_ref = accel * (step*td);
  }
  else if (step <= (t_const + t_accel)/td) {
    q_ref = q0 + accel/2 * pow(t_accel,2) + speed * ((step*td) -
t_accel);
```

```
    qv_ref = speed;
  }
  else if (step <= (t_const+2*t_accel)/td) {
    q_ref = qf - accel/2 * pow(((step)*td) - (t_const+2*t_accel),2);
    qv_ref = accel * ((t_const+2*t_accel) - ((step+1)*td));
  }
  else {
    q_ref = qf;
    qv_ref = 0;
  }
}

void cartesianTrajectory(double pos_rot_0[5], double pos_rot_f[5],
double POS_SPEED, double POS_ACCEL, double ANG_SPEED, double ANG_ACCEL,
double td, double q_traj[5][4096], double qv_traj[5][4096], int &steps)
{
  double delta_X = pos_rot_f[0]-pos_rot_0[0];
  double delta_Z = pos_rot_f[1]-pos_rot_0[1];
  double delta_phi = pos_rot_f[2]-pos_rot_0[2];
  double delta_theta = pos_rot_f[3]-pos_rot_0[3];
  double delta_psi = pos_rot_f[4]-pos_rot_0[4];

  double S = sqrt(pow(pos_rot_f[0]-pos_rot_0[0],2) + pow(pos_rot_f[1]-
pos_rot_0[1],2));

  double max_ang_diff = max(max(delta_phi,delta_theta),delta_psi);

  double t_accel = max(POS_SPEED/POS_ACCEL, ANG_SPEED/ANG_ACCEL);
  double t_const = max((S - POS_ACCEL*pow(t_accel,2))/POS_SPEED,
(max_ang_diff - ANG_ACCEL*pow(t_accel,2))/ANG_SPEED);

  double SPEED_X = delta_X / (t_const+t_accel);
  double ACCEL_X = ( SPEED_X > 0 ) ? SPEED_X / t_accel : 1;

  double SPEED_Z = delta_Z / (t_const+t_accel);
  double ACCEL_Z = ( SPEED_Z > 0 ) ? SPEED_Z / t_accel : 1;

  double ANG_SPEED_PHI = delta_phi / (t_const+t_accel);
  double ANG_ACCEL_PHI = ( ANG_SPEED_PHI > 0 ) ? ANG_SPEED_PHI /
t_accel : 1;

  double ANG_SPEED_THETA = delta_theta / (t_const+t_accel);
  double ANG_ACCEL_THETA = ( ANG_SPEED_THETA > 0 ) ? ANG_SPEED_THETA /
t_accel : 1;

  double ANG_SPEED_PSI = delta_psi / (t_const+t_accel);
```

```
  double ANG_ACCEL_PSI = ( ANG_SPEED_PSI > 0 ) ? ANG_SPEED_PSI /
t_accel : 1;

  Serial.print("aceleracion angular de psi: ");
Serial.println(ANG_ACCEL_PSI);

  if (t_const < 0) {
    t_accel =
max(max(max(max(sqrt(delta_X/ACCEL_X),sqrt(delta_Z/ACCEL_Z)),sqrt(delta
_phi/ANG_ACCEL_PHI)),sqrt(delta_theta/ANG_ACCEL_THETA)),sqrt(delta_psi/
ANG_ACCEL_PSI));
    t_const = 0;
  }

  Serial.print("tiempo de aceleracion: "); Serial.println(t_accel);
  Serial.print("tiempo de movimiento uniforme: ");
Serial.println(t_const);

  steps = ceil((t_const + 2*t_accel)/td)+1;

  Serial.print("Nº de pasos: "); Serial.println(steps);

  double q_ref[5];

  cinInv(pos_rot_0[0], pos_rot_0[1], pos_rot_0[2], pos_rot_0[3],
pos_rot_0[4], q_ref);

  q_traj[0][0] = q_ref[0];
  q_traj[1][0] = q_ref[1];
  q_traj[2][0] = q_ref[2];
  q_traj[3][0] = q_ref[3];
  q_traj[4][0] = q_ref[4];

  qv_traj[0][0] = 0;
  qv_traj[1][0] = 0;
  qv_traj[2][0] = 0;
  qv_traj[3][0] = 0;
  qv_traj[4][0] = 0;

  double x_ref;
  double z_ref;
  double phi_ref;
  double theta_ref;
  double psi_ref;

  double x_ref_v;
```

```
  double z_ref_v;
  double phi_ref_v;
  double theta_ref_v;
  double psi_ref_v;

  for (int i = 1; i < steps; ++i) {
    trapezoidTrajectoryStep(pos_rot_0[0], pos_rot_f[0], SPEED_X,
ACCEL_X, td, t_accel, t_const, i, x_ref, x_ref_v);
    trapezoidTrajectoryStep(pos_rot_0[1], pos_rot_f[1], SPEED_Z,
ACCEL_Z, td, t_accel, t_const, i, z_ref, z_ref_v);
    trapezoidTrajectoryStep(pos_rot_0[2], pos_rot_f[2], ANG_SPEED_PHI,
ANG_ACCEL_PHI, td, t_accel, t_const, i, phi_ref, phi_ref_v);
    trapezoidTrajectoryStep(pos_rot_0[3], pos_rot_f[3],
ANG_SPEED_THETA, ANG_ACCEL_THETA, td, t_accel, t_const, i, theta_ref,
theta_ref_v);
    trapezoidTrajectoryStep(pos_rot_0[4], pos_rot_f[4], ANG_SPEED_PSI,
ANG_ACCEL_PSI, td, t_accel, t_const, i, psi_ref, psi_ref_v);

    cinInv(x_ref, z_ref, phi_ref, theta_ref, psi_ref, q_ref);

    q_traj[0][i] = q_ref[0];
    q_traj[1][i] = q_ref[1];
    q_traj[2][i] = q_ref[2];
    q_traj[3][i] = q_ref[3];
    q_traj[4][i] = q_ref[4];

    Eigen::MatrixXd J(5,5);
    J = jacobian(q_ref[0], q_ref[1], q_ref[2], q_ref[3], q_ref[4]);

    Eigen::VectorXd cart_v(5);
    cart_v << x_ref_v, z_ref_v, phi_ref_v, theta_ref_v, psi_ref_v;

    Eigen::VectorXd qv_ref = J.inverse() * cart_v;

    qv_traj[0][i] = qv_ref[0];
    qv_traj[1][i] = qv_ref[1];
    qv_traj[2][i] = qv_ref[2];
    qv_traj[3][i] = qv_ref[3];
    qv_traj[4][i] = qv_ref[4];
  }
}

void jointVelTrajectroy(double q0[5], double SPEED[5], double tv,
double t_accel, double td, double q_traj[5][4096], double
qv_traj[5][4096], int &steps) {
  if (t_accel > tv/2) {
```

```
    t_accel = tv/2;
  }
  double t_const = min(tv - 2*t_accel,0);

  double qf[5];
  qf[0] = q0[0] + SPEED[0] * (t_accel + t_const);
  qf[1] = q0[1] + SPEED[1] * (t_accel + t_const);
  qf[2] = q0[2] + SPEED[2] * (t_accel + t_const);
  qf[3] = q0[3] + SPEED[3] * (t_accel + t_const);
  qf[4] = q0[4] + SPEED[4] * (t_accel + t_const);

  double ACCEL[5];
  ACCEL[0] = SPEED[0] / t_accel;
  ACCEL[1] = SPEED[1] / t_accel;
  ACCEL[2] = SPEED[2] / t_accel;
  ACCEL[3] = SPEED[3] / t_accel;
  ACCEL[4] = SPEED[4] / t_accel;


  steps = ceil(tv/td)+1;

  for (int i = 1; i < steps; ++i) {
    trapezoidTrajectoryStep(q0[0], qf[0], SPEED[0], ACCEL[0], td,
t_accel, t_const, i, q_traj[0][i], qv_traj[0][i]);
    trapezoidTrajectoryStep(q0[1], qf[1], SPEED[1], ACCEL[1], td,
t_accel, t_const, i, q_traj[1][i], qv_traj[1][i]);
    trapezoidTrajectoryStep(q0[2], qf[2], SPEED[2], ACCEL[2], td,
t_accel, t_const, i, q_traj[2][i], qv_traj[2][i]);
    trapezoidTrajectoryStep(q0[3], qf[3], SPEED[3], ACCEL[3], td,
t_accel, t_const, i, q_traj[3][i], qv_traj[3][i]);
    trapezoidTrajectoryStep(q0[4], qf[4], SPEED[4], ACCEL[4], td,
t_accel, t_const, i, q_traj[4][i], qv_traj[4][i]);
  }
}

void cartesianVelTrajectroy(double pos_rot_0[5], double SPEED[5],
double tv, double t_accel, double td, double q_traj[5][4096], double
qv_traj[5][4096], int &steps) {
  if (t_accel > tv/2) {
    t_accel = tv/2;
  }
  double t_const = min(tv - 2*t_accel,0);

  double pos_rot_f[5];
  pos_rot_f[0] = pos_rot_0[0] + SPEED[0] * (t_accel + t_const);
  pos_rot_f[1] = pos_rot_0[1] + SPEED[1] * (t_accel + t_const);
```

```
  pos_rot_f[2] = pos_rot_0[2] + SPEED[2] * (t_accel + t_const);
  pos_rot_f[3] = pos_rot_0[3] + SPEED[3] * (t_accel + t_const);
  pos_rot_f[4] = pos_rot_0[4] + SPEED[4] * (t_accel + t_const);

  double ACCEL[5];
  ACCEL[0] = SPEED[0] / t_accel;
  ACCEL[1] = SPEED[1] / t_accel;
  ACCEL[2] = SPEED[2] / t_accel;
  ACCEL[3] = SPEED[3] / t_accel;
  ACCEL[4] = SPEED[4] / t_accel;

  steps = ceil(tv/td)+1;

  double q_ref[5];

  cinInv(pos_rot_0[0], pos_rot_0[1], pos_rot_0[2], pos_rot_0[3],
pos_rot_0[4], q_ref);

  q_traj[0][0] = q_ref[0];
  q_traj[1][0] = q_ref[1];
  q_traj[2][0] = q_ref[2];
  q_traj[3][0] = q_ref[3];
  q_traj[4][0] = q_ref[4];

  qv_traj[0][0] = 0;
  qv_traj[1][0] = 0;
  qv_traj[2][0] = 0;
  qv_traj[3][0] = 0;
  qv_traj[4][0] = 0;

  double x_ref;
  double z_ref;
  double phi_ref;
  double theta_ref;
  double psi_ref;

  double x_ref_v;
  double z_ref_v;
  double phi_ref_v;
  double theta_ref_v;
  double psi_ref_v;

  for (int i = 1; i < steps; ++i) {
    trapezoidTrajectoryStep(pos_rot_0[0], pos_rot_f[0], SPEED[0],
ACCEL[0], td, t_accel, t_const, i, x_ref, x_ref_v);
```

```
    trapezoidTrajectoryStep(pos_rot_0[1], pos_rot_f[1], SPEED[1],
ACCEL[1], td, t_accel, t_const, i, z_ref, z_ref_v);
    trapezoidTrajectoryStep(pos_rot_0[2], pos_rot_f[2], SPEED[2],
ACCEL[2], td, t_accel, t_const, i, phi_ref, phi_ref_v);
    trapezoidTrajectoryStep(pos_rot_0[3], pos_rot_f[3], SPEED[3],
ACCEL[3], td, t_accel, t_const, i, theta_ref, theta_ref_v);
    trapezoidTrajectoryStep(pos_rot_0[4], pos_rot_f[4], SPEED[4],
ACCEL[4], td, t_accel, t_const, i, psi_ref, psi_ref_v);

    cinInv(x_ref, z_ref, phi_ref, theta_ref, psi_ref, q_ref);

    q_traj[0][i] = q_ref[0];
    q_traj[1][i] = q_ref[1];
    q_traj[2][i] = q_ref[2];
    q_traj[3][i] = q_ref[3];
    q_traj[4][i] = q_ref[4];

    Eigen::MatrixXd J(5,5);
    J = jacobian(q_ref[0], q_ref[1], q_ref[2], q_ref[3], q_ref[4]);

    Eigen::VectorXd cart_v(5);
    cart_v << x_ref_v, z_ref_v, phi_ref_v, theta_ref_v, psi_ref_v;

    Eigen::VectorXd qv_ref = J.inverse() * cart_v;

    qv_traj[0][i] = qv_ref[0];
    qv_traj[1][i] = qv_ref[1];
    qv_traj[2][i] = qv_ref[2];
    qv_traj[3][i] = qv_ref[3];
    qv_traj[4][i] = qv_ref[4];
  }
}
```

- Librería de órdenes del brazo

```
#ifndef Command_h
#define Command_h

enum CommandType {NONE, POS, VEL};

class CommandTrajectory
{
  public:
```

```
    CommandTrajectory();
    void generateJointPosTrajectory(double q0[5], double qf[5], double
t, double td);
    void generateCartPoseTrajectory(double pos_rot_0[5], double
pos_rot_f[5], double POS_SPEED, double POS_ACCEL, double ANG_SPEED,
double ANG_ACCEL, double td);
    void generateCartJointTrajectory(double pos_rot_0[5], double
pos_rot_f[5], double t, double td);
    void generateJointSpeedTrajectory(double q0[5], double SPEED[5],
double tv, double t_accel, double td);
    void generateCartSpeedTrajectory(double pos_rot_0[5], double
SPEED[5], double tv, double t_accel, double td);
    void update();
    void get_q_ref(double* q);
    void get_qv_ref(double* qv);
    bool isFinished();
    void reset();
  private:
    CommandType _commandType;
    int _steps;
    int _currentStep = 0;
    double _qv_ref[5];
    double _q_ref[5];
    double _q_traj[5][4096];
    double _qv_traj[5][4096];
    double _q0[5];
    double _qf[5];
    bool _flagDone = false;
};


#endif
```

```
#include "Arduino.h"
#include "Command.h"
#include "Trajectory.h"
#include "RobotArmKinematics.h"


CommandTrajectory::CommandTrajectory() {
  _commandType = NONE;
  _steps = 0;
  _currentStep = 0;
}
```

```
void CommandTrajectory::generateJointPosTrajectory(double q0[5], double
qf[5], double t, double td) {
  _commandType = POS;
  for (int i = 0; i < 5 ; i++){
    _q0[i] = q0[i];
    _qf[i] = qf[i];
  }

  _steps = ceil(t/td);
  jointTrajectory(q0, qf, t, td, 0, 0, _q_traj, _qv_traj);
}

void CommandTrajectory::generateCartPoseTrajectory(double pos_rot_0[5],
double pos_rot_f[5], double POS_SPEED, double POS_ACCEL, double
ANG_SPEED, double ANG_ACCEL, double td) {
  _commandType = POS;

  cartesianTrajectory(pos_rot_0, pos_rot_f, POS_SPEED, POS_ACCEL,
ANG_SPEED, ANG_ACCEL, td, _q_traj, _qv_traj, _steps);
}

void CommandTrajectory::generateCartJointTrajectory(double
pos_rot_0[5], double pos_rot_f[5], double t, double td) {
  _commandType = POS;
  cinInv(pos_rot_0[0], pos_rot_0[1], pos_rot_0[2], pos_rot_0[3],
pos_rot_0[4], _q0);
  cinInv(pos_rot_f[0], pos_rot_f[1], pos_rot_f[2], pos_rot_f[3],
pos_rot_f[4], _qf);

  _steps = ceil(t/td);
  jointTrajectory(_q0, _qf, t, td, 0, 0, _q_traj, _qv_traj);
}

void CommandTrajectory::generateJointSpeedTrajectory(double q0[5],
double SPEED[5], double tv, double t_accel, double td) {
  _commandType = VEL;
  jointVelTrajectroy(q0, SPEED, tv, t_accel, td, _q_traj, _qv_traj,
_steps);

  _qf[0] = _q_traj[0][_steps];
  _qf[1] = _q_traj[1][_steps];
  _qf[2] = _q_traj[2][_steps];
  _qf[3] = _q_traj[3][_steps];
  _qf[4] = _q_traj[4][_steps];
}
```

```
void CommandTrajectory::generateCartSpeedTrajectory(double
pos_rot_0[5], double SPEED[5], double tv, double t_accel, double td) {
  _commandType = VEL;
  cinInv(pos_rot_0[0], pos_rot_0[1], pos_rot_0[2], pos_rot_0[3],
pos_rot_0[4], _q0);

  cartesianVelTrajectroy(pos_rot_0, SPEED, tv, t_accel, td, _q_traj,
_qv_traj, _steps);

  _qf[0] = _q_traj[0][_steps-1];
  _qf[1] = _q_traj[1][_steps-1];
  _qf[2] = _q_traj[2][_steps-1];
  _qf[3] = _q_traj[3][_steps-1];
  _qf[4] = _q_traj[4][_steps-1];
}

void CommandTrajectory::update() {
  if (_commandType != NONE) {
    if (_currentStep < _steps) {
      for (int i = 0; i < 5; i++) {
        _q_ref[i] = _q_traj[i][_currentStep];
        _qv_ref[i] = _qv_traj[i][_currentStep];
      }
      _currentStep++;
    }
    else {
      _flagDone = true;
      for (int i = 0; i < 5; i++) {
        _q_ref[i] = _qf[i];
        _qv_ref[i] = 0;
      }
    }
  }
}

void CommandTrajectory::get_q_ref(double* q) {
  for (int i = 0; i < 5; i++) {
    q[i] = _q_ref[i];
  }
}

void CommandTrajectory::get_qv_ref(double* qv) {
  for (int i = 0; i < 5; i++) {
    qv[i] = _qv_ref[i];
  }
}
```

```
bool CommandTrajectory::isFinished() {
  if (_commandType == NONE) {
    return true;
  }
  return _flagDone;
}

void CommandTrajectory::reset() {
  _commandType = NONE;
  _steps = 0;
  _currentStep = 0;
}
```

- Librería de dinámica del brazo

```
#ifndef RobotArmDynamics_h
#define RobotArmDynamics_h

#include <ArduinoEigen.h>

void gravload(double q[5],  double* tau_g);

#endif
```

```
#include <ArduinoEigen.h>
#include "RobotArmDynamics.h"

#define PI 3.14159265358979323846264338327795

void gravload(double q[5],  double* tau_g) {
  Eigen::Matrix4d T0, T1, T2, T3, T4, T5;
  Eigen::Matrix3d R0, R01, R02, R03, R04, R05;

  double m_CDM;
```

```
Eigen::Vector3d p_CDM, F;
Eigen::Vector4d p_CDM_temp;

Eigen::Vector3d tau_1, tau_2, tau_3, tau_4, tau_5;

Eigen::Vector3d g(0, 0, 9.81);

double m_eff = 0;
Eigen::Vector3d p_j5_eff(0, 0, 0);

double m_j5 = 0.015;
Eigen::Vector3d p_j5_CDM5(0, 0, -0.0287);

double m_j4 = 0.028;
Eigen::Vector3d p_j4_CDM4(0, 0, 0.0246);

double m_j3 = 0.548;
Eigen::Vector3d p_j3_CDM3(0, -0.1144, 0);

double m_j2 = 0.051;
Eigen::Vector3d p_j2_CDM2(0, 0, 0.0417);

double m_j1 = 1.3117;
Eigen::Vector3d p_j1_CDM1(-0.3048, -0.0027, -0.001);


T0 << 1, 0, 0, 0,
      0, 0,-1, 0,
      0, 1, 0, 0,
      0, 0, 0, 1;

T1 << cos(q[0]), -sin(q[0]), 0, 0.3*cos(q[0]),
      sin(q[0]),  cos(q[0]), 0, 0.3*sin(q[0]),
      0,          0,         1, 0,
      0,          0,         0, 1;

T2 << -sin(q[1]), 0, cos(q[1]), 0,
      cos(q[1]), 0, sin(q[1]), 0,
      0,         1, 0,         0,
      0,         0, 0,         1;

T3 << cos(q[2]),  0, -sin(q[2]), 0,
      sin(q[2]),  0,  cos(q[2]), 0,
      0,         -1,  0,         0.26,
      0,          0,  0,         1;
```

```
T4 << cos(q[3]),  0,  sin(q[3]), 0,
      sin(q[3]),  0, -cos(q[3]), 0,
      0,          1,  0,         0,
      0,          0,  0,         1;

T5 << cos(q[4]), -sin(q[4]), 0, 0,
      sin(q[4]),  cos(q[4]), 0, 0,
      0,          0,         1, 0.06,
      0,          0,         0, 1;


R0 = T0.topLeftCorner(3,3);
R01 = R0 * T1.topLeftCorner(3,3);
R02 = R01 * T2.topLeftCorner(3,3);
R03 = R02 * T3.topLeftCorner(3,3);
R04 = R03 * T4.topLeftCorner(3,3);


p_CDM = (m_eff * p_j5_eff + m_j5 * p_j5_CDM5) / (m_eff + m_j5);
m_CDM = m_eff + m_j5;

p_CDM_temp << p_CDM, 1;
p_CDM_temp = T5 * p_CDM_temp;
p_CDM = p_CDM_temp.head(3);
F = m_CDM * g.transpose() * R04;

tau_5 = p_CDM.cross(F);


p_CDM = (m_CDM * p_CDM + m_j4 * p_j4_CDM4) / (m_CDM + m_j4);
m_CDM = m_eff + m_j4;

p_CDM_temp << p_CDM, 1;
p_CDM_temp = T4 * p_CDM_temp;
p_CDM = p_CDM_temp.head(3);
F = m_CDM * g.transpose() * R03;

tau_4 = p_CDM.cross(F);


p_CDM = (m_CDM * p_CDM + m_j3 * p_j3_CDM3) / (m_CDM + m_j3);
m_CDM = m_eff + m_j3;

p_CDM_temp << p_CDM, 1;
p_CDM_temp = T3 * p_CDM_temp;
p_CDM = p_CDM_temp.head(3);
F = m_CDM * g.transpose() * R02;
```

```
  tau_3 = p_CDM.cross(F);


  p_CDM = (m_CDM * p_CDM + m_j2 * p_j2_CDM2) / (m_CDM + m_j2);
  m_CDM = m_eff + m_j2;

  p_CDM_temp << p_CDM, 1;
  p_CDM_temp = T2 * p_CDM_temp;
  p_CDM = p_CDM_temp.head(3);
  F = m_CDM * g.transpose() * R01;

  tau_2 = p_CDM.cross(F);


  p_CDM = (m_CDM * p_CDM + m_j1 * p_j1_CDM1) / (m_CDM + m_j1);
  m_CDM = m_eff + m_j1;

  p_CDM_temp << p_CDM, 1;
  p_CDM_temp = T1 * p_CDM_temp;
  p_CDM = p_CDM_temp.head(3);
  F = m_CDM * g.transpose() * R0;

  tau_1 = p_CDM.cross(F);

  tau_g[0] = tau_1[2];
  tau_g[1] = tau_2[2];
  tau_g[2] = tau_3[2];
  tau_g[3] = tau_4[2];
  tau_g[4] = tau_5[2];
}
```

Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería del Diseño

PLIEGO DE CONDICIONES

Trabajo de Fin de Master

Master en Ingeniería Mecatrónica

1. Objeto

Este documento tiene como objetivo detallar los requisitos técnicos y condiciones para el montaje y la operación del prototipo de brazo robot diseñado y fabricado.

2. Condiciones generales

Se detallan deberes y derechos de que asume cada parte de la relación establecida entre contratante y contratista

-La empresa contratista debe conocer el proyecto y cumplir con los términos establecidos con la empresa contratante

-La empresa contratista tiene derecho a solicitar soluciones a problemas de carácter técnico, a la empresa contratante que ocurran durante el desarrollo del proyecto y que no sean imputables a su mala ejecución

-La empresa contratante debe conocer el proyecto y cumplir con los términos establecidos con la empresa contratista

-La empresa contratante puede modificar los tiempos establecidos para cada fase del proyecto, siempre que no se violen los plazos fijados de manera contractual

- La empresa contratista es responsable de cumplir con la ley 31/1995 de prevención de riesgos laborales, deberán cumplirse las normas aplicables a tareas de fabricación y montaje, así como todo lo dispuesto en la normativa sobre trabajos de taller. Por lo tanto, se deberá ocupar a todos los trabajadores con equipos de protección individual.

2.1. Condiciones económicas

2.1.1. Mejoras y modificaciones

Las mejoras y modificaciones propuestas por la empresa contratante correrán a su cargo. Las mejoras y modificaciones propuestas por la empresa contratista no generaran un aumento del importe del presupuesto de ejecución

2.1.2. Revisión de presupuesto

El contrato es a precio alzado y no se contempla una revisión del precio ni presupuestos debido a la corta duración temporal establecida para el proyecto

2.1.3. Abono de los trabajos

El importe del proyecto será abonado por el contratante, dividido en 10 partes, y se pagaran mensualmente durante la duración estipulada del proyecto. Donde se incluirán los costes adicionales surgidos durante el desarrollo del proyecto.

2.2. Condiciones legales

2.2.1. Daños a terceros.

El contratista es responsable de los daños a terceros que pudiesen producirse durante la ejecución del proyecto, y deberá abonad los gastos que estos produjeran

2.2.2. Causas de resolución de contrato

-Por acuerdo de ambas partes

-Un retraso excesivo en la ejecución del proyecto

-Causas administrativas

2.2.3. Arbitraje y jurisdicción

En caso de litigio, ambas partes se comprometen a acudir a una institución de arbitraje, para solventar las posiciones encontradas, a cuya decisión se someten.

3. CONDICIONES TECNICAS

3.1. Componentes fabricados

Los componentes del diseño de fabricación propia se encuentran totalmente definidos en los planos de fabricación, dichos planos cumplen la normativa UNE- EN ISO 5456, y se incluyen tolerancias dimensionales para facilitar el montaje posterior.

Como tolerancia general se aplicará la indicada en la normativa ISO 2768-ck

En el documento correspondiente a la memoria se describen los métodos de fabricación apropiados para cada pieza, así como parámetros de configuración de las impresoras 3D (si aplica) y materiales a utilizar.

3.2. Componentes Estándar

En el diseño realizado son empleados componentes estándar, estos deben servirse de la garantía de funcionamiento adecuados provistas por el fabricante y cumplir la normativa de los estándares europeos para productos electrónicos y componentes mecánicos.

3.3. Tensión de trabajo

Las etapas de la potencia del robot deben estar conectados a una fuente de corriente continua que provea de una tensión mínima de 15 V y no se debe conectar nunca a más de 30 V.

La conexión del microcontrolador y las etapas de potencia al pc, se realiza por medio de un cable microSD, y debe tomarse siempre la precaución de realizarlas con la fuente de corriente continua que alimenta las etapas de

potencia desconectada, para evitar una conexión en caliente, y evitar daños al equipo. Para evitar este riesgo puede hacerse uso de un separador de tierras.

3.4. Condiciones de pruebas y puesta en funcionamiento

Dado que el proyecto presentado define el diseño y construcción de un primer prototipo las normas específicas para equipos eléctricos de asistencia médica son ignorados, sin embargo, un producto final deberá tener en cuenta y cumplir con dicha normativa

Para la operación del robot este deberá estar firmemente unido a una superficie estable con una mordaza.

Las condiciones de almacenamiento de operación del brazo robótico y sus componentes deben ser de una temperatura de 22.5± 12.5ºC y una humedad relativa de relativa de 50 ± 10%. Las ordenes al robot serán dadas por medio del puerto serial disponible en Teensyduino, bien conectado a un PC o a una raspberry; en caso de que el robot no pueda seguir la referencia, por la limitante física de alguna de las articulaciones, el robot se moverá a la posición más próxima hasta ser detenido por el final de carrera por software.

Para la operación correcta del robot, será necesario que se arranque en la posición de “home”, donde las coordenadas articulares son todas 0, o se le envíe una orden de homing, para llevarlo a esta misma posición; se aconseja devolver al robot a dicha posición al finalizar las pruebas, por medio de órdenes a través del puerto serial.

Si se desea devolver el robot manualmente al home, se deberá desconectar los acoples de los ejes y proceder a montarlo en la posición correcta, esto con el objeto de evitar daños a los motores.

4. ENSAYOS A REALIZAR

4.1. Homing

Se envía una orden de homing al robot para su retorno a cero, utilizando el formato de comandos ideado para ello.

4.2. Cinemática directa

A través del puerto serial se le indica al robot la posición en coordenadas articulares de referencia, utilizando el formato del protocolo ad hoc diseñado para ello.

4.3. Cinemática Inversa

A través del puerto serial se le indica al robot las componentes X y Z de la posición de la muñeca, y la rotación deseada de la herramienta en formato “roll pitch yaw”, utilizando el formato del protocolo ad hoc diseñado para ello.

4.4. Rutina

Se establece una rutina para que el robot realice, encadenando ordenes de posiciones articulares. Para que una orden se ejecute, la orden anterior debe haber sido completadapor el robot y los motores deben haber alcanzado la posición de referencia.

4.5. Validación

Para la validación de los resultados obtenidos en el microcontrolador, y en lo referente a la posición alcanzada por el robot, se hace uso de las funciones y herramientas desarrolladas en Matlab, como parte este proyecto.

Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería del Diseño

PRESUPUESTO

Trabajo de Fin de Master

Master en Ingeniería Mecatrónica

SOFTWARE

REFERENCIA	Denominación	Uds.	Cantidad	Precio	Total
LM	Licencia de Matlab	h	50	0.44 €	22.20 €
LMO	Licencia de Microsoft Office	h	60	0.07 €	4.20 €

HARDWARE

REFERENCIA	Denominación	Uds	Cantidad	Precio	Total
PC-1	Ordenador Portatil hacer Nitro	15 h	300	0.20 €	60.00 €
				COSTE RECURSOS INFORMATICOS	86.40 €

PERSONAL

REFERENCIA	Denominación	Uds.	Cantidad	Precio	Total
I-1	Tutores del proyecto (Ingenieros)	h	40	25.00 €	1.000.00 €
I-2	Titulo Universitario (Ingenieros)	h	300	11.22 €	3.366.00 €
I-3	Tecnico de Laboratorio	h	8	20.00 €	160.00 €
				COSTE PERSONAL	4.526.00 €

MOTORES Y CONTROL

REFERENCIA	Denominación	Uds.	Cantidad	Precio Total
539473	Motor Maxon Eci 30W	1	213.82 €	213.82 €
166951	Reductora GP 32 C 32mm	1	171.37 €	171.37 €
499360	Encoder ENC 16 EASY 1000 pulsos	1	81.73 €	81.73 €
539477	Motor Maxon Eci 50W	2	196.64 €	393.28 €
166948	Reductora GP 32 C 32mm	2	197.00 €	394.00 €
499360	Encoder ENC 16 RIO 4096 pulsos	2	166.73 €	333.46 €
283858	Motor Maxon Ec-max 25W	2	201.91 €	403.82 €
166951	Reductora GP 22 HP 22mm	2	210.55 €	421.10 €
499360	Econder MR ML CPT	2	102.82 €	205.64 €
414533	ESCON 36/3 EC	2	171.00 €	342.00 €
438725	ESCON MODULE 50/5	3	163.91 €	491.73 €
565119	Microcontrolador Teensy 4.1	1	40.95 €	40.95 €
619-0323	Rodamiento de bolas 6804-2RS	2	3.47 €	6.94 €
146-9290	Rodamiento lineal HK 609	4	2.74 €	10.96 €
7506732	Acoplamiento Omron	2	24.48 €	48.96 €
S- M4	Tornillos y tuercas M4	14	0.50 €	7.00 €
JFM 151709	Rodamiento IGUS	6	2.76 €	16.56 €
AC -6	Acoplamiento rígido 6mm	3	4.99 €	14.97 €
			TOTAL	3.598.29 €

CONEXIONADO ELECTRÓNICO

REFERENCIA	Denominacion	Cantidad	Precio	Total
CP	Cable plano IDC, 10 pines, 6 m	1	10.99 €	10.99 €
CD	Pack cables Dupont	1	3.35 €	3.35 €
CAMU	Cable USB a microUSB	1	5.00 €	5.00 €
2332760	Carcasa conector Micro Fit 3	5	0.44 €	2.20 €
6701853	Crimpado Micro Fit 3 Hembra	50	0.10 €	5.00 €
403957	Cable de alimentacion Maxon 36/3	2	8.74 €	17.48 €
403964	Cable de E/S analogicas	2	10.68 €	21.36 €
403962	Cable de motor DC 36/3	2	11.02 €	22.04 €
			TOTAL	87.42 €

COMPONENTES MECÁNICOS

REFERENCIA	Denominacion	Cantidad	Precio	Total
V1	Varilla metalica 6mm * 1m	1	7.12 €	7.12 €
A1	Pieza Aluminio eje codo	1	11.00 €	11.00 €
A2	Pieza cuadrada eje hombro	1	6.99 €	6.99 €
AR-1	Arboles 1 y 2	2	7.50 €	15.00 €
AR-2	Arboles 2 y 3	2	5.50 €	11.00 €
E1	Eje 1	1	5.00 €	5.00 €
E2	Eje 2	1	5.00 €	5.00 €
AS1	Arbol Salida muñeca	1	5.00 €	5.00 €
AS2	Arbol Salida codo	1	7.00 €	7.00 €
			TOTAL	73.11 €

Coste PLA [€/kg]	34.7	**Masa de la pieza [kg]**	0.5
Coste luz [€/kWh]	0.4	**Tiempo impresión [h]**	40.8
Consumo medio[kW]	0.9	**Coste material PLA**	16.98 €
Coste por hora de luz [€/h]	0.3	**Electricidad**	12.72 €
Coste de impresora [€]	3494	**Coste amortización**	35.67 €
Tiempo amortización [años]	2	**Coste fallos**	32.68 €
Dias al año	250		
Horas por dia	8		
Coste amortización [€/h]	0.9		
Tasa de fallos	0.5	**Coste Impresión PLA**	98.05 €

Coste Nylon [€/kg]	97.6	**Masa de la pieza [kg]**	0.5
Coste luz [€/kWh]	0.4	**Tiempo impresión [h]**	35.7
Consumo medio[kW]	0.5	**Coste material Nylon**	52.22 €
Coste por hora de luz [€/h]	0.2	**Electricidad**	6.31 €
Coste de impresora [€]	25999	**Coste amortización**	57.96 €
Tiempo amortización [años]	8	**Coste fallos**	58.24 €
Dias al año	250		
Horas por dia	8		
Coste amortización [€/h]	1.6		
Tasa de fallos	0.5	**Coste Impresión SLS**	174.73 €

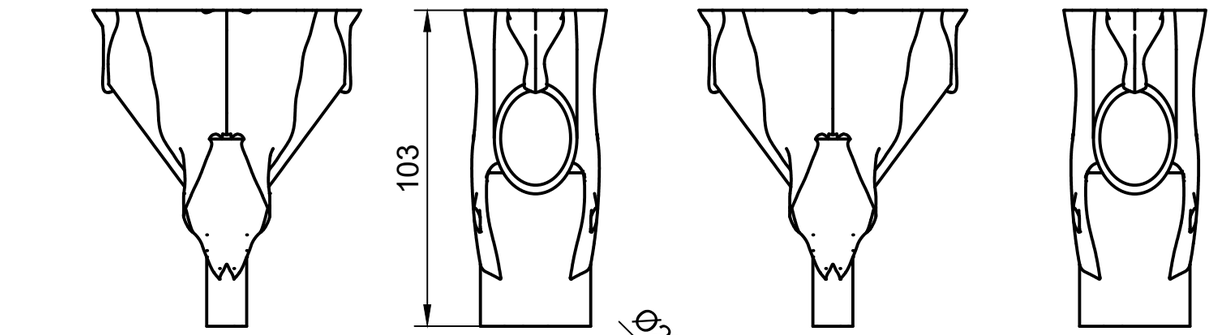
Universidad Politécnica de Valencia

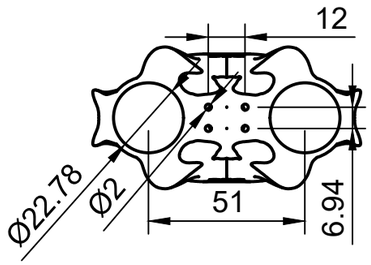
Escuela Técnica Superior de Ingeniería del Diseño

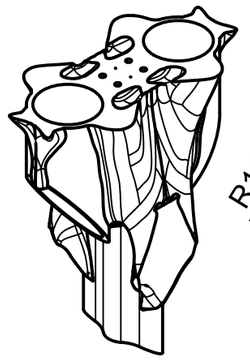
PLANOS

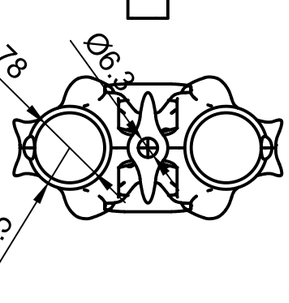
Trabajo de Fin de Master

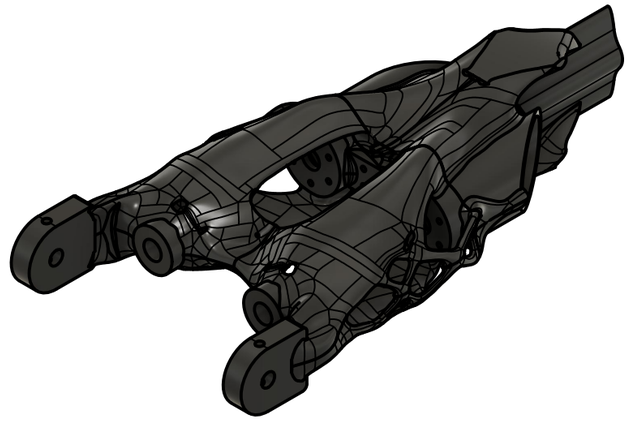
Master en Ingeniería Mecatrónica

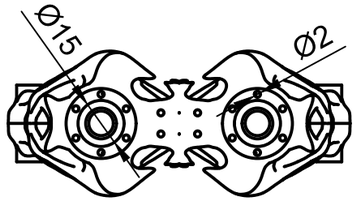


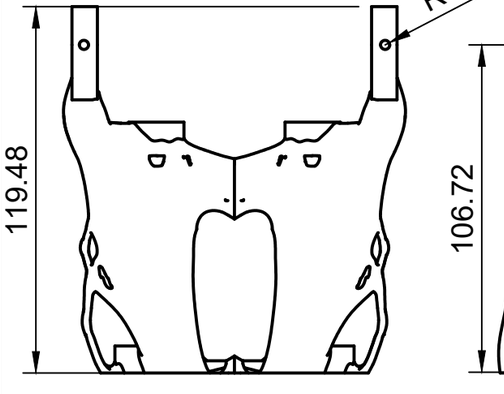


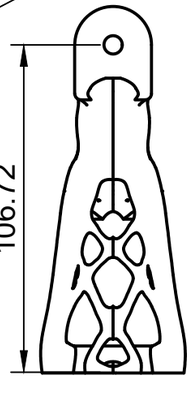


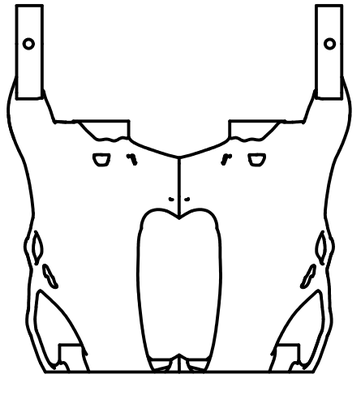


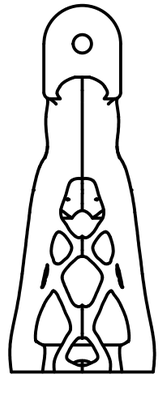


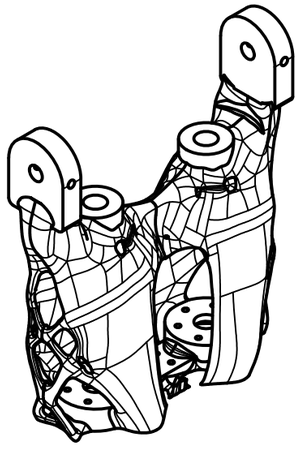


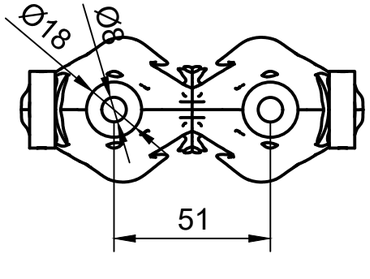




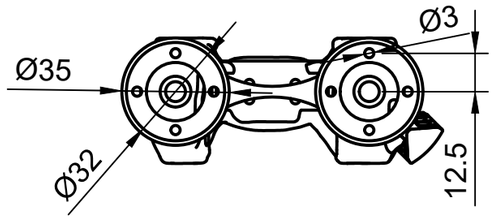


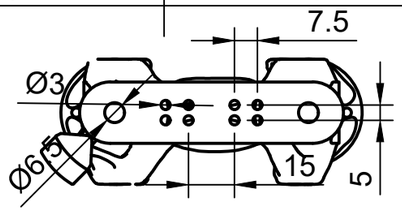


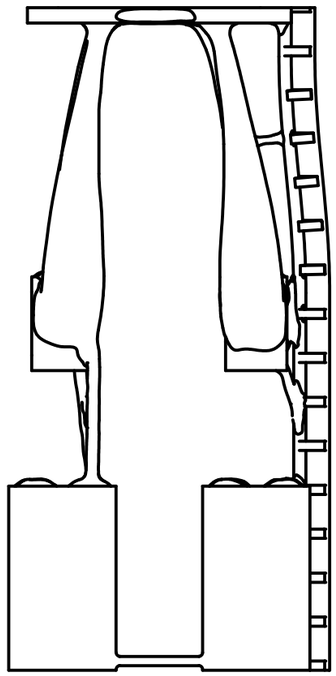


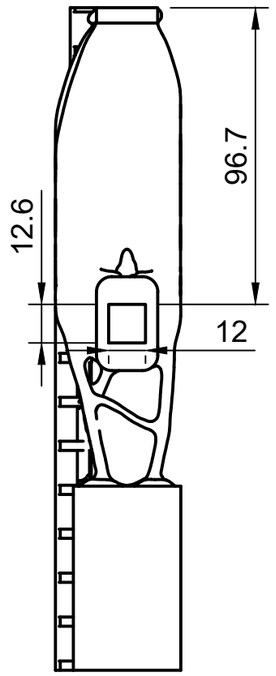


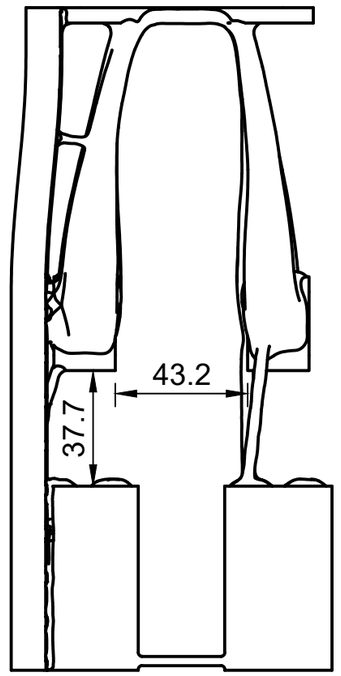
Dept.	Technical reference	Created by Mathias Walter Rodriguez 09/08/2023	Approved by
		Document type	Document status
		Title Ensamble_brazo_fusion_Antebrazo STP	DWG No.
			Rev. / Date of issue / Sheet 1/1

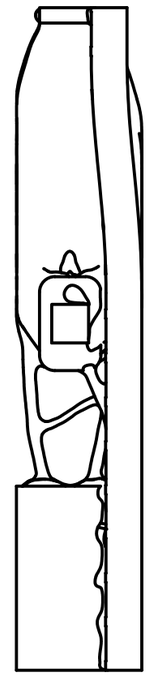


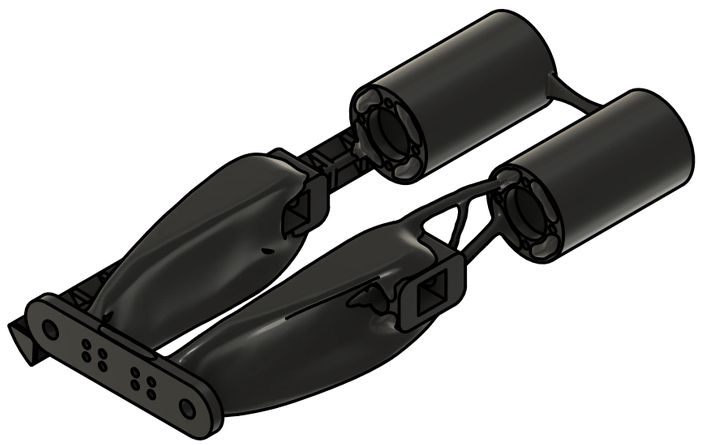




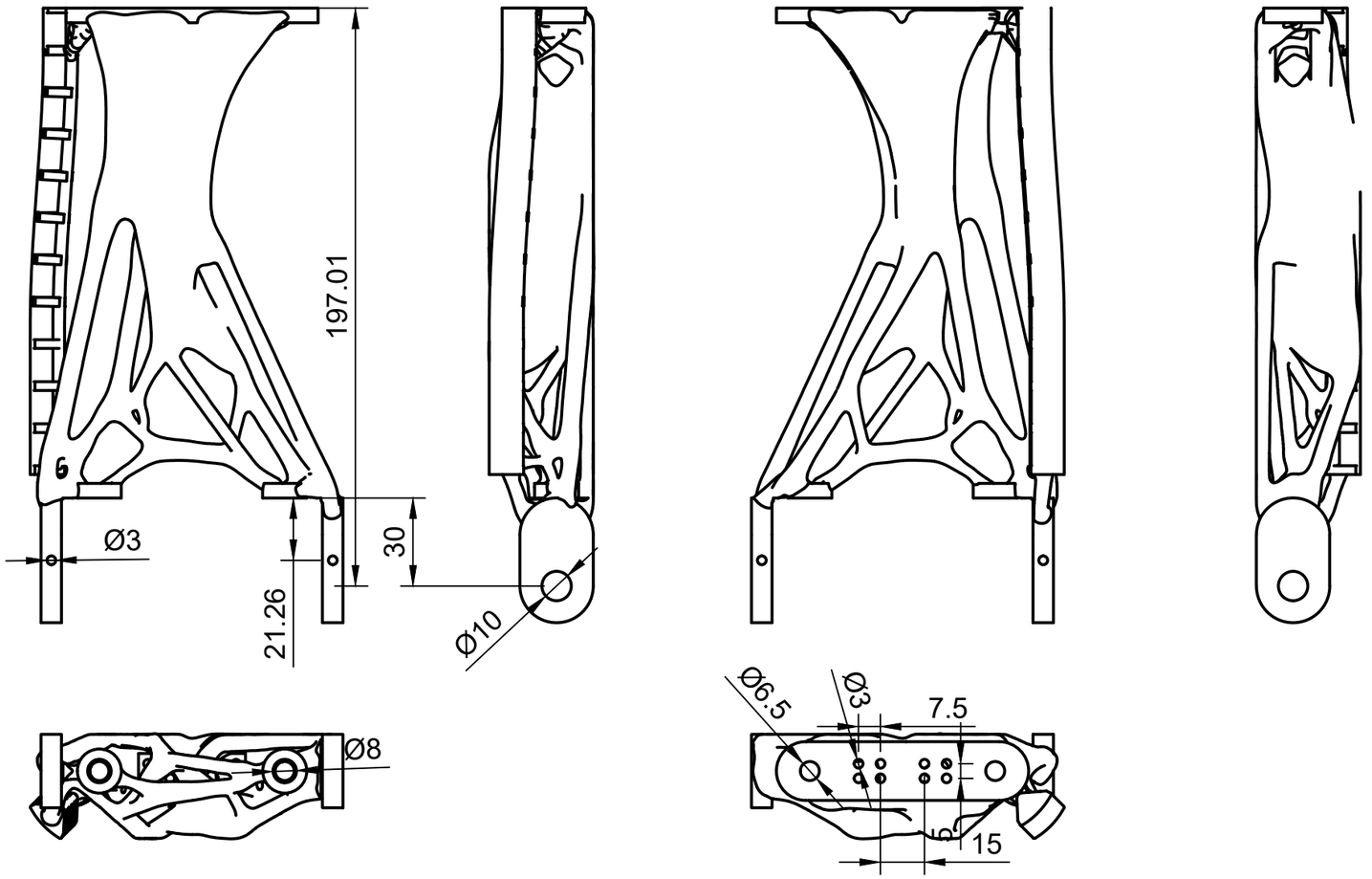


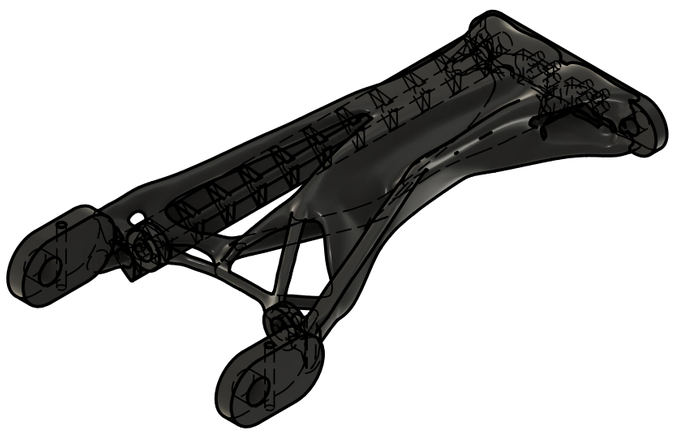




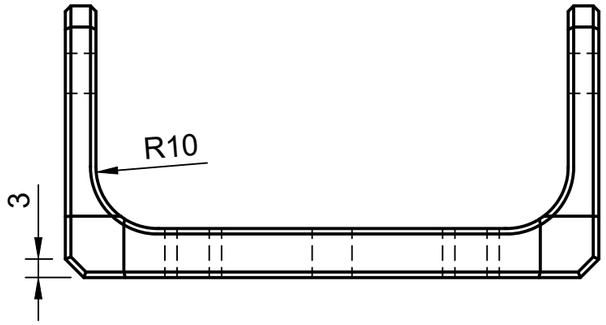


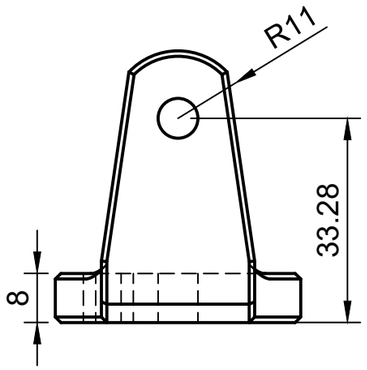
Dept.	Technical reference	Created by	Approved by
		Mathias Walter Rodriguez 08/08/2023	
		Document type	Document status
		Title: Brazo_generativo_parte1_v2	DWG No.
			Rev. / Date of issue / Sheet: 1/1

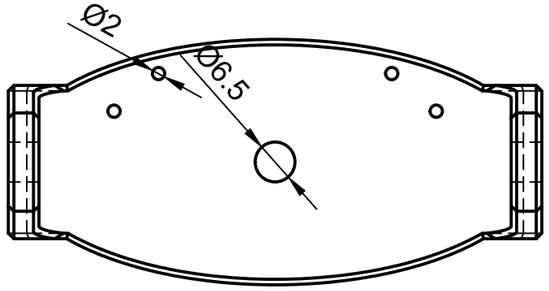


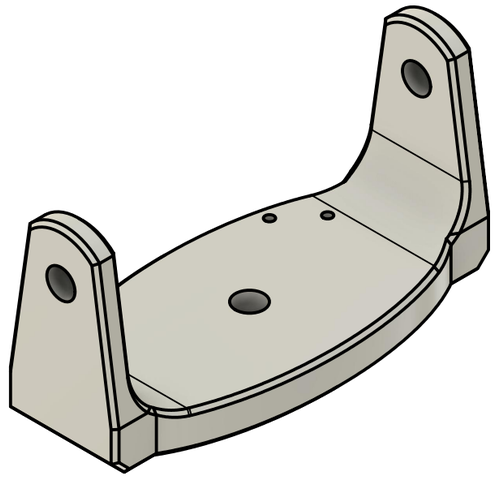


Dept.	Technical reference	Created by	Approved by
		Mathias Walter Rodriguez 08/08/2023	
		Document type	Document status
		Title: Brazo posterior p2	DWG No.
			Rev. / Date of issue / Sheet: 1/1

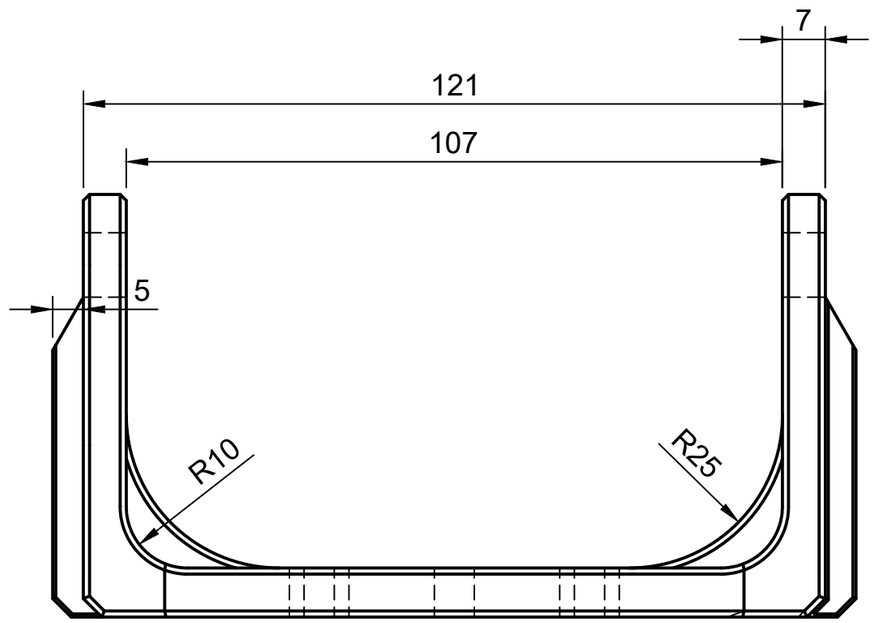


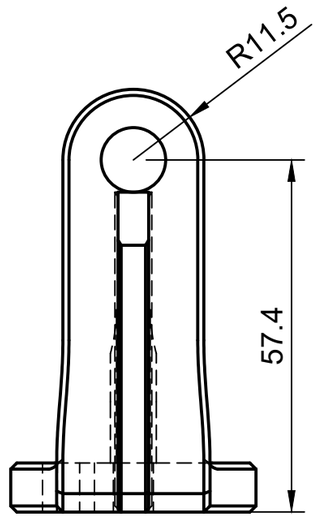


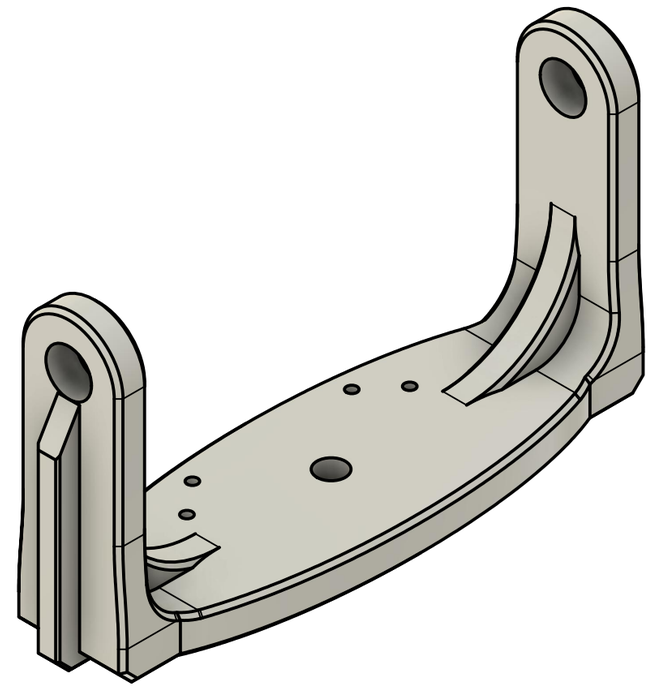


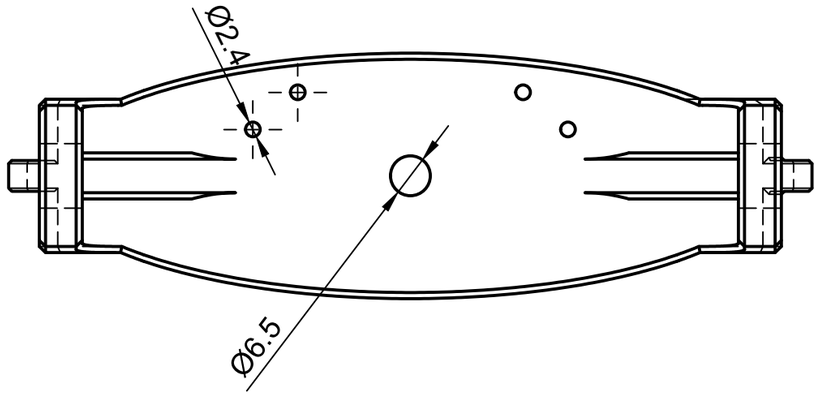


Dept.	Technical reference	Created by Mathias Walter Rodriguez 09/09/2023	Approved by
		Document type	Document status
		Title Tope antebrazo	DWG No.
			Rev. / Date of issue / Sheet 1/1

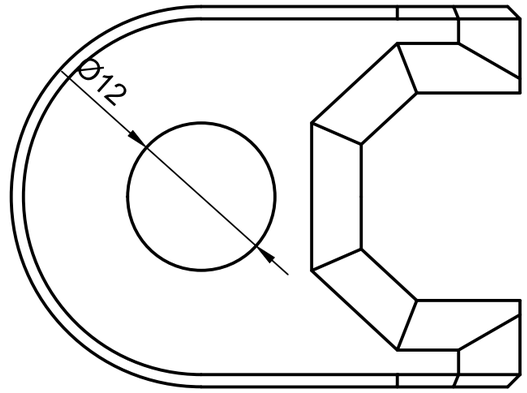


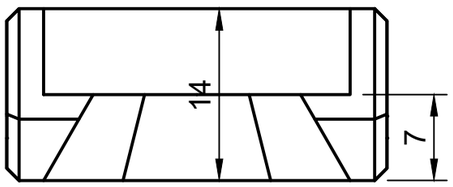


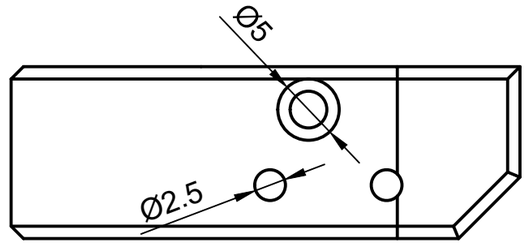


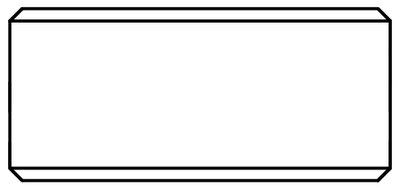


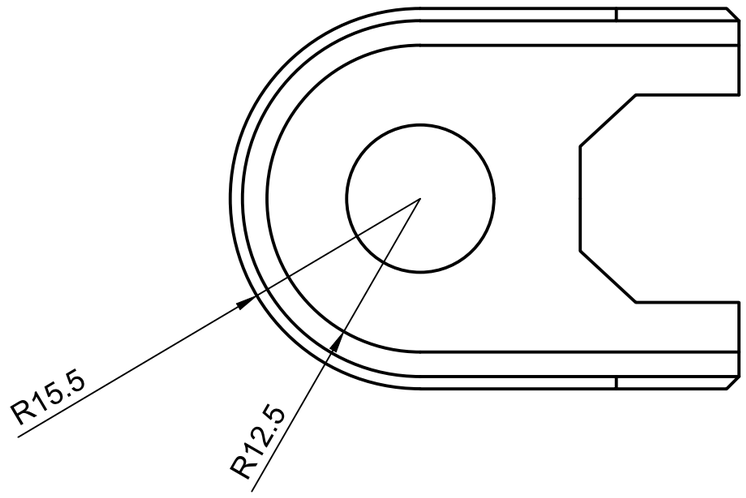
Dept.	Technical reference	Created by Mathias Walter Rodriguez 09/09/2023	Approved by
		Document type	Document status
		Title Tope brazo	DWG No.
			Rev. / Date of issue / Sheet 1/1

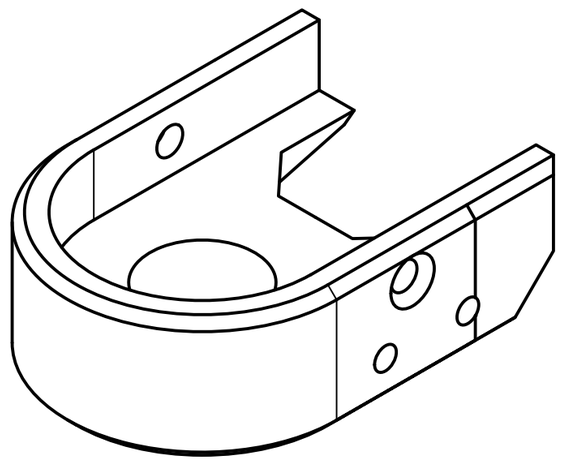




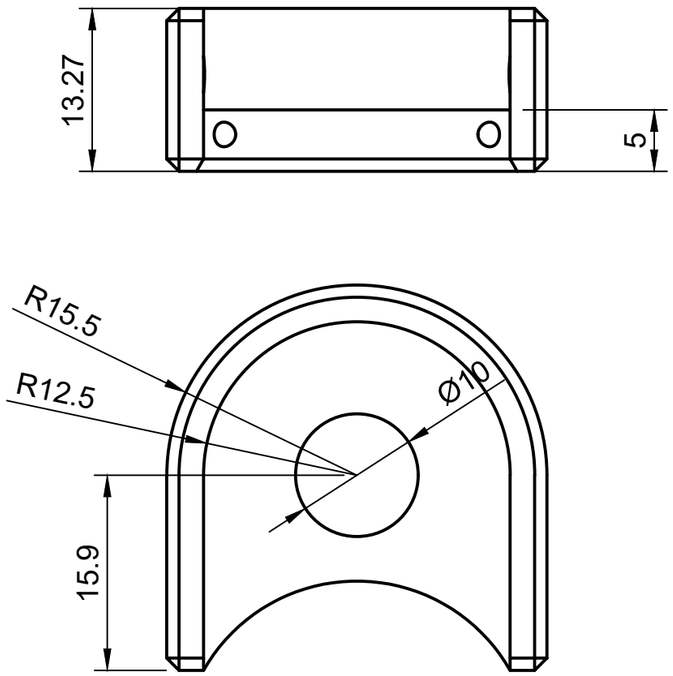


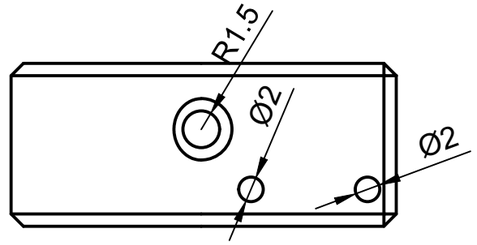


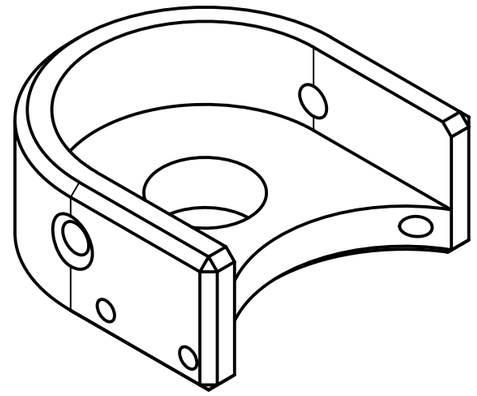




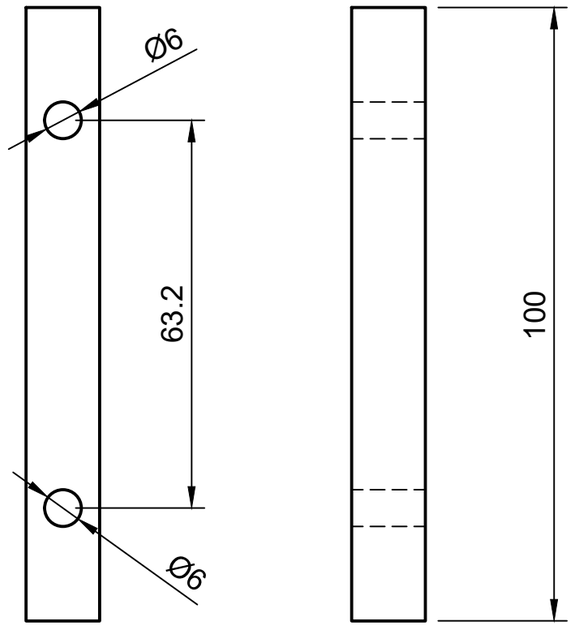
Dept.	Technical reference	Created by Mathias Walter Rodriguez 8/09/2023	Approved by
		Document type	Document status
		Title Montura FdC brazo	DWG No.
			Rev. / Date of issue / Sheet 1/1

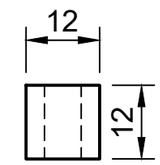


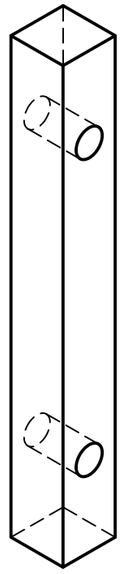




Dept.	Technical reference	Created by Mathias Walter Rodriguez 8/09/2023	Approved by
		Document type	Document status
		Title Montura FdC antebrazo	DWG No.
			Rev. / Date of issue / Sheet 1/1







Dept.	Technical reference	Created by Mathias Walter Rodriguez 03/07/2023	Approved by
		Document type	Document status
		Title Eje cuadrado	DWG No.
			Rev. / Date of issue / Sheet 1/1