Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo Fin de Grado

Diseño, fabricación y control de una mano robótica para el robot IRB120

Autor: Sergio Rodríguez Manzanares

Tutor/es: Rafael Barea Navarro

2017

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería electrónica y automática industrial

Trabajo Fin de Grado

Diseño, fabricación y control de una mano robótica para el robot IRB120

Autor: Sergio Rodríguez Manzanares

Tutor/es: Rafael Barea Navarro

TRIBUNAL:

Presidente: J. ANTONIO JIMÉNEZ CALVO

Vocal 1º: MANUEL OCAÑA MIGUEL

Vocal 2º: RAFAEL BAREA NAVARRO

FECHA:

AGRADECIMIENTOS

Gracias a toda mi familia por el apoyo que me ha brindado durante estos cuatro años, en especial a mi hermano Dani que siempre ha sabido sacar lo mejor de mí y a mis padres por su esfuerzo para que me dedique a mi pasión.

Gracias a mis profesores y compañeros por haberme enseñado tanto estos años, tanto académicamente como personalmente y en especial a mi tutor Rafa por darme la oportunidad de realizar este proyecto y ser comprensivo con las dificultades que han ido surgiendo.

CONTENIDO

Agradecimientos 5

Tabla de figuras 11

Lista de tablas 15

Resumen en castellano 17

Abstract 18

Palabras clave 19

1 Introducción 21

1.1 Antecedentes 21

1.1.1 Mano robótica Shadow Dexterous 21

1.1.2 Mano robótica de la universidad de Washington 22

1.1.3 El programa Delfín 23

1.1.4 Los problemas del diseño actual 24

1.2 Objetivo del proyecto 24

1.3 Arquitectura del proyecto 25

2 Diseño Mecánico 27

2.1 Introducción 27

2.1.1 Robot Inmoov 28

2.1.2 SolidWorks 29

2.1.3 Arduino Uno 30

2.1.4 MakerBot 32

2.1.5 IRB120 34

2.2 Diseño de nuevas piezas 35

2.2.1 Cama de Servos 36

2.2.2 Agarre motor 37

2.2.3 Soporte Mano 38

2.2.4 SPT Servo 39
2.2.5 Ensamblaje final 40
2.3 Ensamblaje de la mano en SolidWorks 42
2.3.1 Descarga de ficheros STL y transformación 43
2.3.2 Creación del ensamblaje en SolidWorks 47
2.4 Impresión 3D 50
2.4.1 Materiales 50
2.4.2 Problemas comunes 51
2.4.3 Proceso de impresión 3D 52
2.5 Montaje de la mano real 55
3 Software de control de la mano robótica 61
3.1 Introducción 61
3.2 Arduino 63
3.2.1 Servomotores Micro Servo 9G 63
3.2.2 Control de posición 64
3.2.3 Comunicación Serie 65
3.2.4 Conexión eléctrica 66
3.2.5 Control de velocidad 67
3.2.6 Diagrama de flujo del software de control de Arduino 68
3.3 Matlab 69
4 Resultados 73
4.1 Cumplimiento de objetivos 73
4.1.1 Objetivo del sistema mecánico 73
4.1.2 Objetivo del software de control de la mano robótica 75
4.2 Errores 79
4.2.1 Errores en el diseño mecánico 79

4.2.2 Errores en el software de control de la mano robótica 80
5 Conclusiones y trabajos futuros 81
6 Planos 83
Software de control Arduino UNO 89
Interfaz gráfica en Matlab 91
7 Presupuesto 99
7.1 Costes materiales 99
7.2 Costes totales 99
8 Bibliografía 101

TABLA DE FIGURAS

Ilustración 1-1: Mano robótica Shadow Dexterous 21

Ilustración 1-2: Mano robotica de la universidad de Washington 22

Ilustración 1-3: Distintas posiciones de la mano robótica de la universidad de Wasinghton 22

Ilustración 1-4: Primer prototipo de la mano robótica del programa Delfín 23

Ilustración 1-5: Comparación entre mano la Inmoov original (Izquierda) y la adaptación realizada por el programa Delfín (Derecha) 23

Ilustración 1-6: Diagramas de bloques del proyecto 25

Ilustración 2-1: Robot Inmoov completo 28

Ilustración 2-2: Arduino UNO R3 30

Ilustración 2-3: Dos Extrusores trabajando a la vez 32

Ilustración 2-4: Diferencias entre la precisión de capa 33

Ilustración 2-5: Brazo robótico IRB120 34

Ilustración 2-6: Herramienta de extrusión de SolidWorks 35

Ilustración 2-7: Herramienta de corte de SolidWorks 35

Ilustración 2-8: Vista general de "Cama de servos" 36

Ilustración 2-9: Taladros avellanados y guiado de cables en "Cama de servos" 36

Ilustración 2-10: Vista general de "Agarre motor" 37

Ilustración 2-11: Ensamblaje de motor y "Agarre motor" 37

Ilustración 2-12: Vista general de "Soporte mano" 38

Ilustración 2-13: Vista general de "SPT Servo" 39

Ilustración 2-14: Vista general de ensamblaje de piezas diseñadas 40

Ilustración 2-15: Distancia entre motores 41

Ilustración 2-16: Proceso de transformación de STL en ensamblajes 42
Ilustración 2-17: Listado de piezas para descargar 43
Ilustración 2-18: Cambiar escenario en FreeCAD 44
Ilustración 2-19: Obtener malla en FreeCAD 45
Ilustración 2-20: Refinar forma en FreeCAD 45
Ilustración 2-21: Beneficios de refinar forma 46
Ilustración 2-22: Exportación a pieza CAD 46
Ilustración 2-23: Solidos en ensamblaje generado en FreeCAD 47
Ilustración 2-24: Menú Abrir en SolidWorks 47
Ilustración 2-25: Menú selección de nuevo documento en SolidWorks 48
Ilustración 2-26: Seleccionar piezas para insertar 48
Ilustración 2-27: Ensamblaje completo en 3D 49
Ilustración 2-28: Menú abrir de ReplicatorG 52
Ilustración 2-29: Colocación del modelo 3D para su impresión 52
Ilustración 2-30: Botón generar "GCode" 53
Ilustración 2-31: Parámetros de impresión 53
Ilustración 2-32: Botón de exportación a X3G 54
Ilustración 2-33:Impresora 3D imprimiendo piezas 54
Ilustración 2-34: Unión de falanges mediante pegamento 55
Ilustración 2-35: Ensamblaje de falanges mediante tornillos 55
Ilustración 2-36: Ensamblaje de la palma de la mano 56
Ilustración 2-37: Ensamblaje completo de mano 56
Ilustración 2-38: Ensamblaje del soporte de motores y mano 57
Ilustración 2-39:Ensamblaje de motor 57

Ilustración 2-40:Ensamblaje de motores al soporte 58
Ilustración 2-41: Unión de soporte con mano 58
Ilustración 2-42: Pieza de guiado de cables 59
Ilustración 2-43:Guiado de cables a través de la mano 59
Ilustración 2-44: Amarre de cables a la sujeción del motor 60
Ilustración 3-1: Elementos utilizados en la interfaz gráfica 61
Ilustración 3-2: Esquema general de control 62
Ilustración 3-3: Dimensiones del motor Micro Servo 9g 63
Ilustración 3-4: Señales PWM para Servomotores 63
Ilustración 3-5: Diagrama de conexión 66
Ilustración 3-6: Control de velocidad de los motores 67
Ilustración 3-7: Diagrama de flujo del código de control 68
Ilustración 3-8: Interfaz diseñada 70
Ilustración 3-9: Flujograma de uso de la interfaz 71
Ilustración 4-1: Manos robóticas vista frontal 73
Ilustración 4-2: Manos robóticas vista lateral 74
Ilustración 4-3: Control mediante comandos 75
Ilustración 4-4: Interfaz de control de la mano robótica 75
Ilustración 4-5: Circuito de control 76
Ilustración 4-6: Mano robótica real cerrada 76
Ilustración 4-7:Mano robótica real en reposo 77
Ilustración 4-8: Mano robótica real abierta 77
Ilustración 4-9:Mano robótica real con índice levantado 78
Ilustración 4-10: Mano robótica real con dedo índice recogido 78

LISTA DE TABLAS

Tabla 1-1: Dimensiones de los servomotores 24

Tabla 3-1: Comandos disponibles 65

Tabla 7-1Costes materiales: 99

Tabla 7-2: Costes totales 99

RESUMEN EN CASTELLANO

El objetivo principal de este proyecto es la construcción de una mano robótica que se pueda conectar al robot IRB120.

Para ello se partirá de un modelo 3D de una mano robótica “Open Source”, se diseñarán las piezas necesarias para adaptarla al IRB120 y, finalmente, con la ayuda de una impresora 3D se construirá el sistema. El control de la mano robótica será gestionado por un microcontrolador Arduino Uno R3. También se desarrollará el código de control, así como una interfaz en Matlab.

ABSTRACT

The main objective of this project is the construction of a robotic hand that can be connected to the IRB120 robot.

It will be based on a 3D model "Open Source" robotic hand, the necessary parts will be designed to adapt it to the IRB120 and finally, using a 3D printer will build the system. The robot hand control will be managed by an Arduino Uno R3 microcontroller. The control code will also be developed, as well as an interface in Matlab.

PALABRAS CLAVE

- Inmoov / Inmoov
- Mano robótica / Robotic Hand
- IRB120 / IRB120
- Arduino / Arduino
- Control / Control

1 INTRODUCCIÓN

1.1 ANTECEDENTES

Desde hace varios años se ha buscado la creación de manipuladores capaces de simular el comportamiento de una mano humana. Durante los años de la revolución industrial y los posteriores, se desarrollaron numerosos prototipos de manos similares a las humanas pero realizadas con elementos mecánicos. Sin embargo, no es hasta finales del siglo XX cuando empiezan a desarrollarse manos robóticas realmente útiles. Esto se debe principalmente al desarrollo de la electrónica y la computación. En el siglo XXI ya existen modelos completamente funcionales y muy similares a una mano humana. Actualmente existen manos robóticas capaces de sentir objetos, con casi tantos grados de libertad como una mano humana y capaces de ejercer las mismas fuerzas. Entre las distintas manos robóticas actuales del mercado destacan las siguientes.

1.1.1 Mano robótica Shadow Dexterous

La mano robótica Shadow Dexterous [1] es la mano robótica más avanzada existente en el mercado. Cuenta con 20 grados de libertad que permiten emular prácticamente cualquier movimiento de una mano humana real. Dispone de 129 sensores en toda la mano (Posición, fuerza, presión, tacto, temperatura), también permite integración con ROS. Esta mano robótica es concebida para centros de investigación y desarrollo.

Ilustración 1-1: Mano robótica Shadow Dexterous

1.1.2 Mano robótica de la universidad de Washington

Los investigadores Zhe Xu y Emanuel Todorov de la universidad de Washington [2] están desarrollando la mano robótica más antropomórfica que se haya creado hasta la fecha. Tomando como modelo la biomecánica de la mano humana, han desarrollado mediante impresión 3D una mano robótica cuya estructura es muy similar a la estructura ósea de una mano humana.

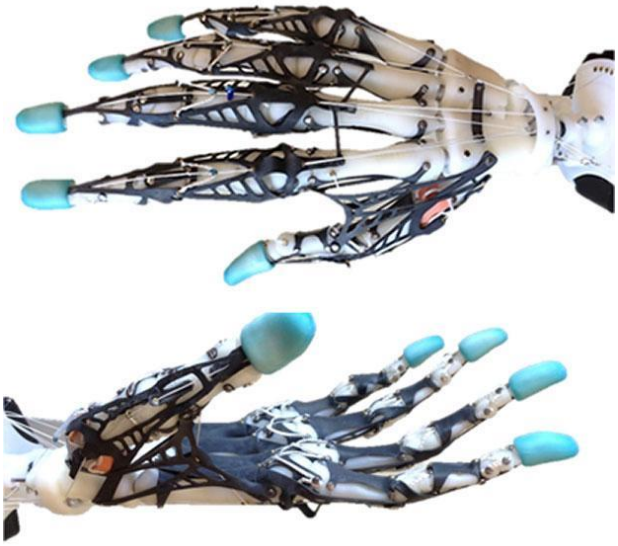


Ilustración 1-2: Mano robotica de la universidad de Washington

Esta mano robótica cuenta con 10 servomotores que permiten realizar distintos tipos de movimientos.

Ilustración 1-3: Distintas posiciones de la mano robótica de la universidad de Wasinghton

1.1.3 El programa Delfín

Desde 1995 existe un programa denominado “Programa Delfín” [3] que promueve la cooperación y la movilidad estudiantil mediante estancias académicas de investigación. En el año 2016, uno de los grupos del programa Delfín destinados en la Universidad de Alcalá se propuso diseñar y fabricar una mano robótica para el brazo robótico IRB120. Este grupo consiguió realizar un primer prototipo funcional.

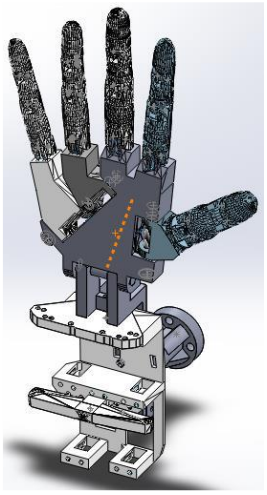


Ilustración 1-4: Primer prototipo de la mano robótica del programa Delfín

Este prototipo es una adaptación del diseño de la mano Inmoov creada por Gael Langevin [4] del cual se hablará más adelante. El diseño original de la mano Inmoov incluía un antebrazo donde se albergaba la electrónica y los motores. Por motivos de espacio no era posible incluir el antebrazo. Debido a esto se tuvo que realizar la adaptación.

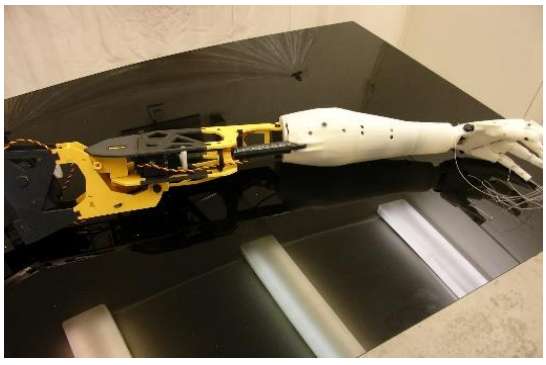


Ilustración 1-5: Comparación entre mano la Inmoov original (Izquierda) y la adaptación realizada por el programa Delfín (Derecha)

Este proyecto toma como base el primer prototipo de la mano robótica diseñada por el grupo del programa Delfín.

1.1.4 Los problemas del diseño actual

El principal problema de este primer prototipo era la poca flexibilidad que aportaba. La conexión se realizaba a través del controlador del IRB120, utilizando como interfaz de control las cuatro salidas digitales del controlador IRC5. Esto limitaba mucho al sistema ya que solo se disponían 16 posibles posiciones.

Otro problema detectado ha sido la falta de material y la dificultad de encontrar repuestos en caso de avería. El sistema emplea en su mayoría piezas impresas en 3D y herramientas libres, sin embargo, los motores por dificultad de construcción deben ser comprados. Los motores que se emplearon en el proyecto Delfín eran motores RS2 de la empresa Modelcraft, servomotores potentes, pero difíciles de encontrar actualmente. Además, son motores bastante caros. Estos motores son los que se utilizaban en el diseño original de la mano Inmoov.

1.2 OBJETIVO DEL PROYECTO

El objetivo de este proyecto es solucionar estos problemas y crear una documentación que sirva de referencia si se desea crear el sistema de nuevo.

El primer problema requiere crear software de control desde cero. Se mantendrán las conexiones eléctricas por si se desea volver al control del proyecto Delfín. El nuevo control ha de ser más versátil en cuanto a la comunicación y más preciso en cuanto a las posiciones que se puedan alcanzar. Se creará software para que cualquier ordenador se pueda comunicar mediante el uso de un cable USB. Con el objetivo de facilitar el uso se diseñará una interfaz gráfica guiada.

Para solventar el segundo problema se cambiarán los motores por otros que actualmente son estándar, baratos y fáciles de encontrar, los motores Micro Servo 9G. Con el cambio de motores también se debe cambiar el socket de adaptación. Por ello se diseñarán y fabricarán nuevas piezas. Como característica añadida se conseguirá una reducción en el espacio empleado, así como del peso general.

Dimensiones

Modelo	Micro Servo 9G	Modelcraft RS2
L (mm)	32	41
W (mm)	12	21
H (mm)	31	42

Tabla 1-1: Dimensiones de los servomotores

Todo esto se implementará en el prototipo ya creado en el proyecto Delfín, reutilizando las piezas que sigan siendo válidas. La placa en la que se encuentra el microcontrolador no se modificará por si se desea realizar el control antiguo.

1.3 ARQUITECTURA DEL PROYECTO

En esta sección se explica de forma superficial la estructura que seguirá la memoria de este trabajo y se aportará una visión general del proyecto con el objetivo de facilitar la comprensión del mismo.

El proyecto consta de dos grandes bloques claramente diferenciados.

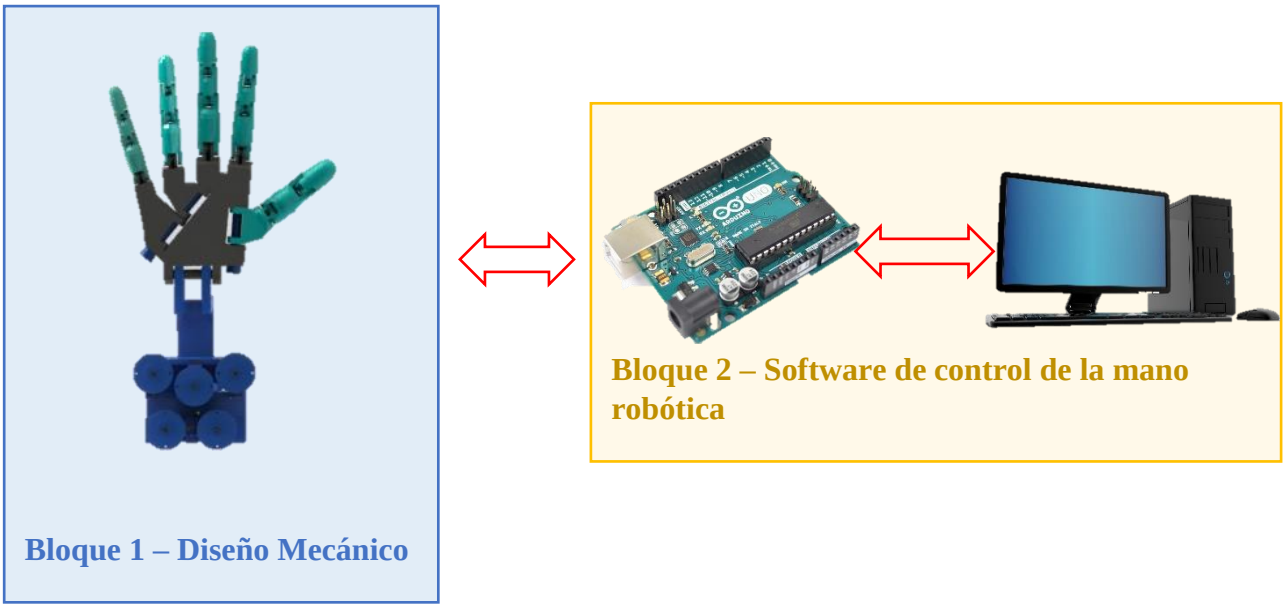


Ilustración 1-6: Diagramas de bloques del proyecto

El bloque "Diseño mecánico" es el referente al diseño y ensamblaje mecánico de la mano robótica Inmoov. En este bloque se abordará la problemática del acoplamiento del diseño, la fabricación y el ensamblaje para realizar el acople al robot IRB120. El capítulo dos describe este bloque y se divide en cuatro secciones correspondientes a:

- Diseño de nuevas piezas
- Ensamblaje de la mano virtual
- Impresión 3D de las piezas reales
- Ensamblaje de la mano robótica real

El bloque "Software de control de la mano robótica" trata sobre el software creado para el control de la mano robótica y la interfaz para llevar acabo dicho control. El capítulo tres detalla este bloque y se divide en dos secciones:

- Programación en Arduino
- Programación de interfaz gráfica

2 DISEÑO MECÁNICO

2.1 INTRODUCCIÓN

El propósito de esta sección de introducción es doble. El primer propósito es servir de guía del capítulo de diseño mecánico. El segundo propósito es describir brevemente las herramientas que se han utilizado para abordar la problemática del diseño y fabricación de la parte mecánica de la mano robótica.

Este capítulo se compone de:

- **Introducción:** Proporciona una guía del capítulo y describe las herramientas utilizadas durante el diseño y creación de la mano robótica.
- **Diseño de nuevas piezas:** Se describen detalladamente cada una de las piezas nuevas diseñadas y como ha sido su proceso de diseño.
- **Ensamblaje de la mano virtual:** Indica el proceso a seguir para obtener un modelo virtual de la mano robótica.
- **Impresión 3D de las piezas reales:** Se muestra el proceso que se debe seguir para realizar una impresión correcta de las piezas en 3D.
- **Ensamblaje de la mano robótica real:** Proporciona unas indicaciones a seguir para realizar el montaje de la mano robótica.

Con esto queda descrita la arquitectura de este capítulo y como se va a proceder en lo que sigue.

Las herramientas que se han utilizado principalmente son:

- **Robot Inmoov:** Como base se ha partido de la mano robótica del robot Inmoov.
- **SolidWorks:** Programa de diseño CAD utilizado en la creación de cada una de las piezas diseñadas.
- **Arduino UNO:** Este es el microcontrolador que se utiliza para realizar el control de posición.
- **Maketbot:** La impresora 3D que se ha utilizado durante todo el proceso.
- **IRB120:** El brazo robótico con el cual se quiere acoplar la mano robótica.

2.1.1 Robot Inmoov

Gael Langevin es un diseñador francés que se embarcó en el proyecto de crear un robot humanoide "Open Source". Con el objetivo inicial de crear simplemente una mano protésica inició el proyecto en enero de 2012. Desde entonces ha diseñado y creado nuevas partes del robot con la ayuda de una extensa comunidad.

El robot Inmoov, en su totalidad es una obra de ingeniería bastante compleja, una prueba del potencial que tienen las licencias "Open Source".

La página oficial de Inmoov proporciona archivos STL (Standard Triangle Language) de todos los componentes del robot Inmoov. Este formato es un estándar en la actualidad, todos los programas de diseño e impresión 3D son capaces de trabajar con STL. La página también proporciona el software de control y manuales para realizar la construcción del robot.

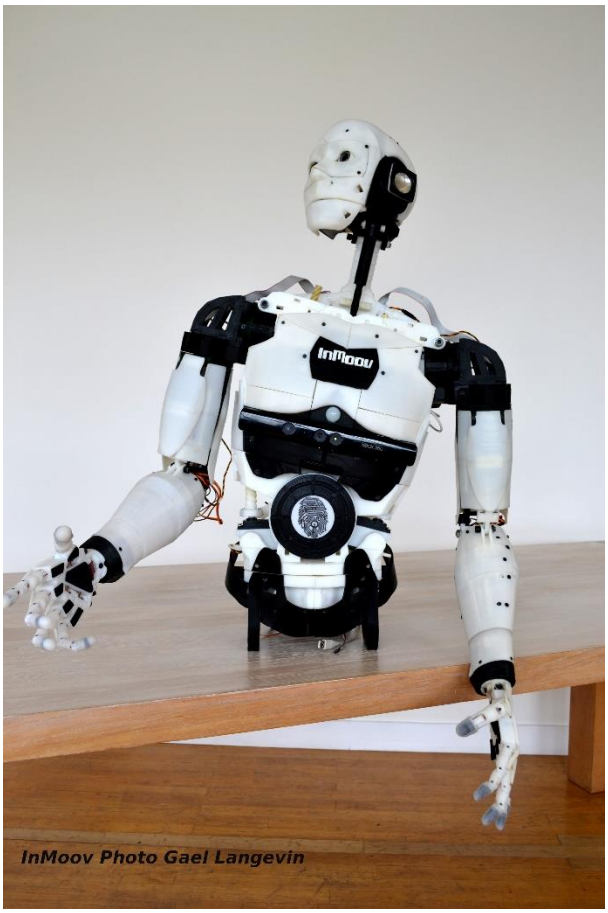


Ilustración 2-1: Robot Inmoov completo

2.1.2 SolidWorks

SolidWorks [5] es un programa de diseño 3D asistido por ordenador propiedad de la empresa *Dassault Systèmes SolidWorks Corp.* Este programa creado en 1997 brinda la oportunidad a más de 2.170.100 ingenieros en el mundo de diseñar, simular, publicar y administrar diseños 3D. Entre sus virtudes destacan:

- Diseño 3D intuitivo.
- Facilidad de exportación a planos 2D.
- Rapidez en el diseño.
- Simulaciones con parámetros reales del entorno.
- Integración completa del sistema gracias a herramientas como SolidWorks Electrical, SolidWorks CAM o SolidWorks Simulation.
- Apoyo de la comunidad y fabricantes.

Este programa permite realizar diseños de piezas y posteriormente unirlas todas bajo un mismo ensamblaje. De esta forma el diseño es modular y escalable, se empiezan diseñando módulos e integrando uno a uno y finalmente se obtiene un sistema completo. El gran beneficio del empleo de este tipo de herramientas es la previsualización del prototipo antes de la fabricación, ahorrando numerosos costes en el proceso de diseño y prototipado.

La integración de distintas suites o módulos del programa han hecho posible que se puedan integrar los sistemas completos dentro de este programa. Gracias al módulo de simulación se pueden simular cargas tensionales, movimientos de fluidos, choques de elementos y otros efectos como deformaciones o admisión de tolerancias. Con el módulo de renderización se pueden crear imágenes fotorrealistas que aportan una visión preliminar del prototipo bastante aproximada. El módulo “Electrical” permite la creación de planos eléctricos y asociación con los componentes físicos en 3D, se utiliza principalmente en la creación de cuadros eléctricos. Finalmente, el módulo “Plastics” ayuda en la predicción de defectos durante el proceso de fabricación de plásticos ya sea en procesos de fundición.

2.1.3 Arduino Uno

Arduino [6] es una empresa dedicada a la creación de microcontroladores. La empresa tiene como filosofía crear herramientas bajo licencias de uso público. Esto quiere decir que cualquier persona o empresa puede modificar y fabricar su propio microcontrolador con los esquemas hardware de Arduino.

Arduino cuenta con múltiples versiones de microcontroladores, desde los más pequeños con microprocesadores AVR de 8 bits pensados para aplicaciones no muy exigentes con un consumo de energía mínimo, hasta microcontroladores más potentes que cuentan con microprocesadores Cortex M3 de 32 bits.

El lenguaje de programación que emplean estos dispositivos es "Wiring" una adaptación de C para hacer la creación de software más sencilla. Arduino cuenta con su propia plataforma que sirve como editor, depurador y cargador de programas.

Arduino, gracias a su filosofía de creación de herramientas libres, cuenta con una de las mayores comunidades que existen en el mundo de los microcontroladores, con miles de proyectos y tutoriales disponibles en su página web, cualquier persona puede iniciarse en el mundo de la electrónica. Además, con la llegada del internet de las cosas se han unido nuevos usuarios con la idea de fabricarse sus propios dispositivos conectados.

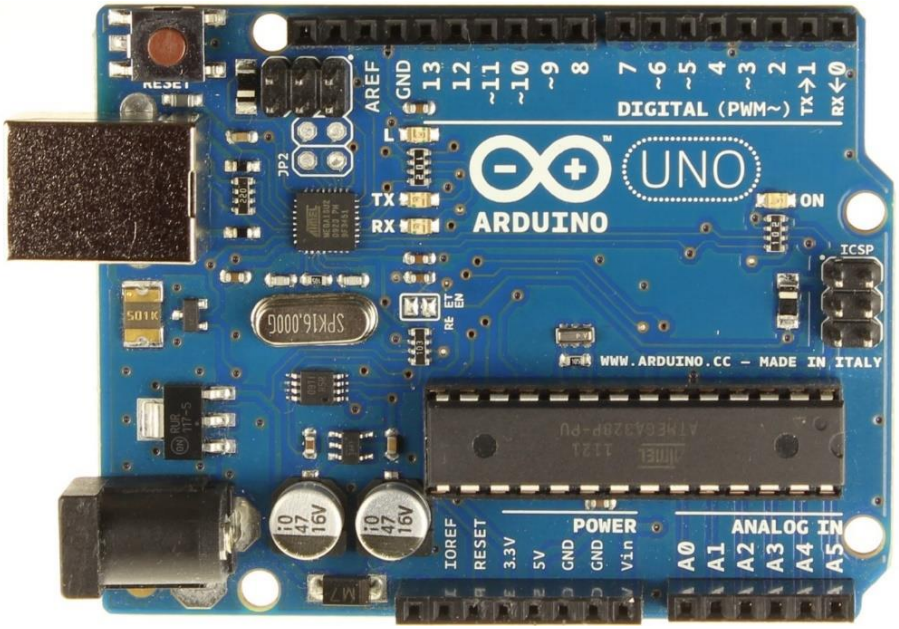


Ilustración 2-2: Arduino UNO R3

En este proyecto se ha empleado el microcontrolador Arduino UNO R3. Este microcontrolador es el estándar. Destaca por su versatilidad a un precio económico, aunque con algunas carencias como la falta de puerto Ethernet o Wifi. Esta versión de microcontrolador cuenta con:

- 14 Pines I/O digitales
- Salidas PWM
- Entradas analógicas
- Memoria flash de 32 kB
- Procesador de 8 bits ATMega328P a 16 MHz

Se ha elegido este microcontrolador debido a su facilidad de programación. Existen múltiples librerías destinadas al control de Servomotores y comunicaciones.

El precio también ha sido determinante en la elección del microcontrolador. Debido a su reducida potencia de procesamiento, Arduino UNO R3 tiene un precio bastante inferior a los microcontroladores estándar del mercado como, por ejemplo, NXP.

2.1.4 MakerBot

MakerBot es una empresa fundada en 2009 dedicada a la creación de hardware y software relacionado con la impresión 3D. MakerBot fue una empresa pionera en su segmento, desarrollando el hardware y el software con licencias de uso público, de forma que cualquier usuario o empresa que adquiriese sus productos pudiesen modificarlos o crear sus propias versiones. El mercado de las impresoras 3D se ha visto enormemente beneficiado por esto.

La impresora utilizada en este proyecto ha sido la Replicator. Una de las primeras impresoras 3D. Esta impresora, que debe ser montada por el usuario, cuenta con unas características dentro de la media del mercado de las impresoras 3D como son:

- Tamaño del extrusor 0.2-0.4 mm.
- Volumen de impresión 20 x 15 x 20 cm.
- Utilización de 2 extrusores.
- Precisión en Z de 0.05 mm.
- Cama caliente.

El tamaño del extrusor determina el tamaño del filamento que se puede utilizar y la precisión que se puede llegar a obtener en el plano XY.

El volumen de impresión indica las dimensiones máximas que puede imprimir la impresora 3D. No es recomendable llevar a la impresora hasta estos límites ya que puede producir errores de impresión.

La utilización de dos extrusores permite crear diseños con dos materiales distintos, o de distintos colores a la vez.

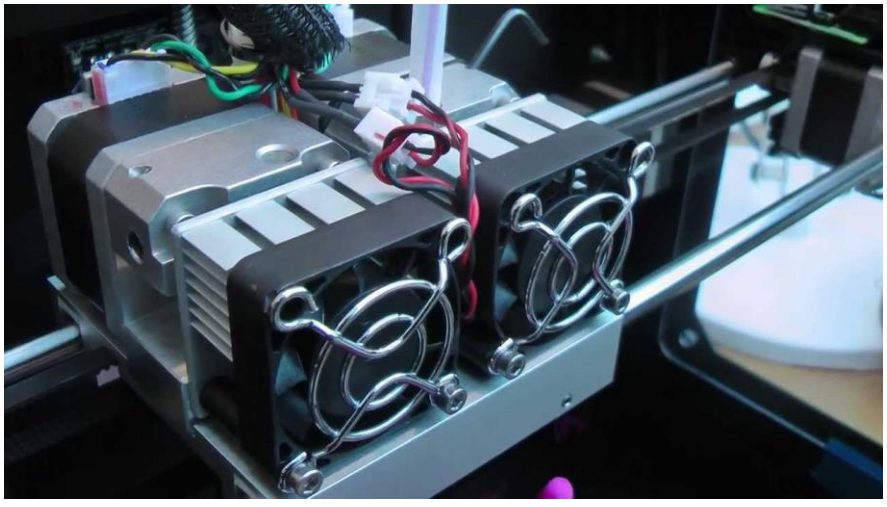


Ilustración 2-3: Dos Extrusores trabajando a la vez

Una de las características más importante es la precisión en el eje Z. Esta precisión determina el número de capas que debe emplear la impresora 3D. Cuanto mayor sea la precisión más capas habrá que generar y la apariencia externa será más similar al diseño. Como inconveniente, la impresora al tener que realizar más capas tardará más tiempo, con todo lo que ello implica, mayor coste, mayor estrés para los componentes, tiempo, etc.

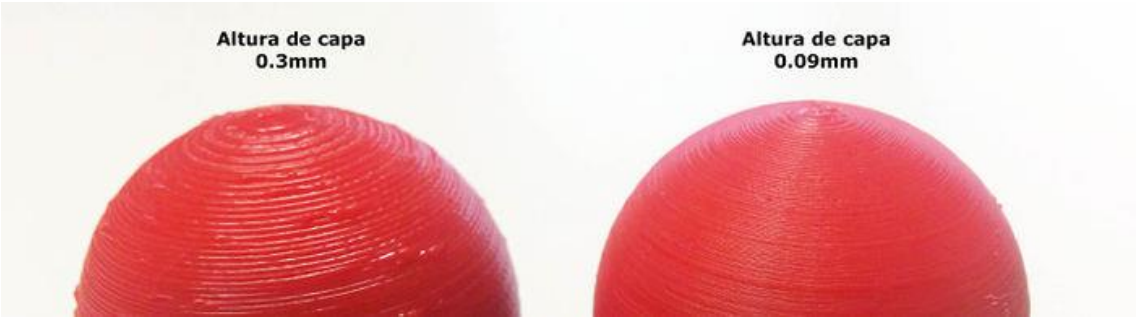


Ilustración 2-4: Diferencias entre la precisión de capa

La cama caliente es la base sobre la cual se imprime. Cuando se utilizan ciertos materiales como PLA, el plástico sale del extrusor a 200 ºC, al entrar en contacto con una superficie a temperatura ambiente se producirá un efecto de contracción rápidamente que desencadenará en una impresión defectuosa. Además, mantener una superficie caliente es muy recomendable para la adhesión de la pieza durante la impresión.

El software Replicator genera un archivo GCode, un lenguaje de programación universal ampliamente utilizado en máquinas CNC. Este lenguaje utiliza una sintaxis y una semántica específica para transmitir posiciones y configuraciones. Posteriormente realiza una conversión de este archivo GCode a un archivo de tipo ".x3g" que es propio de la impresora 3D de MakerBot.

2.1.5 IRB120

En el sector de la robótica industrial destaca la empresa ABB con su gama de robots IRB. La gama IRB [7] abarca desde brazos robóticos con poca capacidad de carga, manipuladores de herramientas, hasta robots con grandes capacidades de carga destinados a mover objetos de pesos elevados. ABB proporciona una suite de programación de estos robots. El lenguaje de programación es un lenguaje propietario de ABB denominado RAPID. RAPID sirve para programar tanto el control del brazo robótico, así como las conexiones y configuraciones del mismo. La suite, denominada "Robotstudio", no solo es una suite de programación, también permite el modelado de ciertos elementos y la simulación tridimensional del sistema que se desee implementar.

Ilustración 2-5: Brazo robótico IRB120

Desde ABB proporcionan los archivos CAD del modelo tridimensional del brazo robótico. Esto es de gran ayuda cuando se desea fabricar una herramienta para el IRB120 o cuando se desea diseñar una estación de trabajo fuera del entorno de trabajo de Robotstudio.

2.2 *DISEÑO DE NUEVAS PIEZAS*

Se han diseñado cinco piezas nuevas. Durante el proceso de diseño se ha perseguido la funcionalidad del sistema y el tamaño del mismo. La mano Inmoov original disponía de un gran espacio para ubicar los motores. En este diseño, sin embargo, el espacio ocupado es crítico. Las piezas creadas son:

- **Cama de servos:** Servirá como apoyo para colocar los servomotores encargados de realizar el movimiento. En la mano original Inmoov estos motores se colocan en el antebrazo. En este diseño el espacio utiliza tiene que ser menor.
- **Agarre motor:** Estas piezas son las encargadas de unir los cables tensores con el eje del motor. Para hacer más eficiente el giro es necesaria una pieza que obligue al cable tensor a realizar una curvatura más amplia que en la mano original Inmoov. La pieza debe ser optimizada para aprovechar el máximo espacio disponible.
- **Soporte Mano:** Esta pieza está diseñada para unir todas las piezas con la mano Inmoov.
- **SPT Servo:** Esta pieza está diseñada específicamente para unirla al brazo robótico IRB120. La pieza consta de dos lados, uno de ellos diseñado para acoplarlo en prácticamente cualquier objeto, el otro diseñado para el acople específico del robot IRB120.

Las operaciones básicas en la creación de piezas han sido la extrusión y el corte. Mediante estas dos operaciones se pueden realizar la mayoría de las piezas.

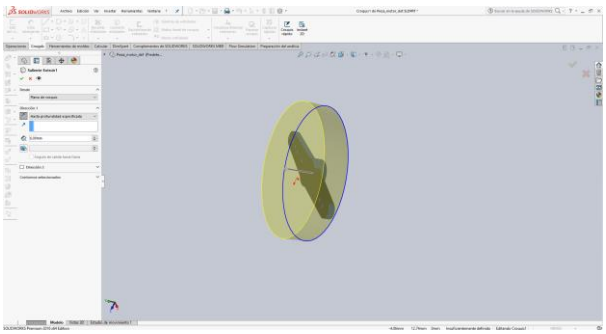


Ilustración 2-6: Herramienta de extrusión de SolidWorks

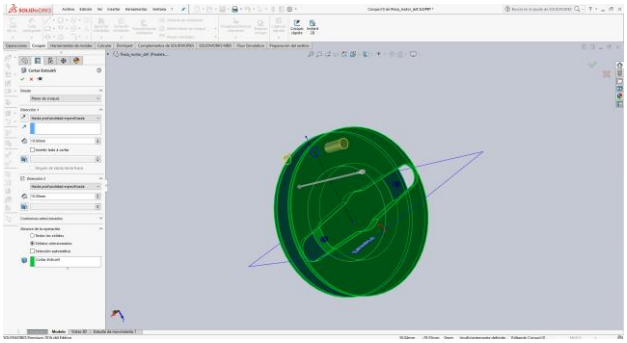


Ilustración 2-7: Herramienta de corte de SolidWorks

2.2.1 Cama de Servos

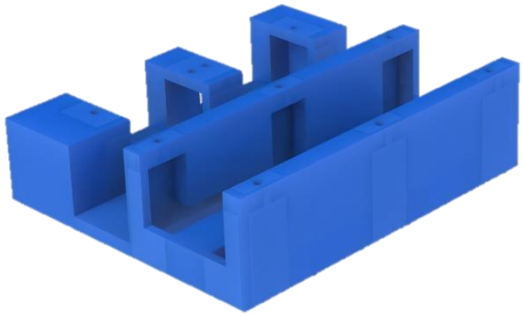


Ilustración 2-8: Vista general de "Cama de servos"

La cama de servos posee ranuras específicas para el guiado de los cables de los Servomotores. En anteriores diseños se comprobó que los cables quedaban aprisionados entre la pieza “Soporte Mano” y la “Cama de servos”, con este guiado se ha evitado este hecho. Incorpora también los taladros específicos para el acople de los motores y de la pieza “Soporte Mano”. Gracias a estos taladros ya incluidos en el modelo de impresión se consigue que estén correctamente alineados.

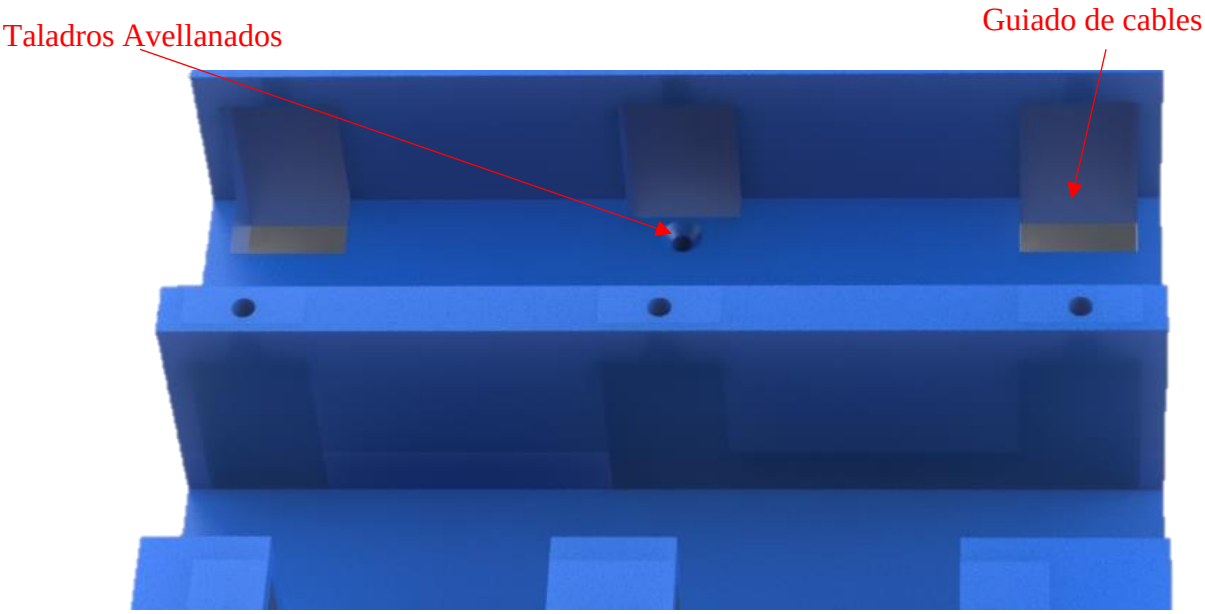


Ilustración 2-9: Taladros avellanados y guiado de cables en "Cama de servos"

Los taladros de unión con la pieza “Soporte Mano” deben ser avellanados para permitir colocar el motor central encima de este.

La pieza consta de dos alturas para evitar el cruce de los cables tensores.

2.2.2 Agarre motor

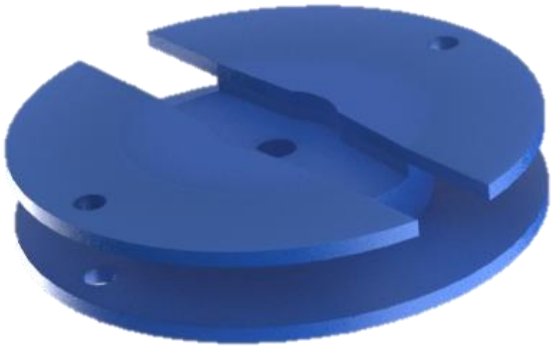


Ilustración 2-10: Vista general de "Agarre motor"

Esta pieza ha sido diseñada para permitir unir los adaptadores que incluyen los motores seleccionados con los cables tensores. La pieza del motor se acopla en el centro de esta pieza y es atornillada a través del taladro central uniendo de esta manera el eje, el adaptador incluido con el motor y esta pieza "Agarre motor". Los cables tensores rodean la pieza por el interior de la ranura y son amarrados a los taladros laterales mediante tornillos o nudos. La pieza ha sido diseñada doblemente simétrica para evitar posiciones correctas o incorrectas, solo hay una posición posible.

Ilustración 2-11: Ensamblaje de motor y "Agarre motor"

2.2.3 Soporte Mano

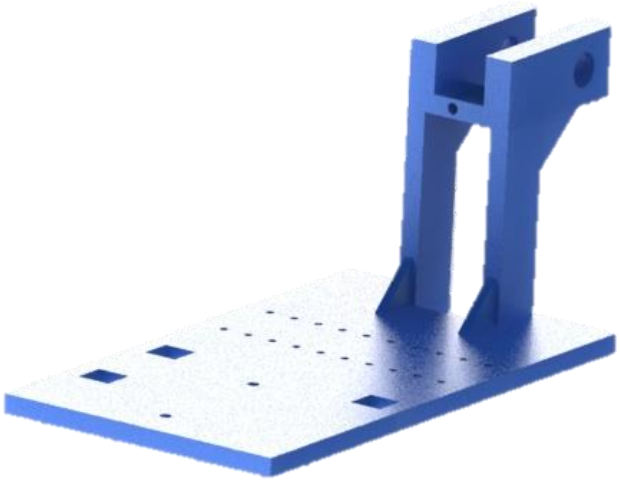


Ilustración 2-12: Vista general de "Soporte mano"

Esta pieza consta de cinco taladros principales. Los dos más grandes están destinados a albergar un perno que sujete la mano Inmoov. Los dos taladros situados en la parte inferior se encargarán de unir la pieza “SPT Servo” y la pieza “Cama de servos” mediante el paso de un par de tornillos. Se han colocado dos nervios para aportar una mayor rigidez a la estructura. El ultimo taladro ha sido diseñado de cara a implementaciones futuras, en concreto una pieza que permita el guiado de cables y la tensión de los mismos. La pieza también consta de ranuras especiales para guiar los cables de los motores.

La pieza ha de ser bastante alta debido a la altura de los motores. Esta altura supone un problema cuando se exigen varios movimientos de motores pues es la pieza que soporta la mayor tensión. Por este motivo es recomendable que cuando se imprima en 3D esta pieza, la impresión tenga un alto porcentaje de llenado.

Esta pieza ha sido diseñada pensando en una futura mejora, la inclusión de tensores de cables. Los veinte taladros extras son para este sistema tensor que no son objeto de este trabajo.

2.2.4 SPT Servo

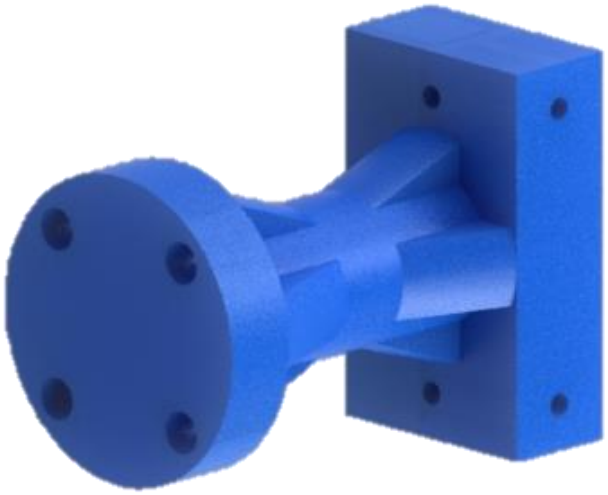


Ilustración 2-13: Vista general de "SPT Servo"

Esta pieza consta de cuatro taladros dispuestos en una matriz circular y de dos taladros en cada una de las caras de un prisma exceptuando la superior e inferior. Los taladros dispuestos en el círculo son los encargados de amararse al IRB120 mediante el uso de pernos. Este tipo de agarre es el que se está utilizando con todas las herramientas fabricadas en la universidad. El resto de taladros son para distintas aplicaciones dependiendo de si el prisma está embutido en la pieza o no. En este diseño se emplearán los taladros que no embuten el soporte en la otra pieza.

2.2.5 Ensamblaje final

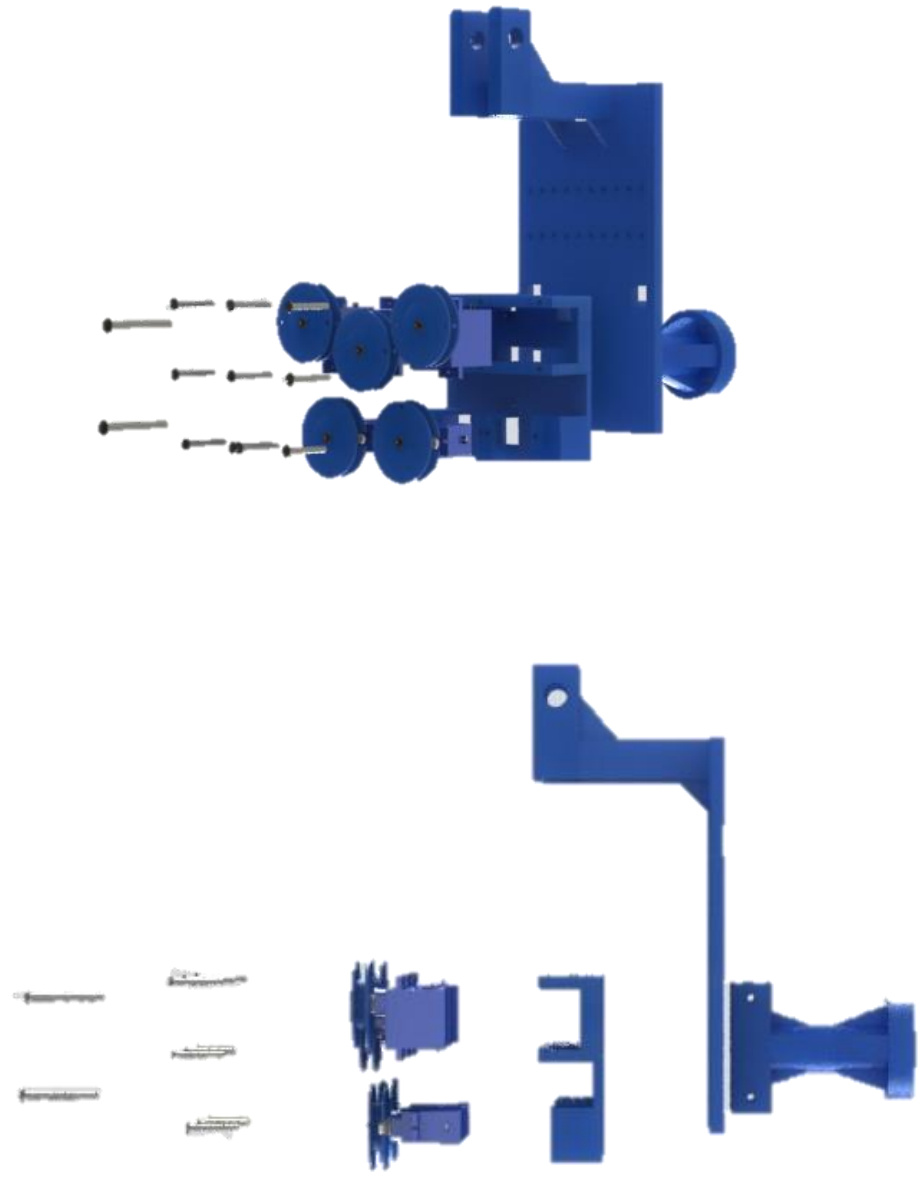


Ilustración 2-14: Vista general de ensamblaje de piezas diseñadas

En estas imágenes se puede observar cómo quedaría el ensamblaje final de las piezas diseñadas.

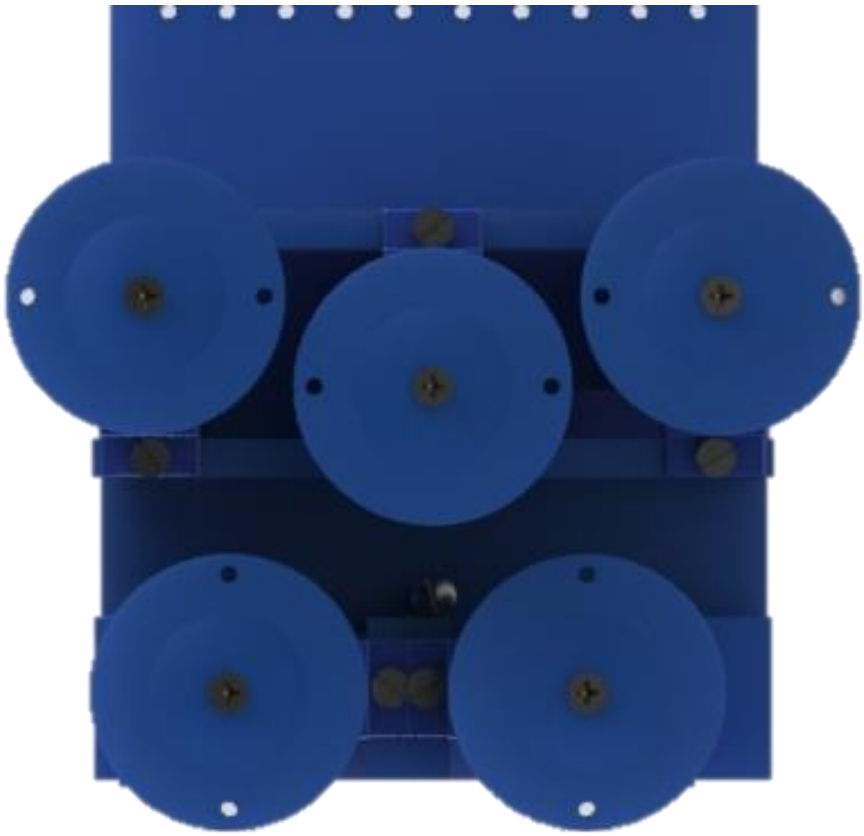


Ilustración 2-15: Distancia entre motores

En esta imagen se aprecia lo ajustados que se encuentran los motores. Este diseño no es el más eficiente para los motores. Los motores se encuentran embutidos y muy próximos unos de otros favoreciendo que el calor se propague rápidamente. Esto no supone un problema salvo en situaciones en las que se exija mucha carga a los motores o se realicen múltiples movimientos de forma continuada.

Con este ensamblaje termina la sección de creación de piezas. En el capítulo del manual se indica cómo realizar el ensamblaje completo. En el capítulo de planos se encuentran los planos de las piezas con todas las cotas de valor para su consulta y revisión.

2.3 ENSAMBLAJE DE LA MANO EN SOLIDWORKS

El ensamblaje de la mano virtual se realiza en SolidWorks. El principal problema es que la página de Inmoov proporciona sus archivos en STL. El tipo de archivo STL es ideal para imprimir y compartir un modelo en 3D, sin embargo, no es un buen formato con el que trabajar en una herramienta CAD. Por ello antes de ensamblar la mano en SolidWorks, se debe realizar un proceso de transformación de formato STL a formato SLDPRT. En esta sección se detalla el proceso de transformación y el posterior ensamblaje de la mano en forma virtual.

Con el objetivo de situar al lector a lo largo del proceso, se adjunta un flujograma del proceso de creación del ensamblaje.

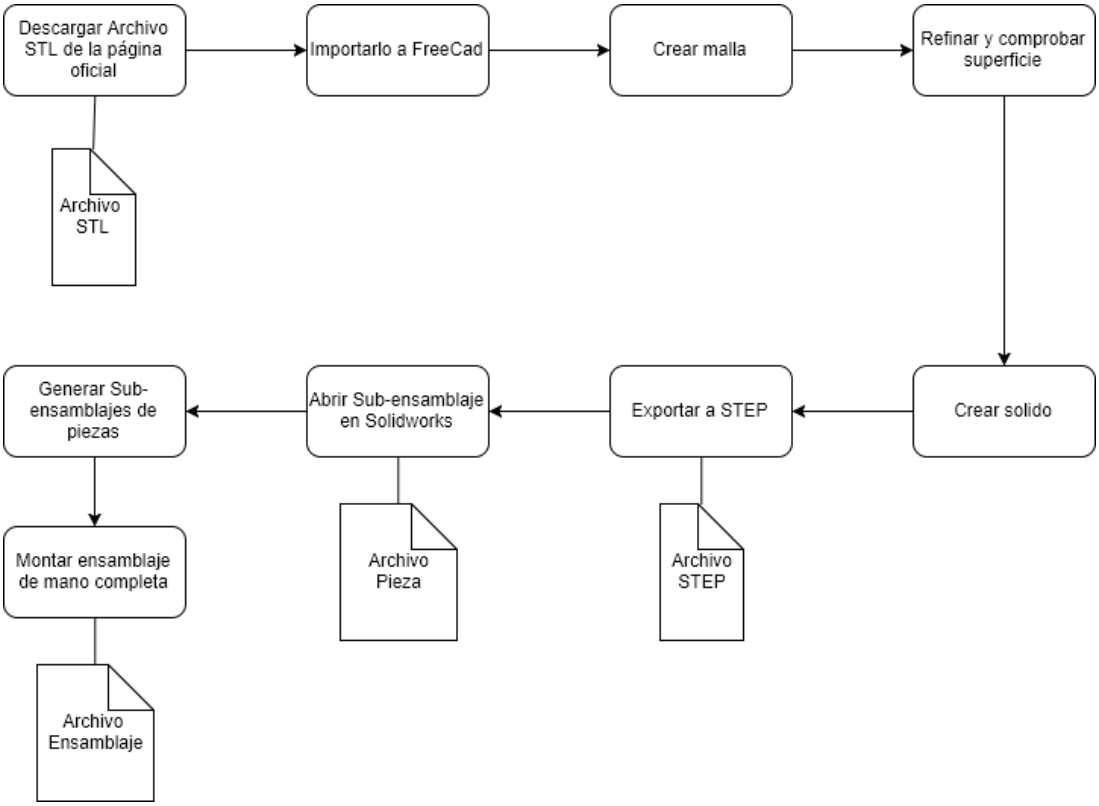


Ilustración 2-16: Proceso de transformación de STL en ensamblajes

2.3.1 Descarga de ficheros STL y transformación

Los archivos STL se pueden descargar de la página:

http://inmoov.fr/inmoov-stl-parts-viewer/?bodyparts=Right-Hand

En este diseño no serán necesarios los archivos:

- Robcap3V2STL.
- Robpart2V4STL.
- Robpart3V4STL.
- Robpart4V4STL.
- Robpart5V4STL.
- ArduinosupportSTL.

Los siguientes archivos son opcionales, solamente aportan una mayor estética:

- Coverfinger1STL.
- Topsurface6STL.
- TopsurfaceUP6STL.

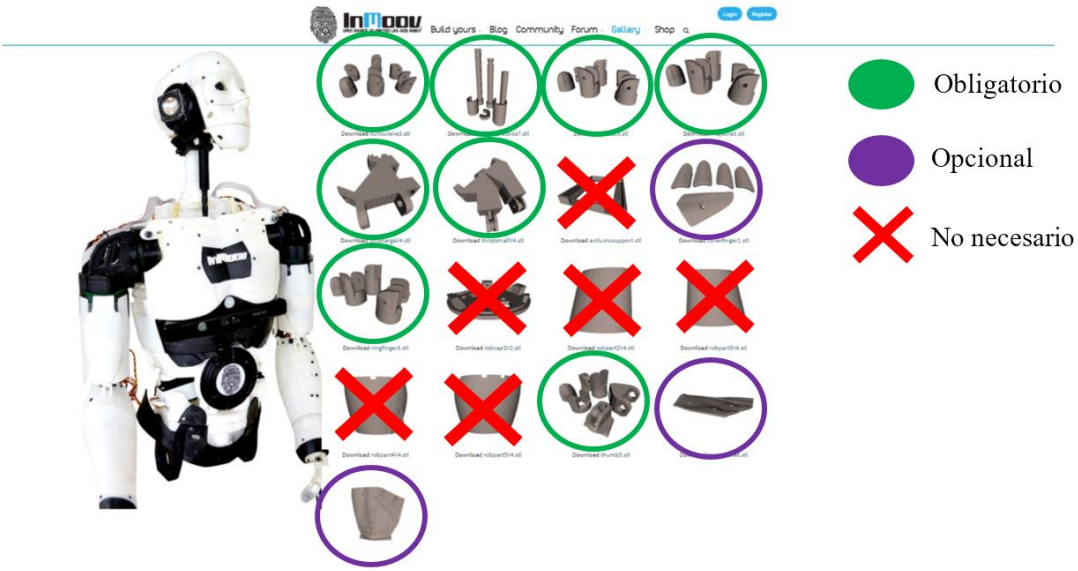


Ilustración 2-17: Listado de piezas para descargar

Los archivos STL ya están preparados para imprimir en 3D mediante el proceso de impresión que se explica en la siguiente sección Como se ha indicado anteriormente, si se desea trabajar con el modelo 3D en un programa CAD, primero hay que realizar un proceso mediante el cual transformar estos archivos.

En este proceso se utilizará el programa FreeCAD. Este programa, aun siendo menos potente que SolidWorks, posee una característica muy interesante, la capacidad para

reconocer y coser superficies de una forma sencilla. Esto evitará numerosos errores cuando la pieza se importe en SolidWorks.

El proceso es el siguiente:

1. Abrir el archivo STL que se desee transformar.
2. Establecer el “Escenario” en “Part”. Para ello, desplegar el menú “Ver” y en la sección “Escenario” seleccionar “Part”.

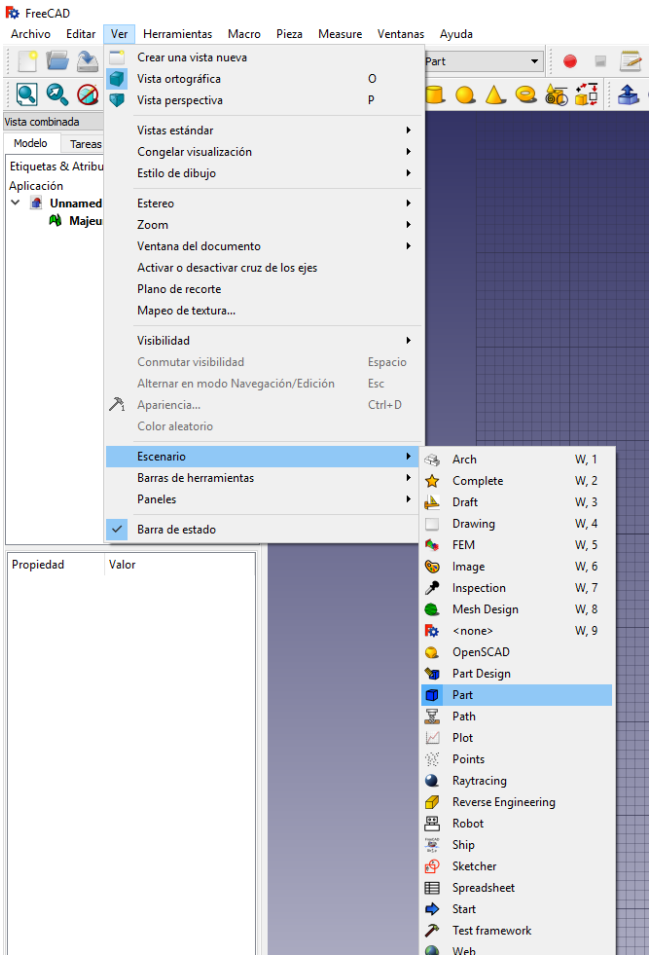


Ilustración 2-18: Cambiar escenario en FreeCAD

3. Crear la forma de malla. Seleccionar la pieza que se desea transformar y en el menú “Pieza” seleccionar “Crear forma de malla” y aceptar la tolerancia por defecto.

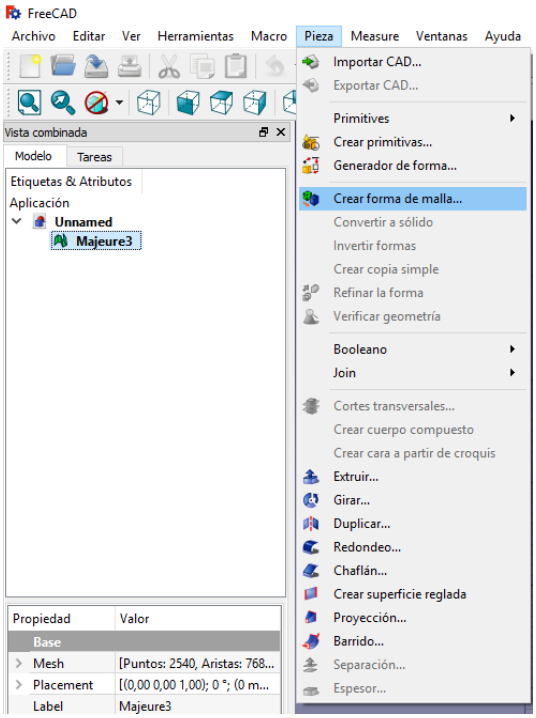


Ilustración 2-19: Obtener malla en FreeCAD

4. Refinar y verificar la forma. Este proceso, también es conocido como "Simplificar", convierte los triángulos coplanarios en superficies planas y comprueba que el resultado sea una superficie cerrada. Para ello, en el menú "Pieza" seleccionar "Refinar forma" y a continuación "Verificar geometría".

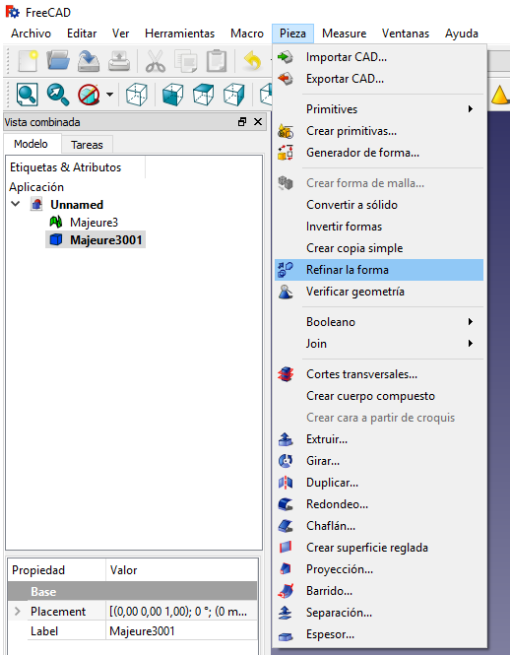


Ilustración 2-20: Refinar forma en FreeCAD

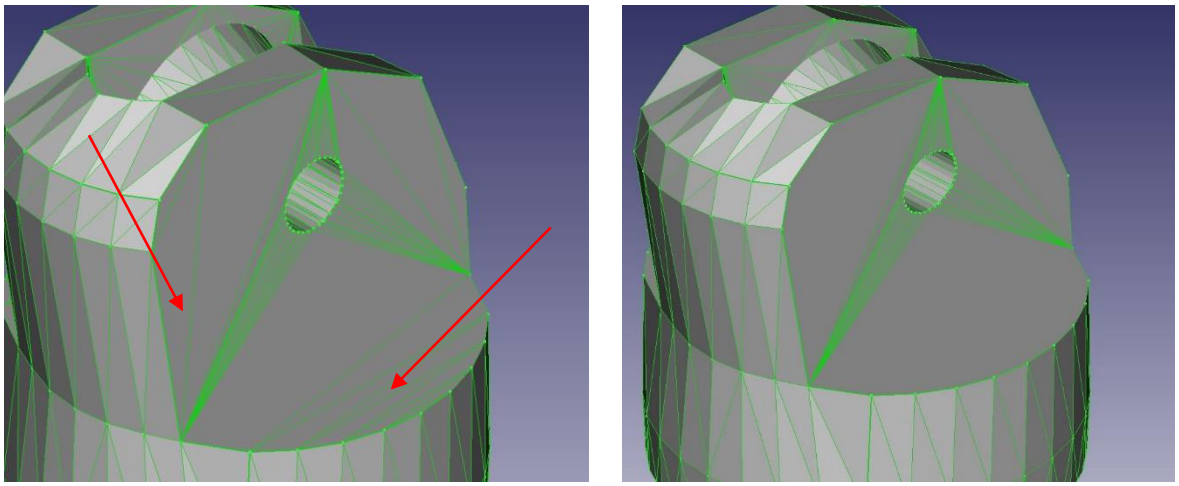


Ilustración 2-21: Beneficios de refinar forma

En estas imágenes se pueden observar cómo se han transformado ciertos triángulos coplanarios en superficies planas.

5. Convertir la pieza en sólido. En el menú “Pieza” seleccionar “Convertir en sólido”.
6. Exportar a un tipo de archivo apto para trabajar con programas CAD. En el menú “Archivo” seleccionar “Exportar” y guardar con el formato “STEP”.

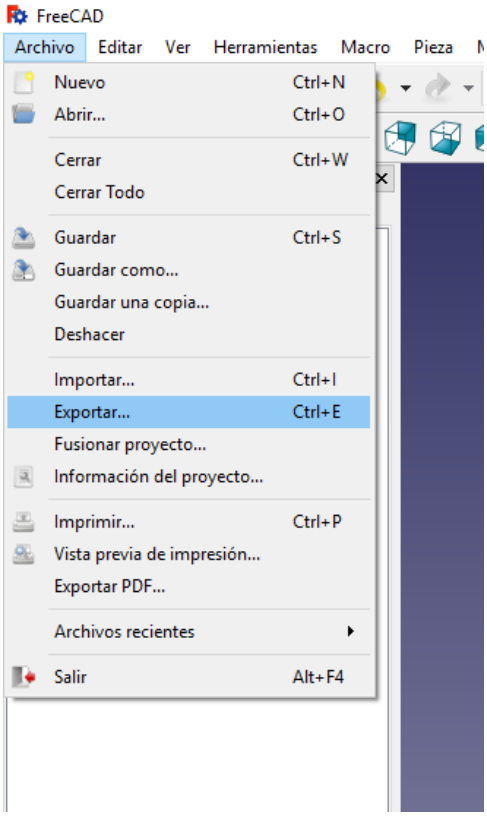


Ilustración 2-22: Exportación a pieza CAD

2.3.2 Creación del ensamblaje en SolidWorks

1. Abrir el archivo STEP generado con SolidWorks.
2. El archivo generado se abrirá en SolidWorks como un ensamblaje de los distintos sólidos.

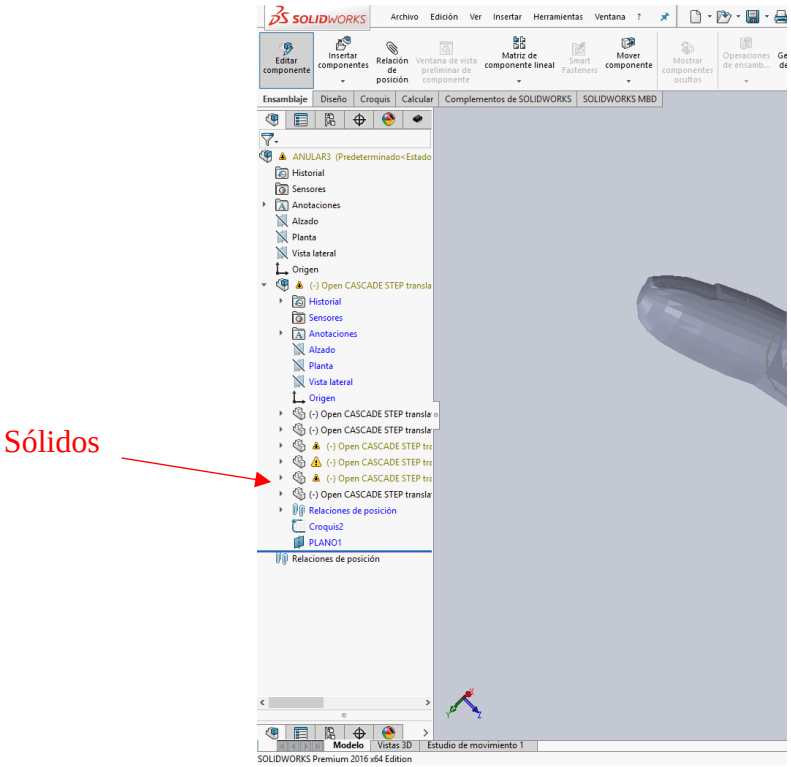


Ilustración 2-23: Solidos en ensamblaje generado en FreeCAD

3. Abrir el sub-ensamblaje y mover las piezas para obtener un modelo similar a un dedo. Esto se puede realizar mediante el uso de relaciones de posición o moviendo las piezas manualmente.
4. Realizar estas operaciones con todas las piezas provenientes de archivos STL.
5. Guardar todos los ensamblajes y piezas generadas en la misma carpeta.
6. Abrir SolidWorks y crear un nuevo ensamblaje. Para ello en el menú “Archivo” seleccionar “Nuevo” y posteriormente “Ensamblaje”.

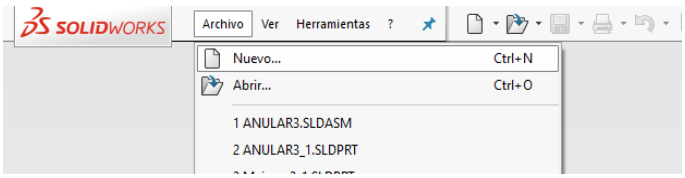


Ilustración 2-24: Menú Abrir en SolidWorks

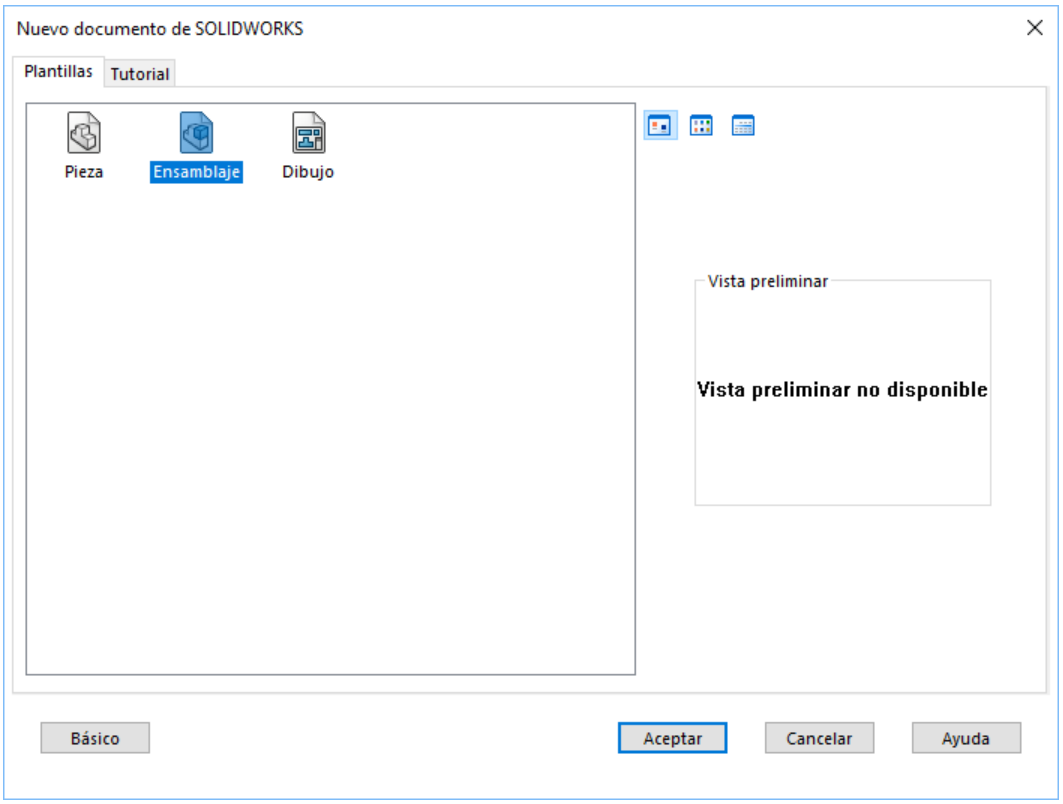


Ilustración 2-25: Menú selección de nuevo documento en SolidWorks

7. En “Pieza/Ensamblaje para insertar” buscar los archivos generados anteriormente e ir agregándolos uno a uno.

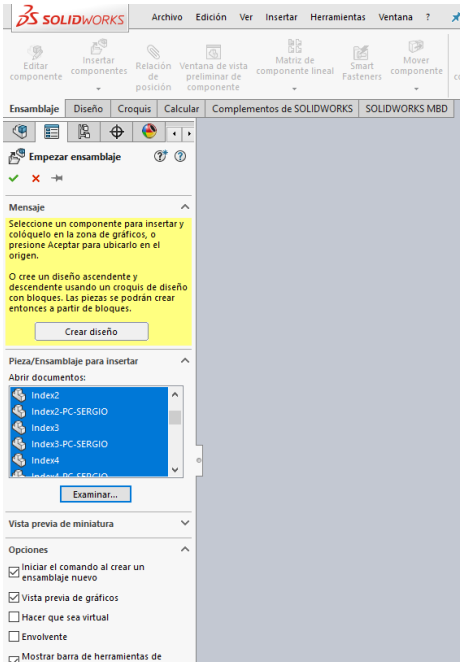


Ilustración 2-26: Seleccionar piezas para insertar

8. Mediante relaciones de posición o simplemente moviendo los componentes, construir la mano de una forma lógica. Se entiende que, con la descripción del proceso de montaje real detallado en la siguiente sección, el lector podrá realizar el ensamblaje virtual sin ningún problema.

Una vez generado el modelo de la mano, se puede utilizar para generar imágenes renderizadas fotorrealistas, simulaciones de movimientos, creación de planos, servir de base para nuevas piezas, etc.

Se debe obtener algo similar a esto:

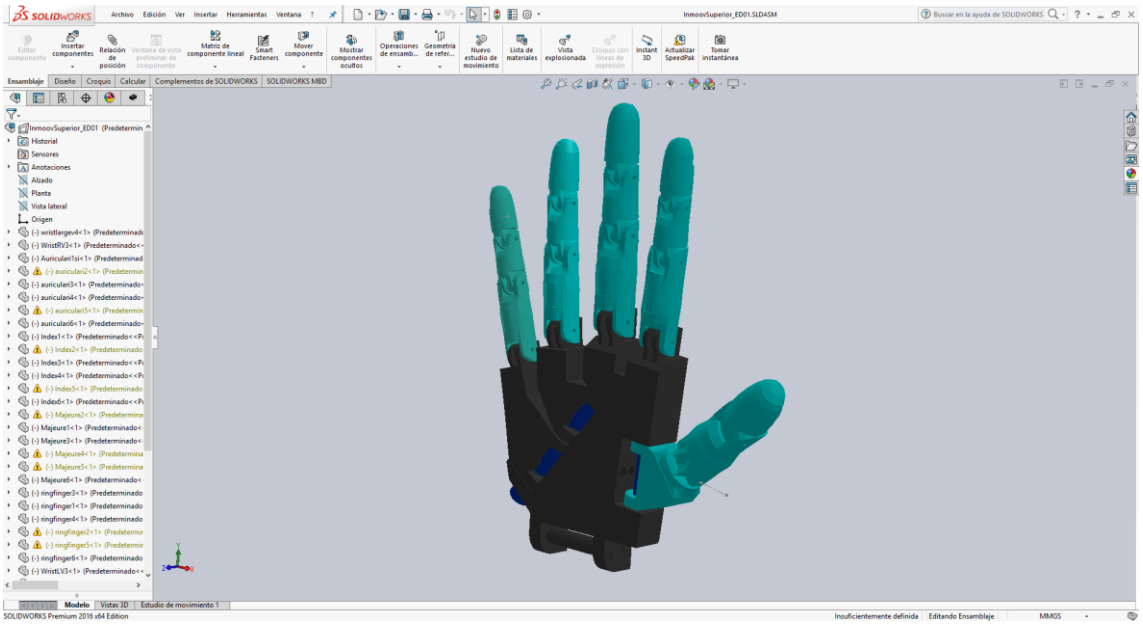


Ilustración 2-27: Ensamblaje completo en 3D

Es posible que a la hora de realizar algunas conversiones de STL a archivos STEP, SolidWorks encuentre algunos errores de construcción que manifiesta en forma de "Warnings". Estos "Warnings" no suponen ningún problema a la hora de realizar las operaciones que se han descrito anteriormente.

2.4 IMPRESIÓN 3D

En esta sección se muestran los detalles que se deben tener a la hora de realizar una impresión en 3D. Posteriormente se detalla el proceso de impresión 3D.

Existen múltiples factores que determinan la calidad de impresión de los objetos impresos mediante impresoras 3D.

2.4.1 Materiales

El material que se va a emplear en la impresión determina en gran medida los parámetros de la impresión, así como la calidad obtenida en la misma. Existen numerosos materiales para imprimir en 3D, sin embargo, hay ciertos materiales que son ampliamente utilizados como el PLA, ABS y el Nylon.

PLA

El PLA o Acido Poliláctico es un plástico biodegradable que se obtiene de distintas fuentes naturales como del almidón de maíz, el azúcar de caña o el trigo. Este plástico destaca entre los demás para su aplicación en impresiones 3D debido a:

- Es un material biodegradable.
- Su temperatura de límite viscoso no es muy elevada (60 ºC).
- Barrera contra gases y grasas.
- Velocidad de impresión elevada.
- Gama de colores.
- Precio reducido.

Estos factores convierten al PLA en un material ideal para el prototipado rápido mediante impresión 3D.

No es conveniente su uso en aplicaciones que requieran temperaturas de trabajo algo elevadas. No es un material tóxico por lo que puede ser empleado en oficinas o viviendas particulares.

ABS

El ABS o acrilonitrilo butadieno estireno es un plástico que se obtiene mediante un proceso complejo de polimerización de estireno y acrilonitrilo en presencia de polibutadieno. Es muy utilizado en el sector de la automoción debido a su gran resistencia a impactos. Destaca por:

- Excelentes propiedades mecánicas.
- Soporta mayores temperaturas que el PLA.
- Se pueden conseguir buenos acabados superficiales mediante el uso de acetona.
- Es muy ligero.
- Gran resistencia química.

El ABS se utiliza en impresiones que busquen más calidad superficial que la que el PLA es capaz de aportar. En general, es un material más caro y su impresión también es más cara debido a que necesita una mayor temperatura de fusión.

Nylon

El nylon es un material ampliamente utilizado en forma de fibras. Su formación se realiza mediante policondensación de un ácido con una diamina. Entre sus características destaca su flexibilidad y resistencia. Se emplea principalmente en medias, cuerdas, paracaídas y cepillos de dientes. Sus propiedades de ductilidad y temperatura de fusión, muy similares a los de los otros plásticos expuestos anteriormente, lo convierten en el material ideal para aportar materiales flexibles al mundo de la impresión 3D.

La impresión de piezas de nylon no es sencilla y requiere de cierta experiencia por parte del usuario y algunos ajustes avanzados de la impresión 3D.

2.4.2 Problemas comunes

Existen distintos problemas que se pueden dar durante la impresión en 3D. Para evitarlos se recomienda seguir los siguientes consejos:

- **Falta de adherencia a la cama:** Se puede dar la situación en la que las piezas se levantan y despegan de la cama caliente pudiendo causar un problema grave durante la impresión. Para evitar esto, se debe ajustar la temperatura de la cama caliente según el material con el que se esté imprimiendo. También se debe evitar la impresión de piezas que abarquen todo el largo de la cama ya que la impresora genera una base más grande que la propia pieza para asegurar la adhesión. Si estos consejos no funcionan, se pueden aplicar lacas sobre la cama caliente para dar mayo adherencia.
- **Impresión de huecos:** Si se desea imprimir huecos horizontales en la pieza hay que recordar que este tipo de impresión es de adición. Si no existe una superficie sobre la que adherir la capa, se producirá un error de impresión. Las impresoras actuales cuentan con una función llamada “Full Support” que genera estructuras para proporcionar una base a este tipo de huecos o formas.
- **Encaje de piezas:** Dependiendo de los parámetros de impresión se puede dar que la tolerancia de las piezas no sea adecuada debido a que son más grandes de lo que deberían. Esto se soluciona aplicando después un proceso de lijado.
- **Resistencia:** Es bastante común evitar la impresión de piezas completamente sólidas. Esto tiene como beneficio un ahorro en los costes del plástico utilizado, pero se pueden perder propiedades de resistencia. Normalmente se imprime con un porcentaje de llenado del 10% al 15%. Si la pieza es poco resistente se puede subir, siendo válido normalmente el 25% de llenado.

Estos han sido los principales problemas durante la impresión y las soluciones que se han encontrado.

2.4.3 Proceso de impresión 3D

El primer paso es obtener las piezas 3D. Para ello se utiliza el programa “ReplicatorG”.

1. Abrir ReplicatorG con privilegios de administrador.
2. Abrir el archivo STL que se desea imprimir. Para ello hacer click en la opción “Open…” en el menú “File”.

Ilustración 2-28: Menú abrir de ReplicatorG

3. Colocar la pieza sobre la plataforma mediante el menú derecho de “Move”.

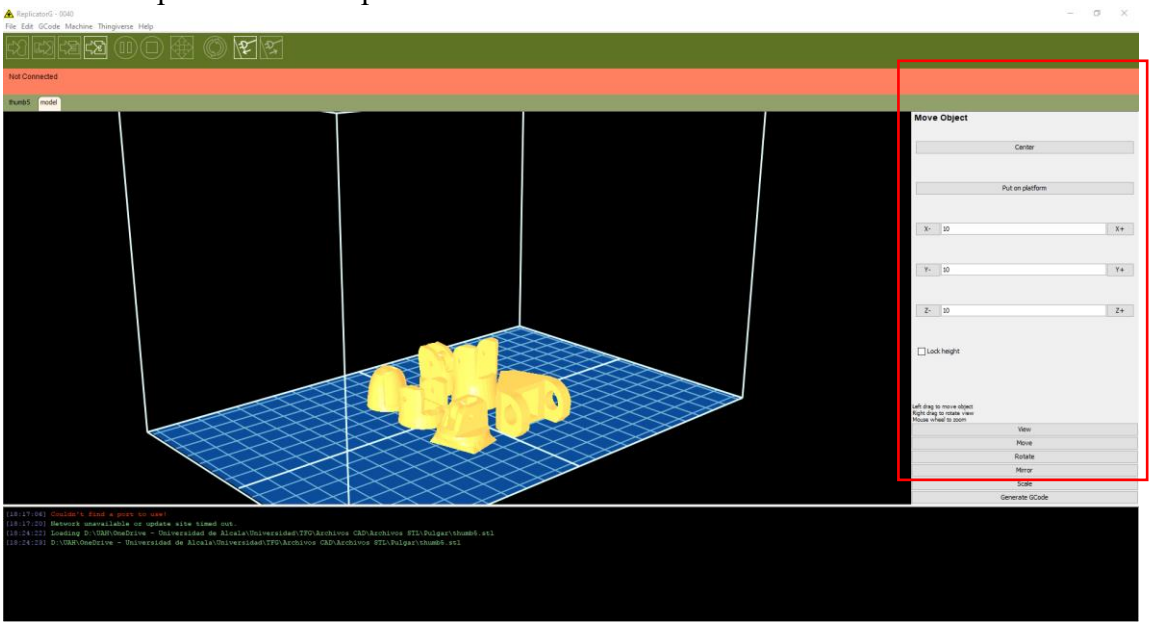


Ilustración 2-29: Colocación del modelo 3D para su impresión

4. Hacer click en “Generate GCode”.

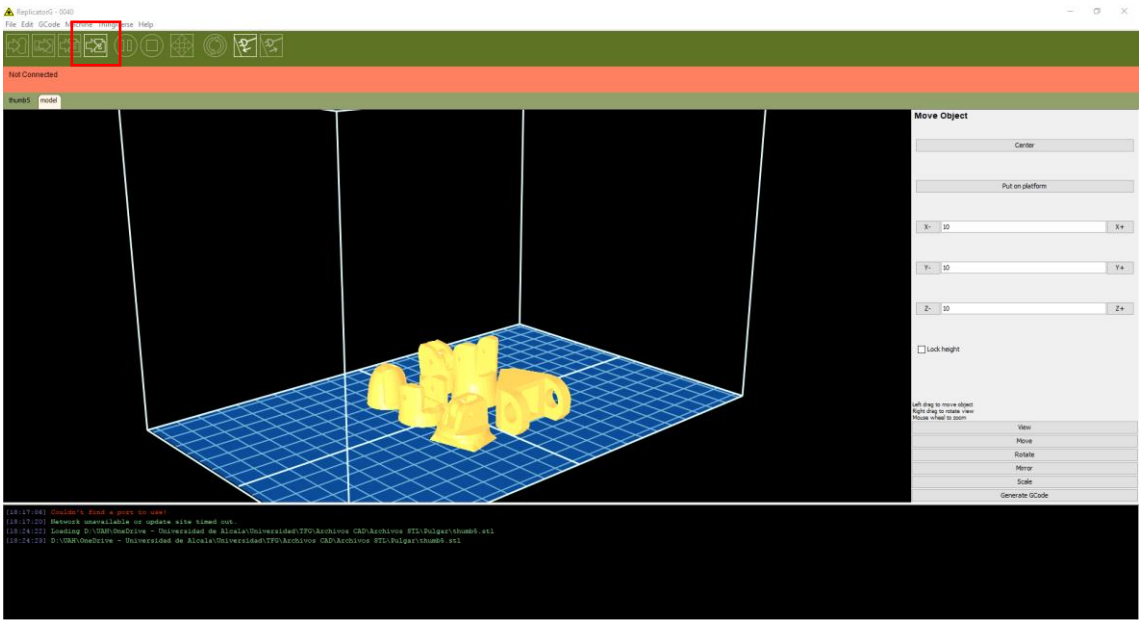


Ilustración 2-30: Botón generar "GCode"

5. Aparecerá un menú con los parámetros de impresión. Se recomienda aplicar los mismos que los que se adjuntan en esta imagen.

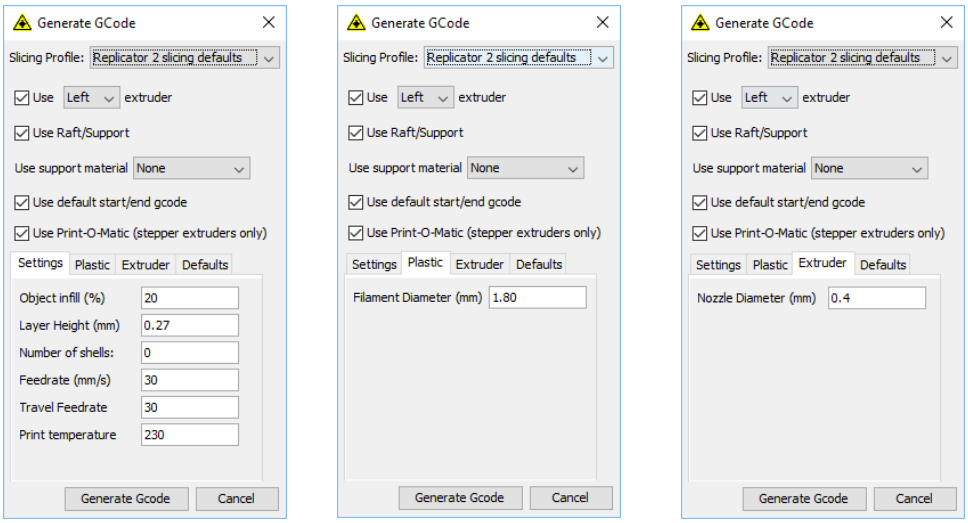


Ilustración 2-31: Parámetros de impresión

6. Hacer click en “Generate GCode” y esperar a que termine el proceso.

7. Una vez haya terminado el proceso hacer click en “Build to file for use with SD Card” y seleccionar “Tipo de archivo” como X3G al exportarlo.

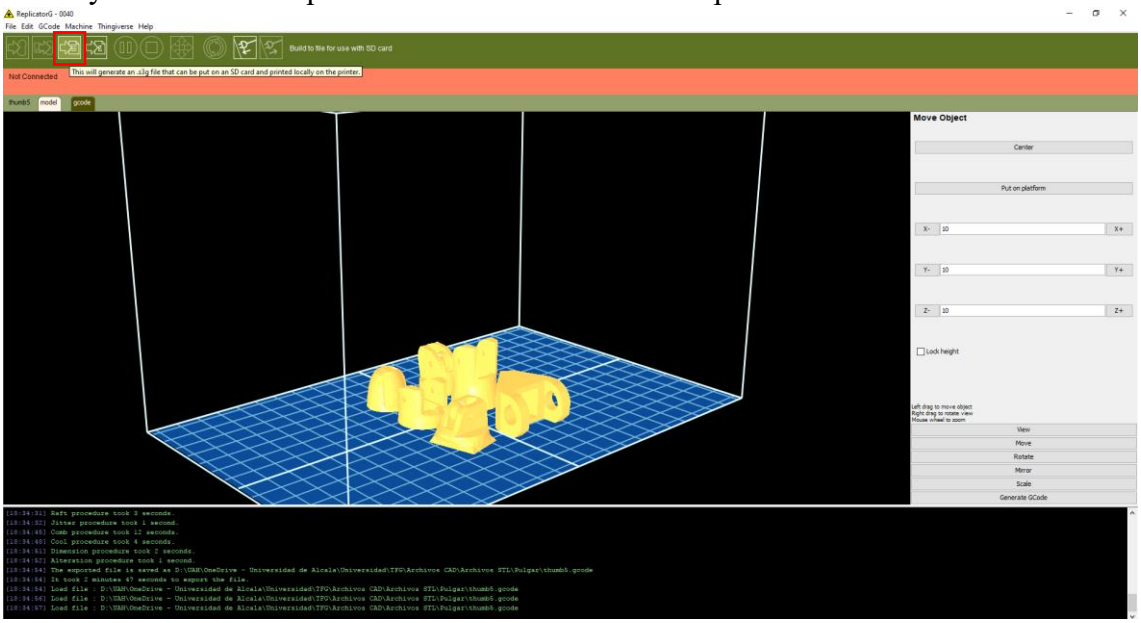


Ilustración 2-32: Botón de exportación a X3G

8. Guardar el archivo X3G en una tarjeta SD.
9. Llevar la tarjeta SD a la impresora 3D y seleccionarlo para su impresión.

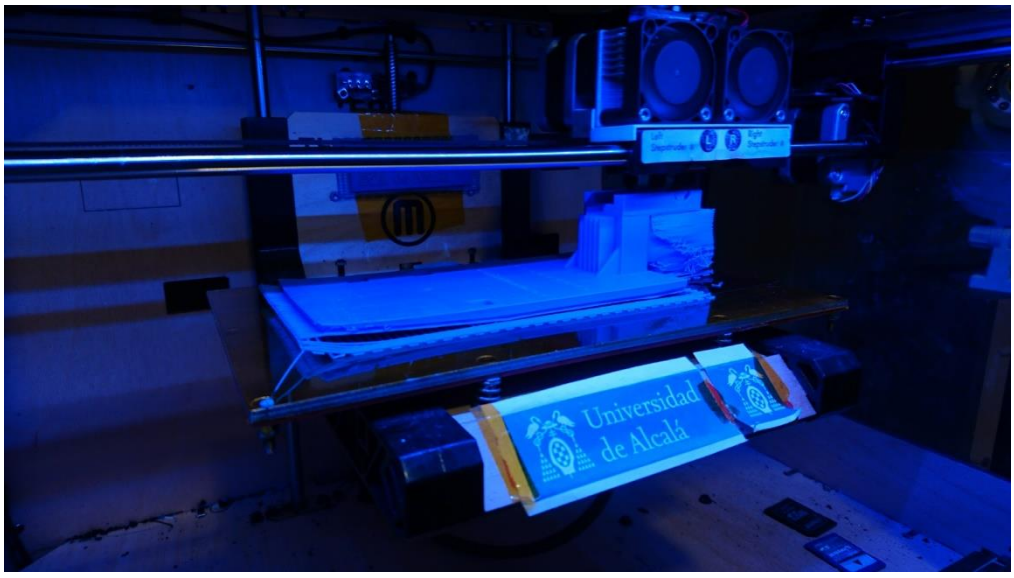


Ilustración 2-33:Impresora 3D imprimiendo piezas

10. Después de realizar la impresión es recomendable llevar a cabo un proceso de limado de las piezas.

2.5 MONTAJE DE LA MANO REAL

En esta sección se explica el montaje y ensamblaje de la mano robótica. Se explicará el montaje de un único dedo, el resto se realizan de forma similar. El resto de montaje se explicará entero.

1. Unir con pegamento las piezas intermedias, excepto la falange superior. Como referencia se puede utilizar las marcas realizadas a lo largo de las piezas.

Ilustración 2-34: Unión de falanges mediante pegamento

2. Una vez unidas estas piezas con pegamento se deben ensamblar con tornillos más pequeños que el taladro para permitir el movimiento entre si. Dado el tamaño del taladro, es posible que no se encuentren tornillos de una métrica tan pequeña y tan largos. Por ello se debe repasar cada taladro. En el diseño se utilizaron tornillos de métrica M2,5 y un taladro de métrica M3.

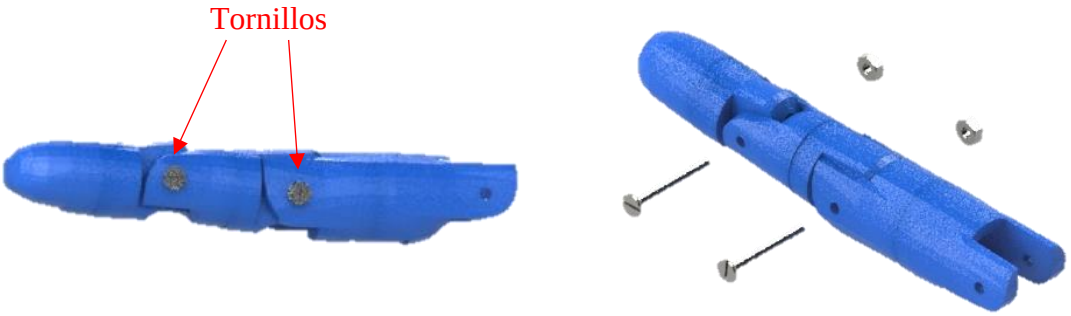


Ilustración 2-35: Ensamblaje de falanges mediante tornillos

3. Una vez montados todos los dedos, se procede a montar la palma. Para ello se utiliza uno de los tres pernos impresos en 3D.

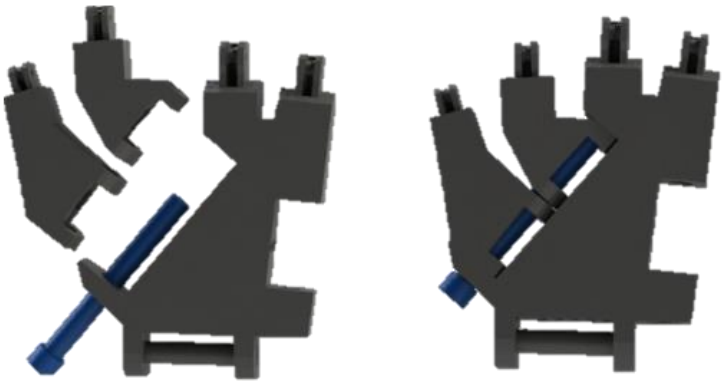


Ilustración 2-36: Ensamblaje de la palma de la mano

4. Unir los dedos con la palma mediante tornillos de forma similar a como se unieron las falanges entre si.

5. El dedo pulgar no utiliza un tornillo convencional. Utiliza uno de los tres pernos que hay que imprimir.

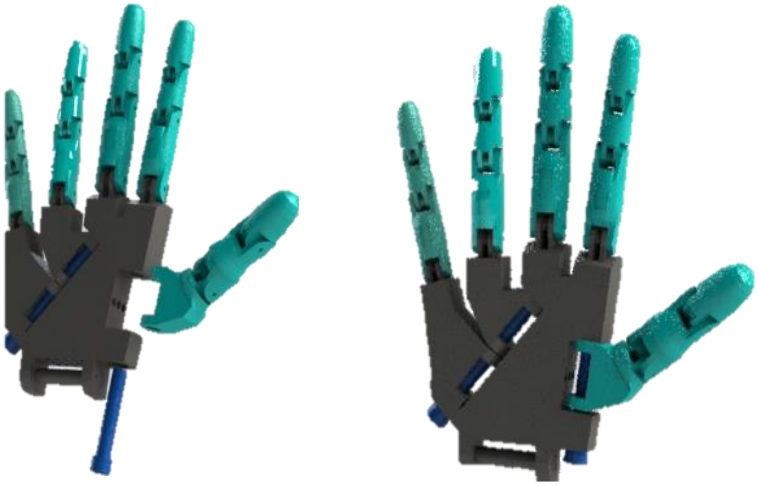


Ilustración 2-37: Ensamblaje completo de mano

6. Montar la parte que alojará los motores. Los tornillos que unen la “Cama de servos” con la pieza “Servodhand” y con “SPT Servo” deben ser al menos de métrica 3M.

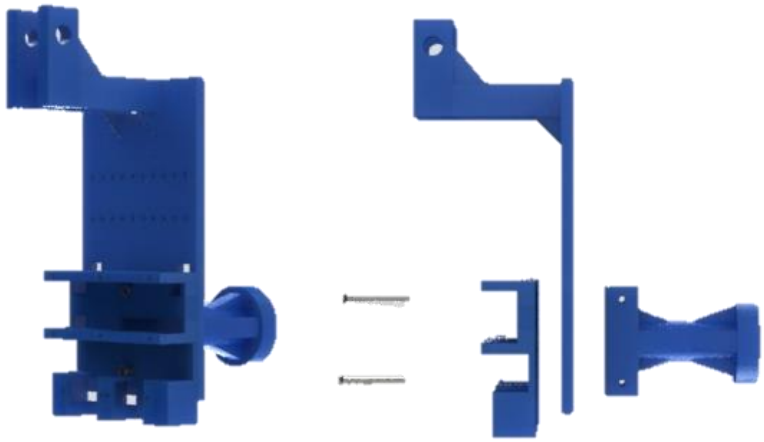


Ilustración 2-38: Ensamblaje del soporte de motores y mano

7. Unir los agarres de los motores a los motores con tornillos de métrica 2M. Hay que asegurarse de que el motor se encuentra en su posición inicial para aprovechar al máximo del rango disponible.

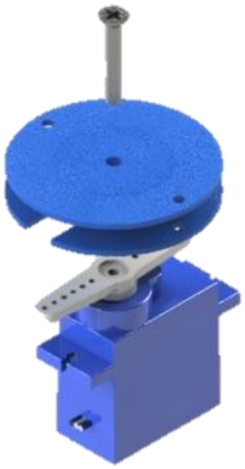


Ilustración 2-39:Ensamblaje de motor

8. Unir los motores a la pieza “Cama de servos” con tornillos de métrica 2M. Prestar especial atención al guiado de los cables. Los motores entran a presión.

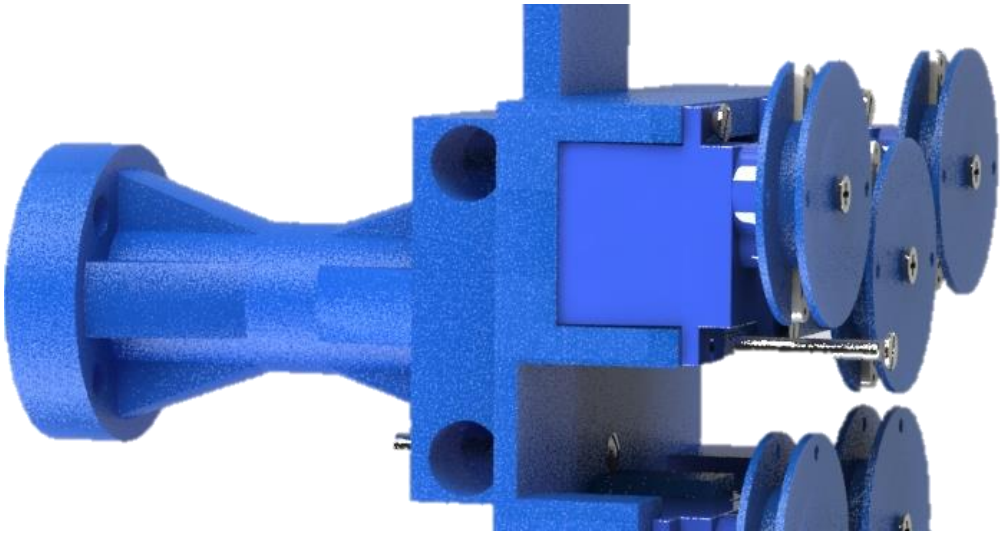


Ilustración 2-40:Ensamblaje de motores al soporte

9. En este punto se tienen dos piezas ensambladas, el soporte de los motores y la mano robótica. Unir estas dos piezas con el ultimo perno.

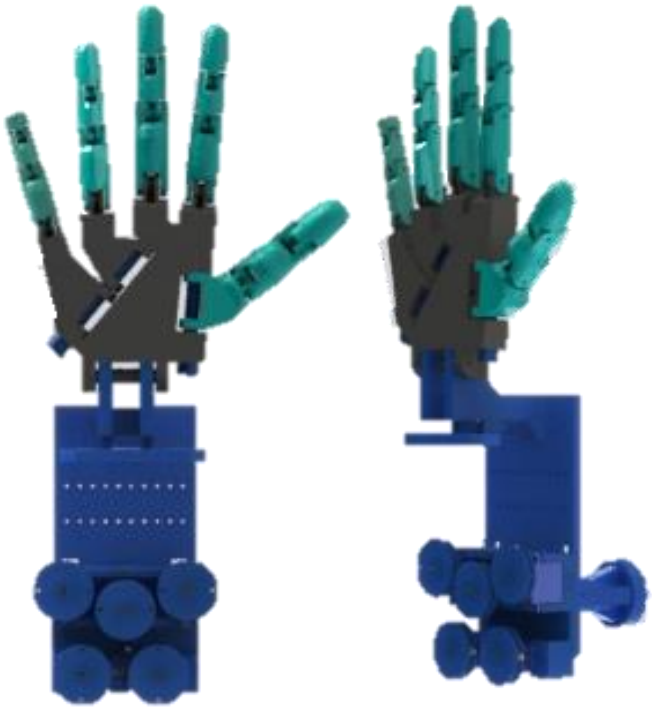


Ilustración 2-41: Unión de soporte con mano

10. Unir el guiador de cables al ensamblaje final. Se puede unir con pegamento y un tornillo atravesando la pieza “Servohand”.

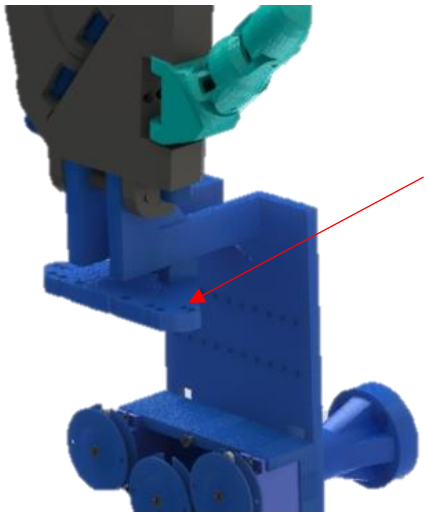


Ilustración 2-42: Pieza de guiado de cables

11. Introducir los cables tensores por las falanges superiores y guiarlos por los carriles hasta los motores ayudándose de unas pinzas.

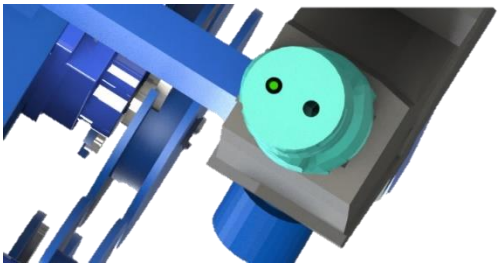


Ilustración 2-43:Guiado de cables a través de la mano

12. Finalmente amarrar los cables tensores a las piezas de los motores mediante un tornillo o un nudo.

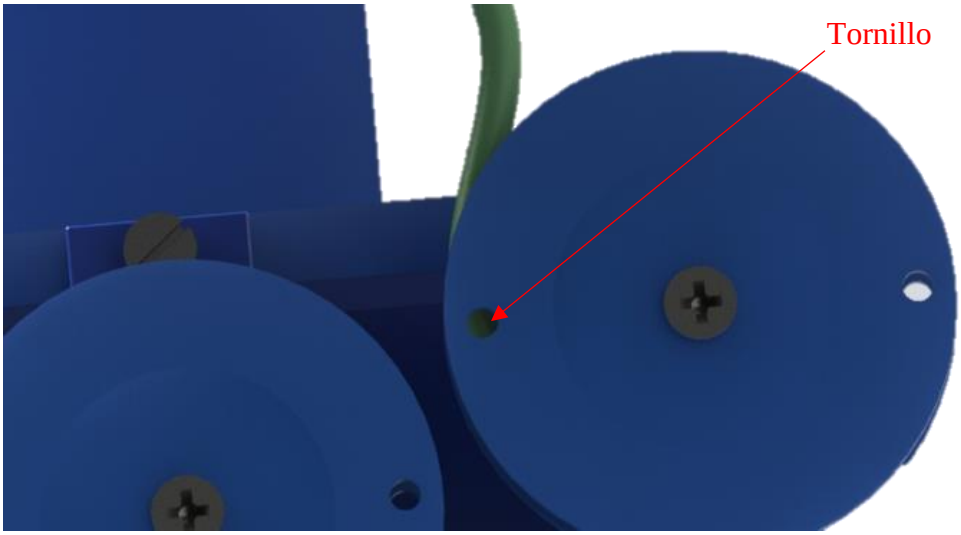


Ilustración 2-44: Amarre de cables a la sujeción del motor

3 SOFTWARE DE CONTROL DE LA MANO ROBÓTICA

3.1 INTRODUCCIÓN

El objetivo de esta sección de introducción es proporcionar una guía que sirva de referencia al lector y describir brevemente la problemática del software.

El diseño del software es una parte esencial en prácticamente cualquier trabajo de ingeniería. El software es la principal vía de comunicación del usuario con los equipos, una interfaz que facilita la utilización del equipo.

Se distinguen dos secciones en este capítulo:

- **Arduino:** Es el microcontrolador encargado del control de los motores. En esta sección se describirá brevemente el hardware utilizado, las conexiones del mismo y las técnicas de control empleadas.
- **Matlab:** Se empleará en la creación de la interfaz gráfica, utilizando la herramienta GUIde. En esta sección se mostrarán algunas de las funciones más utilizadas en la creación de la interfaz.

En el primer prototipo de la mano robótica creada por el grupo Delfín, el software permitía pocas posiciones. Ese primer prototipo estaba orientado al control mediante el controlador IRC5 de ABB. Esto limitaba mucho las opciones ya que no permite más que la comunicación mediante entradas y salidas digitales.

En este proyecto se ha optado por la comunicación mediante puerto serie. Este tipo de comunicación permite mandar al microcontrolador ordenes más complejas. Con estas órdenes más complejas se pueden mandar posiciones precisas de cada uno de los dedos y ajustar la velocidad con la que se moverán.

También en el primer prototipo, era necesario conocer las salidas digitales del controlador IRC5 que actuaban sobre la mano robótica, pues esta carecía de interfaz gráfica. Con la comunicación por puerto serie es posible y necesario desarrollar una interfaz gráfica que simplifique el control de la mano robótica.

En este proyecto se desarrolla una interfaz gráfica, desarrollada con GUIde que mediante el uso de “Sliders”, botones y menús, permitirá el control de la mano robótica en su totalidad.

Ilustración 3-1: Elementos utilizados en la interfaz gráfica

Para situar al lector se adjunta a continuación un esquema general de la comunicación establecida con la interfaz gráfica.

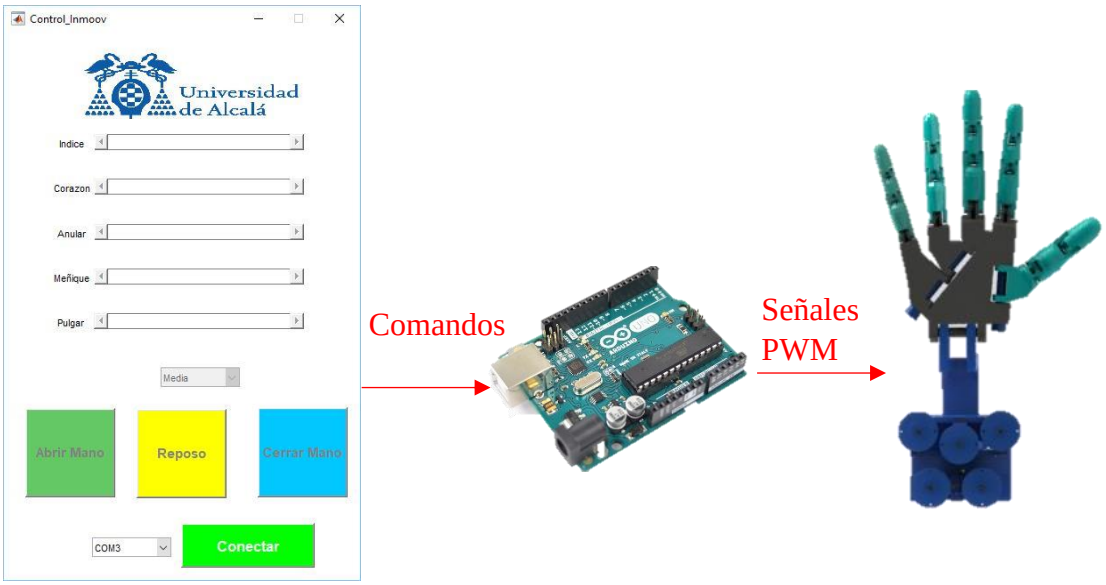


Ilustración 3-2: Esquema general de control

3.2 ARDUINO

Arduino UNO R3 es un microcontrolador de bajo nivel, por lo tanto, para explicar el código empleado primero se debe hacer una descripción del hardware empleado y sus conexiones.

3.2.1 Servomotores Micro Servo 9G

Los servomotores Micro Servo 9G son un tipo de motores controlados en posición ampliamente utilizados en los proyectos con Arduino. Estos motores se caracterizan por ser de un tamaño bastante reducido y poseer un control sencillo.

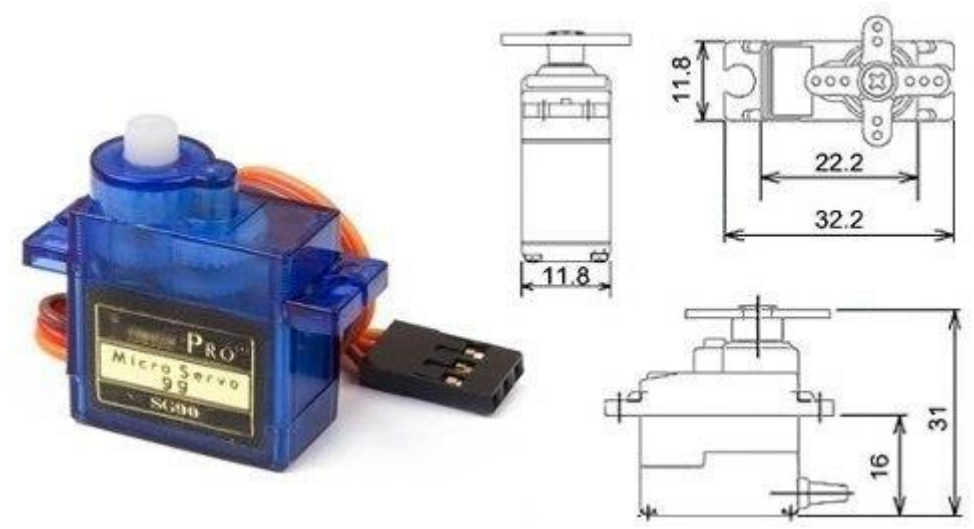


Ilustración 3-3: Dimensiones del motor Micro Servo 9g

El control de estos servomotores es un control en posición mediante señales PWM. La frecuencia y tensión de esta señal la define el fabricante a 4,8VDC/50Hz teniendo que mantener un pulso de 1 ms para situar la posición del motor en -90º y un pulso de 2ms para situarlo en 90º. Con esto se logra un control bastante preciso en cuanto a posición, sin embargo, no se puede actuar sobre el control de la velocidad.

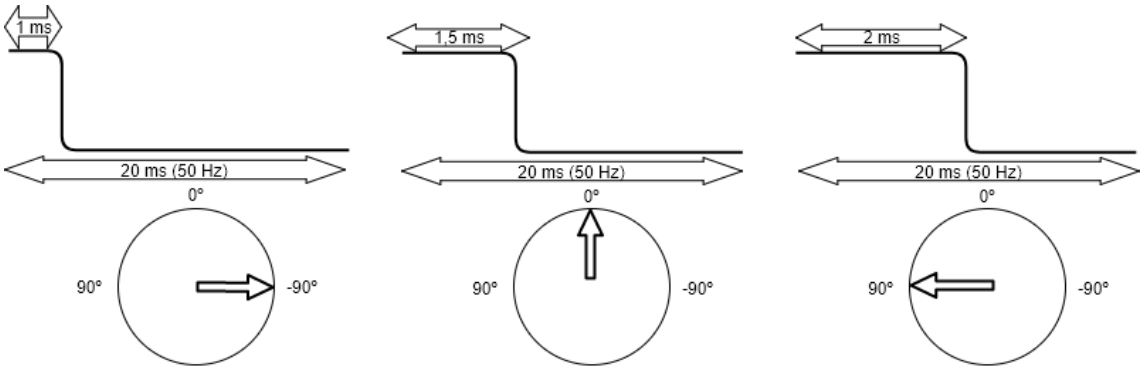


Ilustración 3-4: Señales PWM para Servomotores

3.2.2 Control de posición

La dificultad del control en posición mediante el uso de PWM se reduce gracias a la librería "Servo.h" [8] de Arduino. Esta librería proporciona un nivel de abstracción mayor ya que, para motores estándar, no es necesario conocer los parámetros de la señal PWM. Se debe establecer una configuración mediante código indicando en que pin se encuentra la señal del servomotor, el mínimo valor que puede tomar y el máximo que puede llegar a alcanzar. Los siguientes pasos indican como se realiza el control de posición:

1. Declarar los objetos referentes a los servomotores.

```
1. Servo thumbservo, indexservo, middleservo, ringservo,
   pinkyservo;
```

2. Indicar a que pines se encuentran conectados los servomotores.

```
1.   thumbservo.attach(3);  // PWM del dedo pulgar en pin 5
2.   indexservo.attach(5); //PWM del dedo índice en pin 6
3.   middleservo.attach(6);//PWM del dedo medio en pin 9
4.   ringservo.attach(9);//PWM del dedo anular en pin 10
5.   pinkyservo.attach(10); //PWM del dedo meñique en pin 11
```

3. Establecer la posición deseada entre el mínimo y el máximo.

```
1.   indexservo.write(90);
2.   middleservo.write(90);
3.   ringservo.write(90);
4.   pinkyservo.write(90);
5.   thumbservo.write(90);
```

Con estas sentencias proporcionadas por la librería "Servo.h", ya se puede establecer un control de posición que permite resoluciones de hasta 1º.

La librería "Servo.h" en vez de realizar la rotación del motor como indica el fabricante, de -90º a 90º, la realiza de 0º a 180º.

Inicialmente los motores son situados en la posición de 90º. Esto es debido a que si se iniciasen en 0º, tras varias iteraciones, los cables tensores no serían capaces de volver a situar los dedos en su posición inicial debido a fenómenos de distensión. Situando los motores inicialmente en 90º, se permiten 90º extras para intentar contrarrestar los fenómenos de distensión.

3.2.3 Comunicación Serie

Arduino UNO R3 posee un puerto UART (Universal Asynchronous Receiver Transmitter) que permite la comunicación mediante USB. Para realizar esta comunicación se deben configurar los mismos parámetros tanto en el ordenador como en el microcontrolador. Estos parámetros son:

- **Baudrate:** 57.600.
- **Paridad:** No.
- **NºBits:** 8.

La estructura con la que se mandan los datos es [L][NN]. Donde [L] es una letra que hace referencia al comando que se quiere enviar y [NN] es el valor que se establece a dicho comando.

Los comandos que se pueden mandar mediante comunicación serie son:

Letra	Descripción	Valor mínimo	Valor máximo
T	Posición del dedo pulgar	0	180
I	Posición del dedo índice	0	180
M	Posición del dedo corazón	0	180
R	Posición del dedo anular	0	180
P	Posición del dedo meñique	0	180
V	Velocidad	0	200

Tabla 3-1: Comandos disponibles

Un ejemplo sería el siguiente:

I30

La “I” hace referencia a la posición que se desea en el dedo índice. El “30” indica la posición a la que se debe mover el servomotor indicado en el comando.

El comando de velocidad “V” sigue un comportamiento inversamente proporcional ya que el valor introducido no es la velocidad en si, si no el tiempo de espera entre sucesivos movimientos.

3.2.4 Conexión eléctrica

La conexión de los elementos responde al siguiente diagrama:

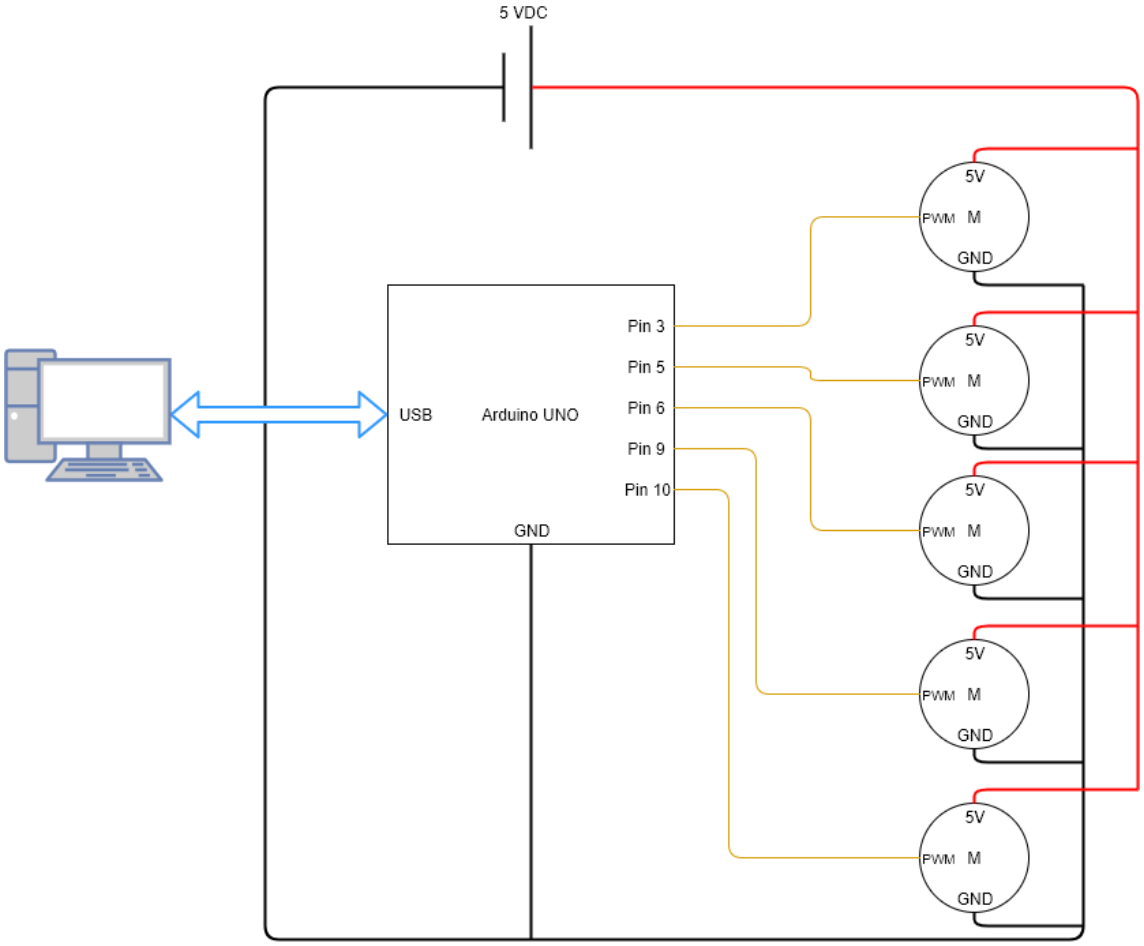


Ilustración 3-5: Diagrama de conexión

Debido al elevado número de motores, es necesario recurrir a una fuente de alimentación externa. Un puerto USB de un PC normal puede llegar a proporcionar unos 500 mA, suficiente para alimentar el Arduino Uno mediante el USB, pero no para alimentar a los motores.

La fuente de alimentación que suministra energía a los motores debe ser capaz de proporcionar mínimo 2 A. Si la fuente no es capaz de proporcionar 2 A no se asegura que los motores ejerzan suficiente par como para realizar los movimientos solicitados.

3.2.5 Control de velocidad

Desde el punto de vista de la seguridad, tanto para el equipo como para el usuario, no es interesante que la velocidad del motor sea siempre la máxima, por ello es necesario un método para actuar sobre la velocidad del mismo. En este diseño, al no tener acceso al control de velocidad, se ha decidido limitar cada cuanto tiempo tiene que realizar el movimiento.

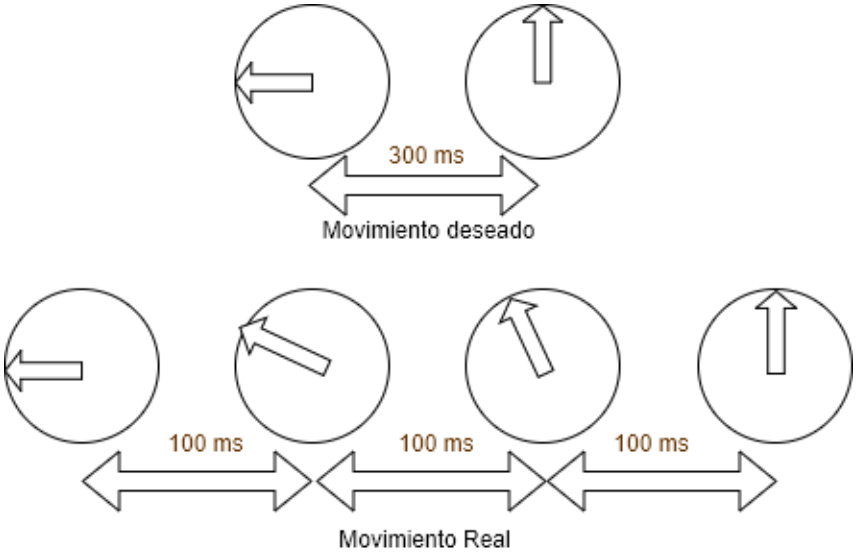


Ilustración 3-6: Control de velocidad de los motores

Este control divide el movimiento en una consecución de movimientos más pequeños espaciados en el tiempo. Estos pequeños movimientos estarán más o menos espaciados en el tiempo en función de la velocidad que se haya seleccionado.

Para realizarlo se utiliza esta función cuyos parámetros de entrada son “Pos”, que es la posición que se desea alcanzar y “Pos_Cal”, que es la posición actual del motor. Esta función es evaluada cada cierto tiempo en función de la velocidad seleccionada.

```
int Calcula_posicion(int Pos, int Pos_Cal){

    if (Pos!=Pos_Cal){

      if(Pos<Pos_Cal) Pos_Cal--;

      if(Pos>Pos_Cal) Pos_Cal++;

      }

   return Pos_Cal;}delay(Velocidad);
```

3.2.6 Diagrama de flujo del software de control de Arduino

El diagrama de flujo del código del microcontrolador es:

1. Inicializamos variables.
2. Abrimos comunicación mediante UART.
3. Comprobamos recepción de mensaje.
4. Decodificamos mensaje.
5. Actuación sobre los motores si fuera necesario.

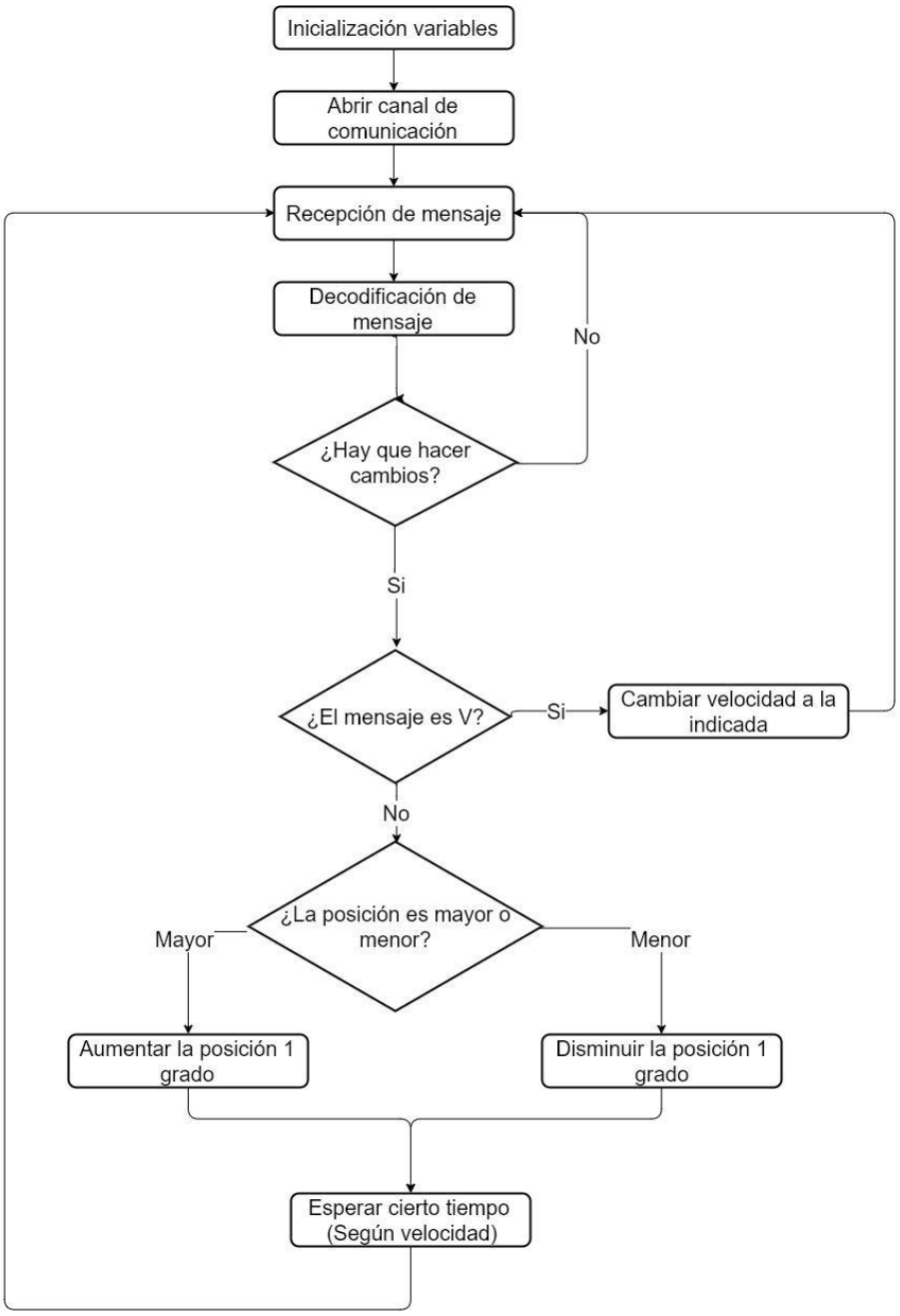


Ilustración 3-7: Diagrama de flujo del código de control

3.3 MATLAB

Matlab dispone de muchas herramientas útiles para los ingenieros e investigadores, una de ellas es GUIDE. Graphic User Interface Design Editor (GUIDE) permite, a cualquier persona con nociones básicas de programación orientada a objetos, crear una interfaz de usuario de forma rápida y sencilla.

Las formas de control más sencillas son los Sliders y las posiciones fijas.

Los Sliders permiten un control bastante preciso a la vez que son atractivos visualmente, además, aportan información del estado actual del sistema.

Los botones de posición fija definen unas posiciones típicas de forma que si se desea esa posición no es necesario manejar los Sliders uno a uno.

Si se observa el código de los Sliders se puede comprender su funcionamiento rápidamente. La única función del Slider es mandar a través del puerto serie una trama de datos que interpretará el microcontrolador.

```
1. function Slider_indice_Callback(hObject, eventdata, handles)
2. Posicion=get(hObject,'Value');
3. fwrite(handles.Serial,strcat('I',int2str(Posicion)))
4. set(handles.Text_mensajes,'String',strcat('Se ha cambiado la
   posicion INDICE a -->',int2str(Posicion)));
5. pause(0.8);
```

Esta función es llamada cuando se realiza algún cambio en el Slider correspondiente al dedo Índice. Se obtiene la posición del Slider y se manda por el puerto serie con la letra "I" para hacer referencia a que el cambio debe ser realizado en el dedo Índice. Finalmente se muestra un mensaje por pantalla para indicar que se ha enviado la posición correctamente y se espera un tiempo para evitar mandar varios mensajes a la vez y bloquear el sistema.

Si se observa el código de los botones de posición fija se puede ver que el código no es muy diferente al de los Sliders.

```
1. function Boton_cerrar_Callback(hObject, eventdata, handles)
2.    set(handles.Slider_pulgar,'Value',170);
3.    set(handles.Slider_indice,'Value',170);
4.    set(handles.Slider_corazon,'Value',170);
5.    set(handles.Slider_anular,'Value',170);
6.    set(handles.Slider_menique,'Value',170);
7.    Posicion=170;
8.    fwrite(handles.Serial,strcat('M',int2str(Posicion)))
9.    fwrite(handles.Serial,strcat('T',int2str(Posicion)))
10.    fwrite(handles.Serial,strcat('I',int2str(Posicion)))
11.    fwrite(handles.Serial,strcat('R',int2str(Posicion)))
12.    fwrite(handles.Serial,strcat('P',int2str(Posicion)))
13.    set(handles.Text_mensajes,'String','Se ha CERRADO la
   mano');
```

Este botón corresponde a uno de posiciones fijas, en concreto la posición de la mano cerrada. Este código, además de enviar los datos al Arduino UNO como en el caso de los Sliders, modifica los valores de los Sliders para que el usuario sea consciente de en que situación se encuentra cada dedo.

La función para establecer la comunicación es sencilla gracias a las funciones aportadas por Matlab.

- Puerto = serial(Nombre_Puerto, Baudrate)
- fopen(Puerto)

Se crea una conexión con el puerto que se desee, estableciendo los parámetros de la comunicación

- Baudrate

- Paridad

- Nº Bits

Esta parte del software funciona mediante interrupciones. Matlab se encarga de la gestión de estas interrupciones de forma totalmente transparente para el programador de la interfaz gráfica. En ciertas aplicaciones esto supone un problema pues Matlab no da la opción de definir la prioridad de estas interrupciones o eventos. En este caso, al realizar la comunicación de una forma rápida y no ser necesario respuestas instantáneas para ninguno de los procesos, no supone ningún problema.

La interfaz gráfica desarrollada es la siguiente:

Ilustración 3-8: Interfaz diseñada

El objetivo de la interfaz es hacer la comunicación sencilla e intuitiva. Aun así, para establecer la conexión se debe seguir un pequeño diagrama de flujo:

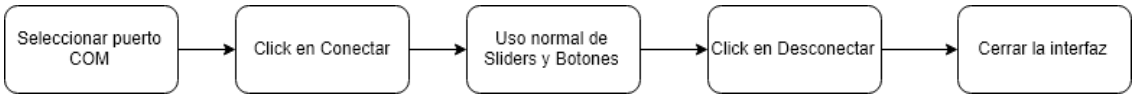


Ilustración 3-9: Flujograma de uso de la interfaz

Primero se debe seleccionar el puerto COM con menú que hay a la izquierda del botón "Conectar". Posteriormente se debe hacer click en conectar. Si la conexión se ha establecido correctamente, se habilitarán los demás botones y los sliders, y el botón de conectar cambiará de color y pondrá "Desconectar". Con esto la conexión ya está establecida y se pueden mandar comandos a la mano robótica. Finalmente, si se desea salir, aunque el programa ha sido diseñado para evitar problemas por cierres accidentales, se recomienda hacer click en "Desconectar" para evitar problemas durante futuras conexiones.

4 RESULTADOS

En este capítulo se muestran los resultados obtenidos y se realiza una crítica de los mismos.

4.1 CUMPLIMIENTO DE OBJETIVOS

El objetivo de este proyecto era doble, por un lado, había que diseñar y fabricar nuevas piezas para la utilización de nuevos motores y por otro lado había que desarrollar un software que proporcionase un control más versátil con una interfaz simple.

4.1.1 Objetivo del sistema mecánico

Se ha conseguido diseñar un sistema que permita la utilización de unos motores distintos y se ha dejado preparada para una futura mejora. La futura mejora diseñada para tensar los cables le proporcionará al sistema una mayor facilidad de mantenimiento.

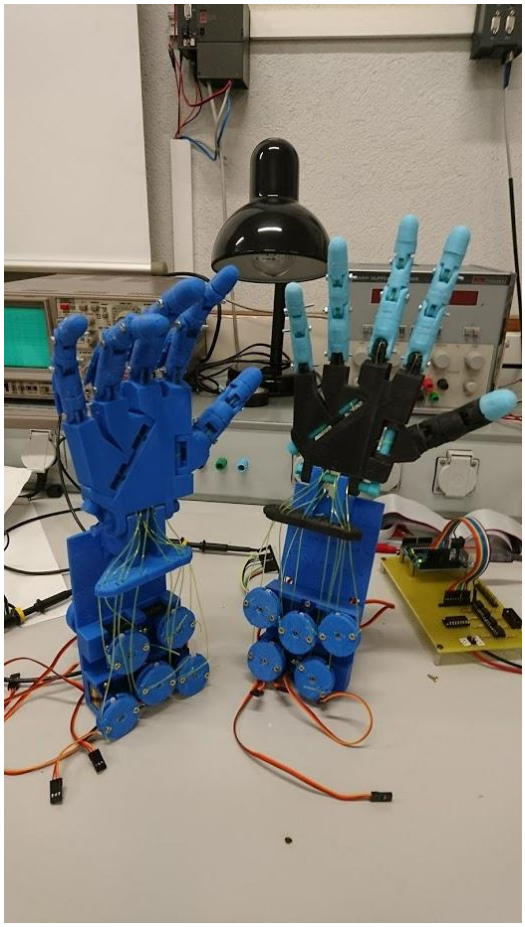


Ilustración 4-1: Manos robóticas vista frontal

Inicialmente se construyó una mano reutilizando las piezas del prototipo anterior. Posteriormente, con la ayuda del grupo del programa Delfín de 2017 se construyó una segunda mano desde cero, utilizando el mismo diseño en ambas.

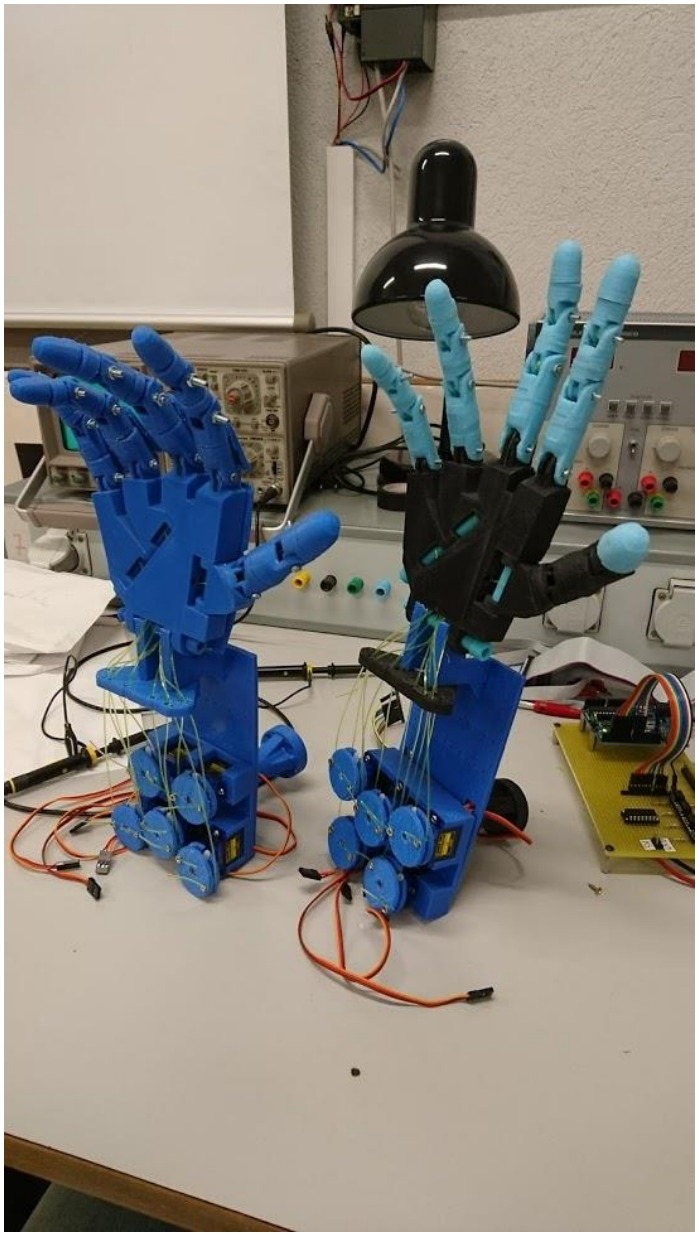


Ilustración 4-2: Manos robóticas vista lateral

4.1.2 Objetivo del software de control de la mano robótica

El control diseñado con Arduino en este modelo es absoluto, permitiendo controlar tanto posición como velocidad de una forma sencilla mediante comandos.

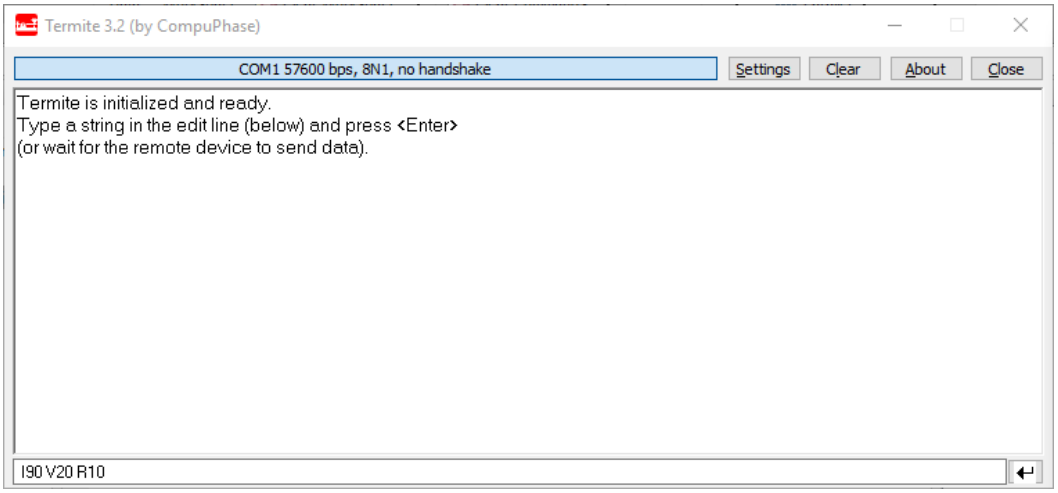


Ilustración 4-3: Control mediante comandos

La interfaz diseñada establece un método de comunicación más visual que el prototipo anterior.

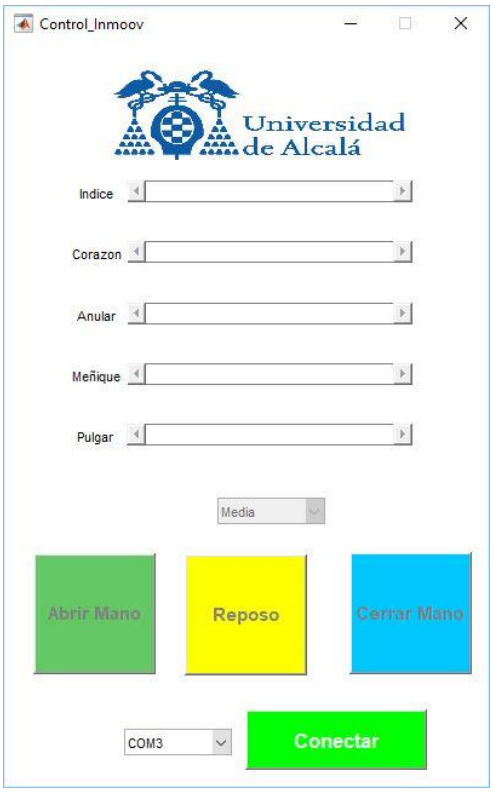


Ilustración 4-4: Interfaz de control de la mano robótica

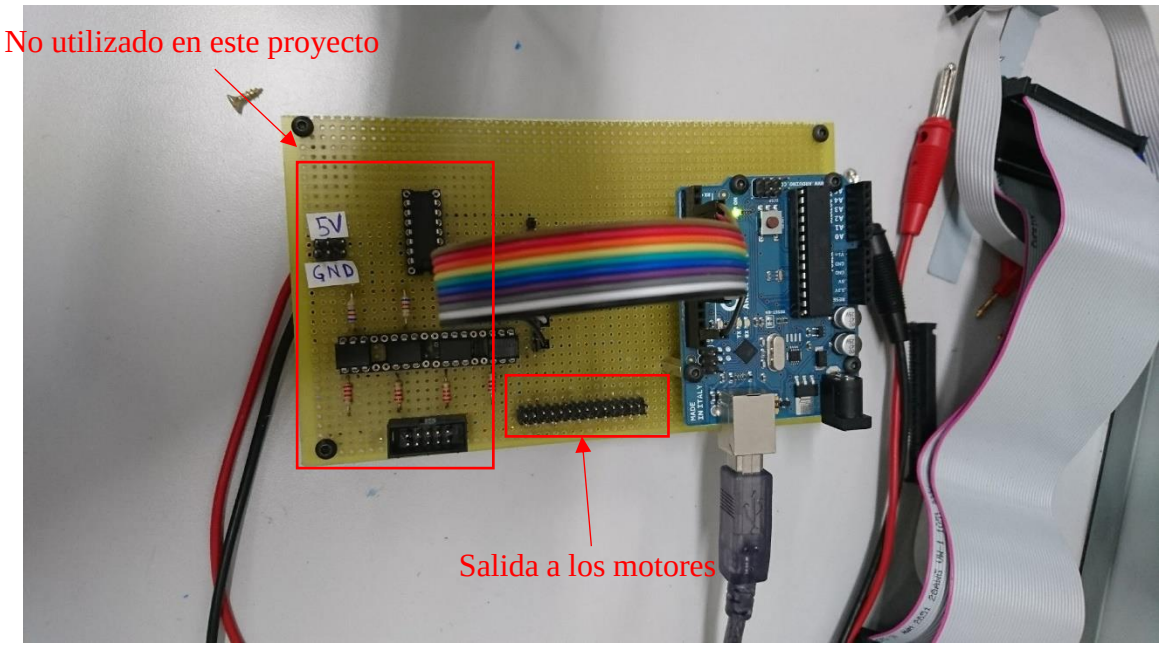


Ilustración 4-5: Circuito de control

4.1.3 Posiciones de la mano robótica

A continuación, se adjuntan unas imágenes que muestran las distintas posiciones preestablecidas que puede alcanzar la mano y otras no preestablecidas con el objetivo de mostrar la versatilidad del control:

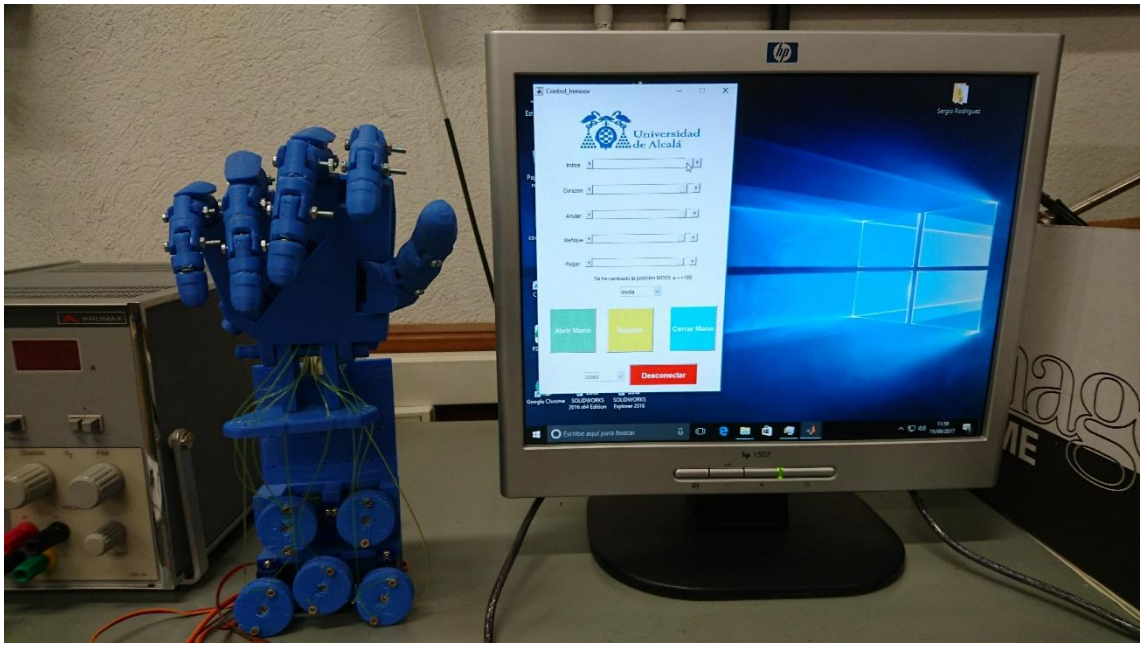


Ilustración 4-6: Mano robótica real cerrada

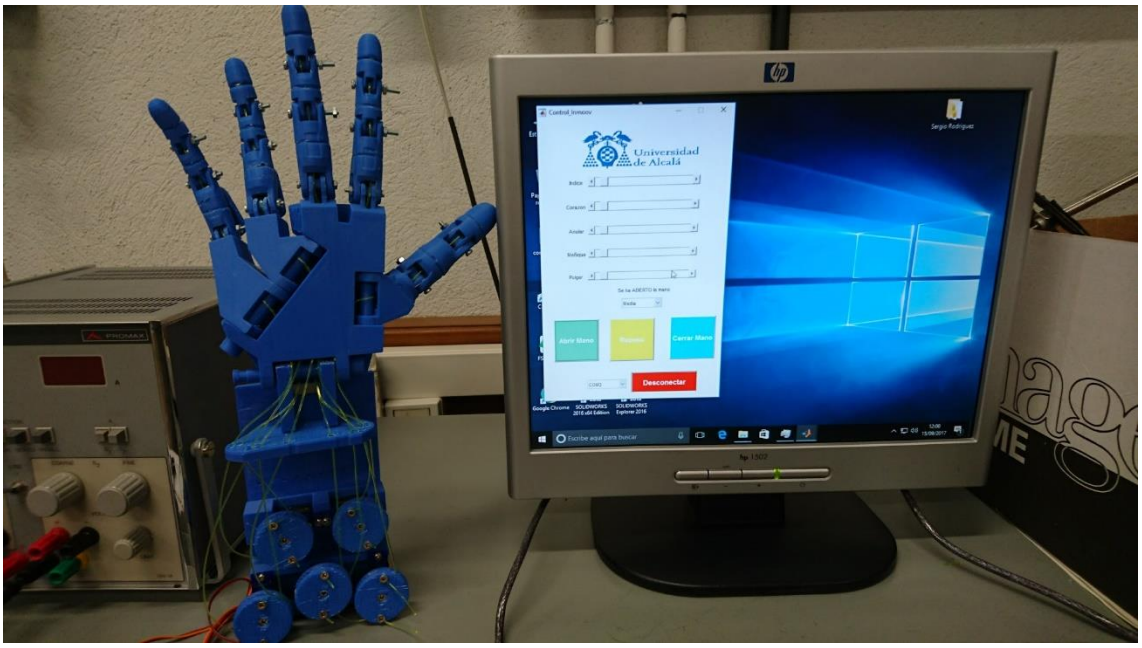


Ilustración 4-8: Mano robótica real abierta

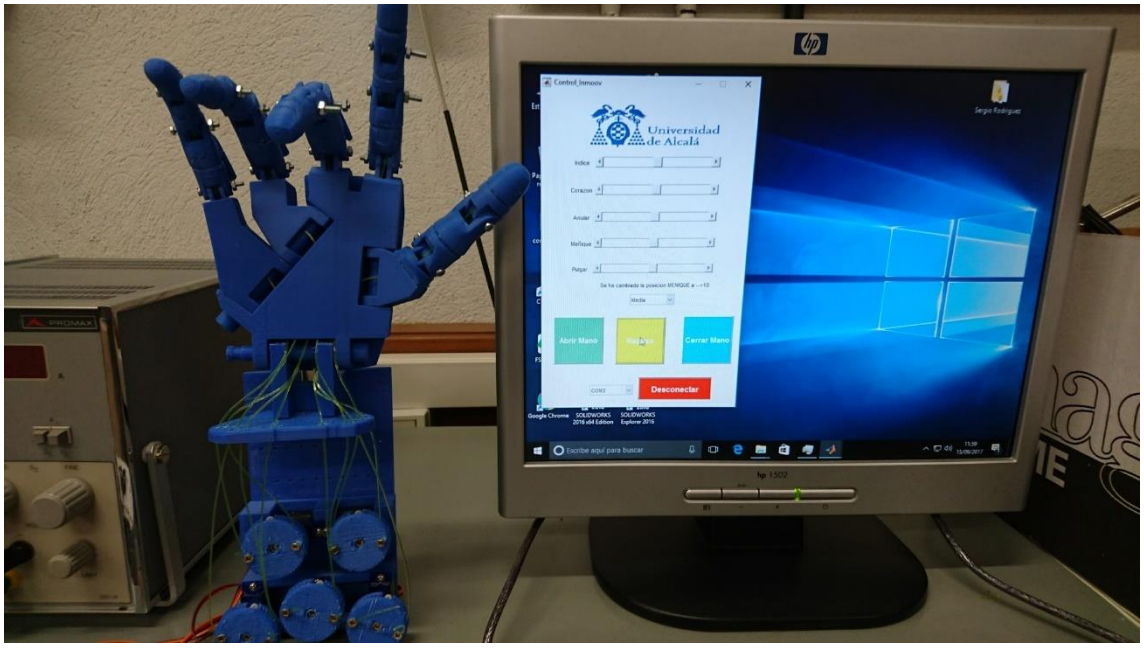


Ilustración 4-7:Mano robótica real en reposo

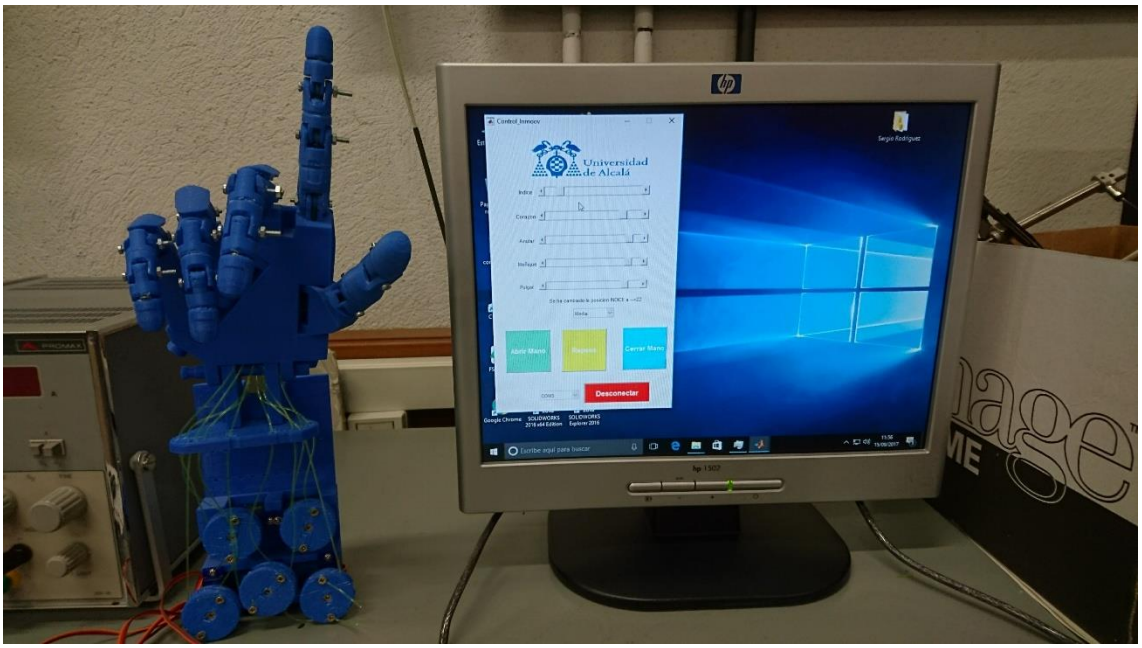


Ilustración 4-9:Mano robótica real con índice levantado

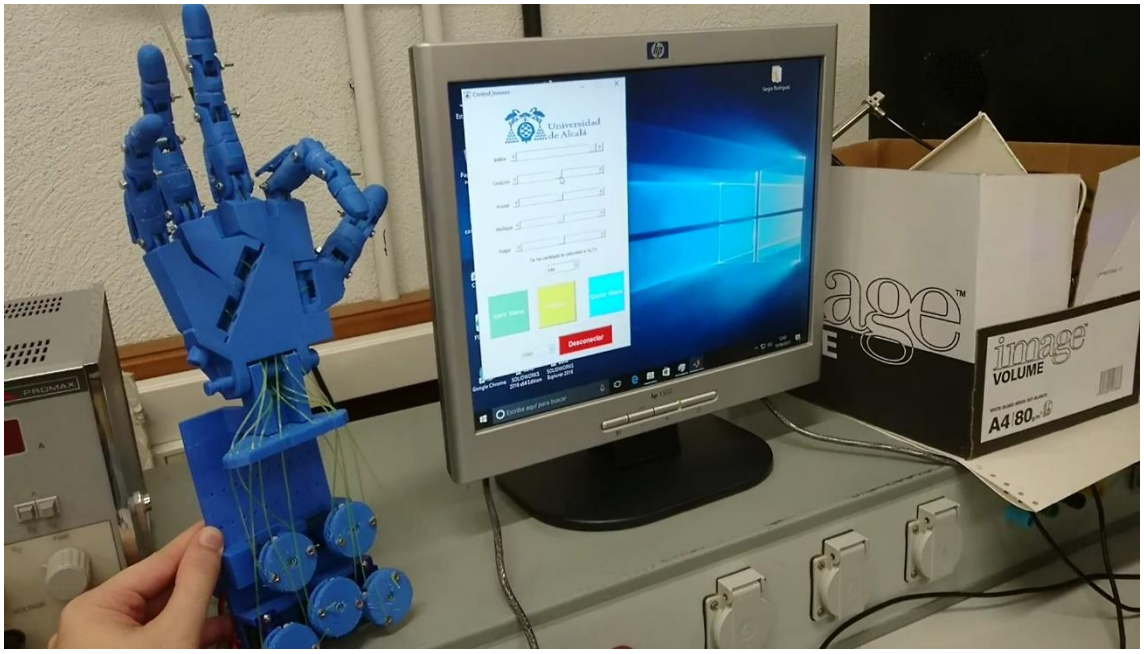


Ilustración 4-10: Mano robótica real con dedo índice recogido

4.2 ERRORES

En la creación de todo diseño se cometen errores. Algunos de estos errores son críticos y deben subsanarse inmediatamente, otros no son tan críticos y pueden ser solucionados en la siguiente revisión de diseño.

A continuación, se detallan algunos de los errores cometidos y su solución en caso de haberse aplicado.

4.2.1 Errores en el diseño mecánico

Los errores en el diseño mecánico han sido los más críticos ya que los fallos de diseño normalmente se convierten en roturas, averías o destrucción del sistema en general. Las soluciones a estos errores normalmente conllevan realizar el proceso de diseño de nuevo, sin embargo, en ocasiones el daño causado puede ser fácilmente solucionado de forma instantánea.

Los errores más comunes han sido los siguientes.

4.2.1.1 Tolerancias

En el primer diseño que se realizó no se tuvieron en cuenta las tolerancias de fabricación. Esto se tradujo en piezas que no encajaban dentro de otras, rozamientos indeseados, choques con otras piezas y, en general, un sistema que no funcionaba de forma fluida.

La solución para este problema fue aplicar un procedimiento de mecanizado a cada pieza, eliminando capas de material y realizando taladros con una broca mayor. En los siguientes diseños se tuvo en cuenta las tolerancias durante la fabricación y se redujo notablemente el mecanizado posterior.

4.2.1.2 Falta de relleno

En el capítulo de impresión 3D se mencionó este tipo de problema, que es bastante común en las impresiones 3D. Debido a la falta de relleno durante la impresión 3D, las zonas que soportan más tensión pueden llegar a romper. Este fue el caso con la pieza “Servohand” en numerosas ocasiones.

La solución instantánea a este problema fue volver a unir la pieza mediante un adhesivo líquido. Si esta solución no era válida se procedía a la impresión de una pieza con mayor nivel de relleno.

En futuros diseños se recomienda establecer el parámetro de relleno en un valor elevado para las piezas que vayan a soportar más tensión.

4.2.2 Errores en el software de control de la mano robótica

Los errores en el software no suelen ser tan críticos. Estos errores suelen ser detectados durante simulaciones o durante pruebas que no ponen en riesgo la integridad del sistema.

Los principales errores son los siguientes:

4.2.2.1 Errores de gestión en la desconexión

Durante la creación de la interfaz gráfica en Matlab, se detectó que si el canal de comunicación no era cerrado correctamente este se quedaba abierto y no permitía la comunicación ni el cierre de la aplicación.

La solución de este error fue el empleo de sentencias de control de flujo para establecer el procedimiento correcto de cierre. Sin embargo, se ha detectado un error no crítico en este control de flujo. Concretamente cuando se intenta establecer una comunicación que no existe, el programa se bloquea. Este error debe ser subsanado en la siguiente revisión del prototipo.

4.2.2.2 Errores en control de velocidad

El método de control de velocidad aplicado en este proyecto no es un método muy robusto. Durante el desarrollo de este control se han producido diversos errores debidos principalmente al número de grados que se mueve en cada iteración. En las primeras pruebas este valor era muy elevado, produciendo un movimiento errático y la rotura de algunas piezas.

La solución a este problema ha sido utilizar el valor mínimo que se puede emplear. Sin embargo, este valor no es muy eficiente desde el punto de vista del procesador.

5 CONCLUSIONES Y TRABAJOS FUTUROS

Las principales conclusiones que se han sacado a lo largo del proyecto han sido las siguientes.

Este proyecto integra distintos campos de la ingeniería tales como electrónica, mecánica y software. Esta capacidad de integrar tecnologías de distintos ámbitos es una de las principales características del ingeniero industrial.

El modelado de una parte del cuerpo humano no es sencillo, sin embargo, si se trabaja con licencias "Open Source" y se recibe apoyo por parte de la comunidad, se pueden lograr diseños muy eficaces como en el caso de la mano Inmoov.

Actualmente existen herramientas muy potentes que permiten la creación de sistemas o elementos complejos de una forma relativamente sencilla.

El mundo de la impresión 3D está muy desarrollado actualmente, sin embargo, aún le falta mucho camino por recorrer para que los usuarios medios puedan disfrutar de acabados profesionales de forma sencilla.

Este proyecto ha servido para sentar una base a futuros trabajos. Durante el desarrollo del mismo se han ido proponiendo ciertas mejoras:

- Utilización de tensores mecánicos para los cables encargados del movimiento.
- Protección para la caja de motores y la electrónica.
- Implementación de galgas para detección de fuerza.
- Creación de otro modelo de interfaces del tipo guante.
- Implementación de almohadillas en las yemas de los dedos.

Finalmente, para obtener un buen producto se podría cambiar el material por otro más rígido y duradero.

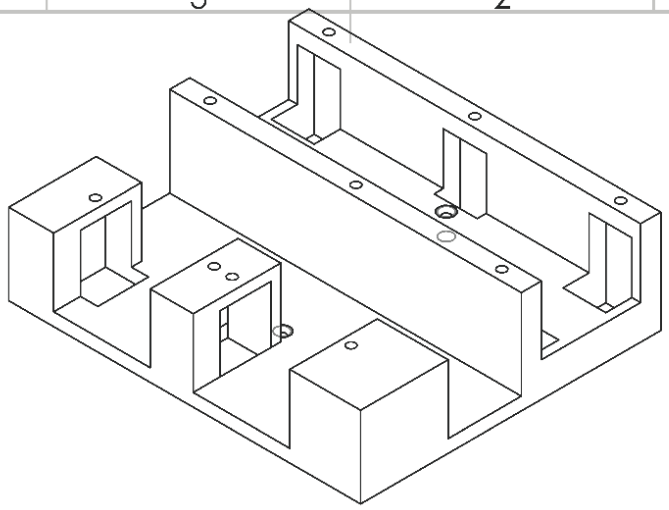
6 PLANOS

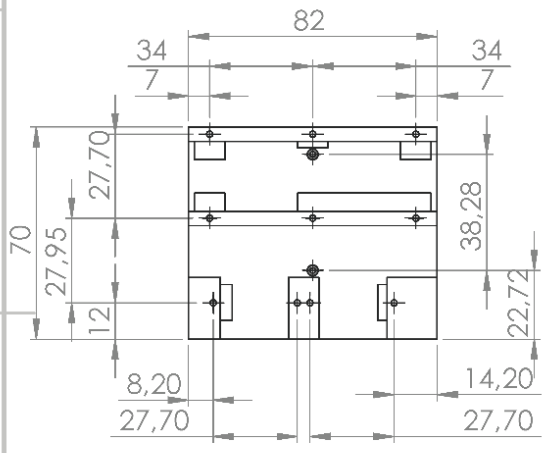
En este capítulo se incluyen todos los planos necesarios para la ejecución del proyecto.

El capítulo se divide en dos secciones principales

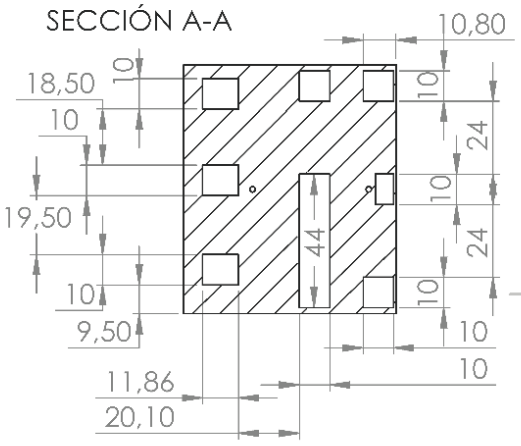
- **Piezas:** En esta primera sección se encontrarán los planos de las piezas diseñadas para este proyecto específico. No se incluyen los planos correspondientes a la mano robótica Inmoov.

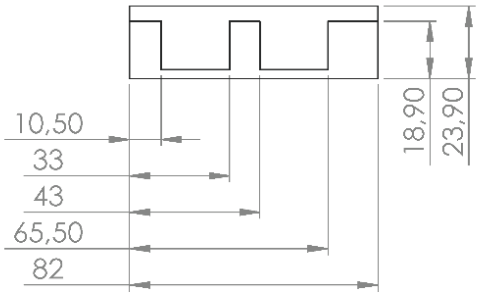
- **Software de control:** En esta sección se incluyen los códigos generados tanto para Arduino como para la interfaz gráfica en Matlab.





SECCIÓN A-A





REBARBAR Y ROMPER ARISTAS VIVAS	NO CAMBIE LA ESCALA	REVISIÓN 3

	NOMBRE	FIRMA	FECHA
DIBU...	Sergio Rodríguez		08/08/17
VERIF.			
APROB.			
FABR.			
CALID.			

TÍTULO:

Cama de servos

MATERIAL:

PLA

N.º DE DIBUJO

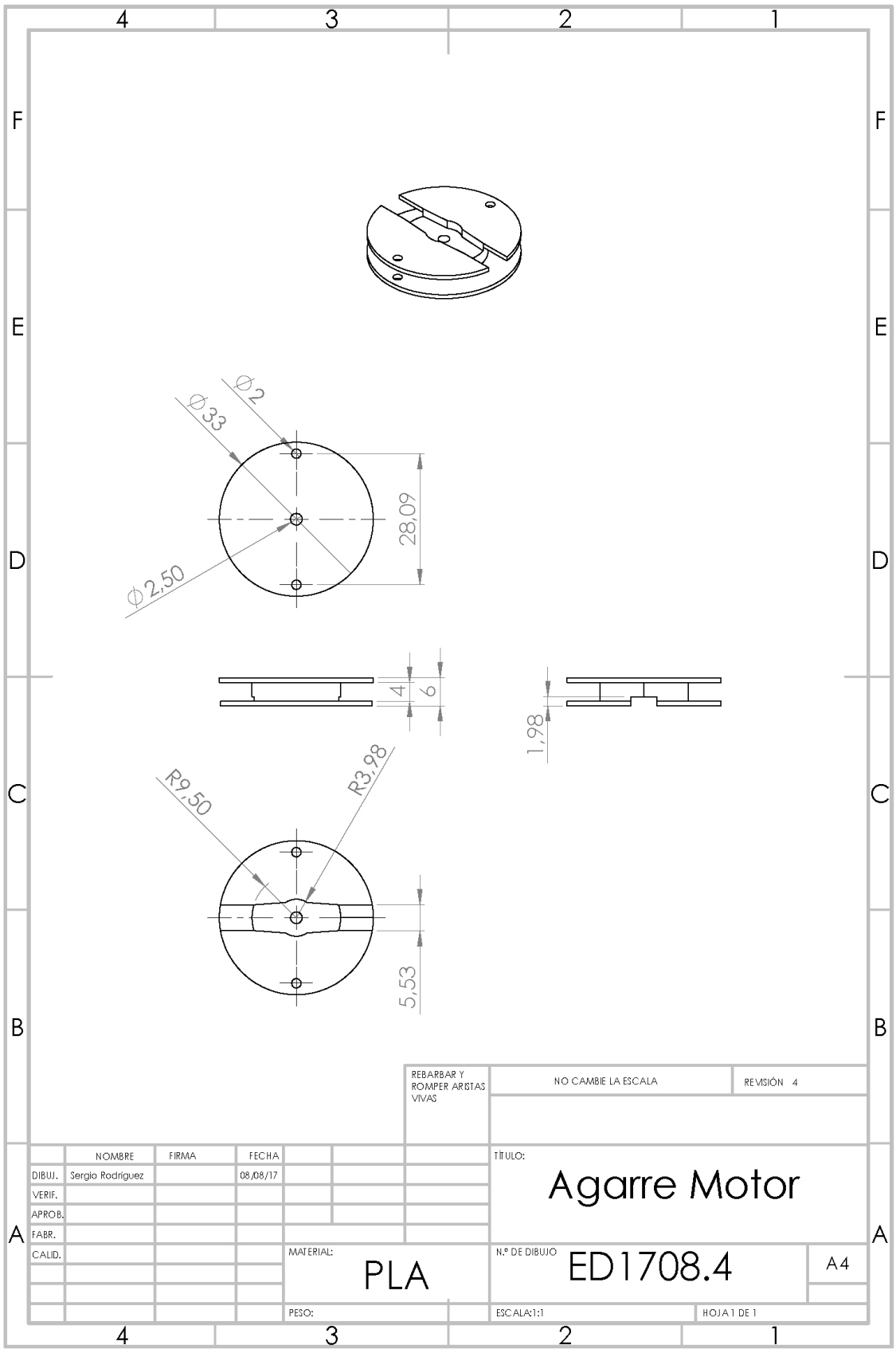
ED1708.3

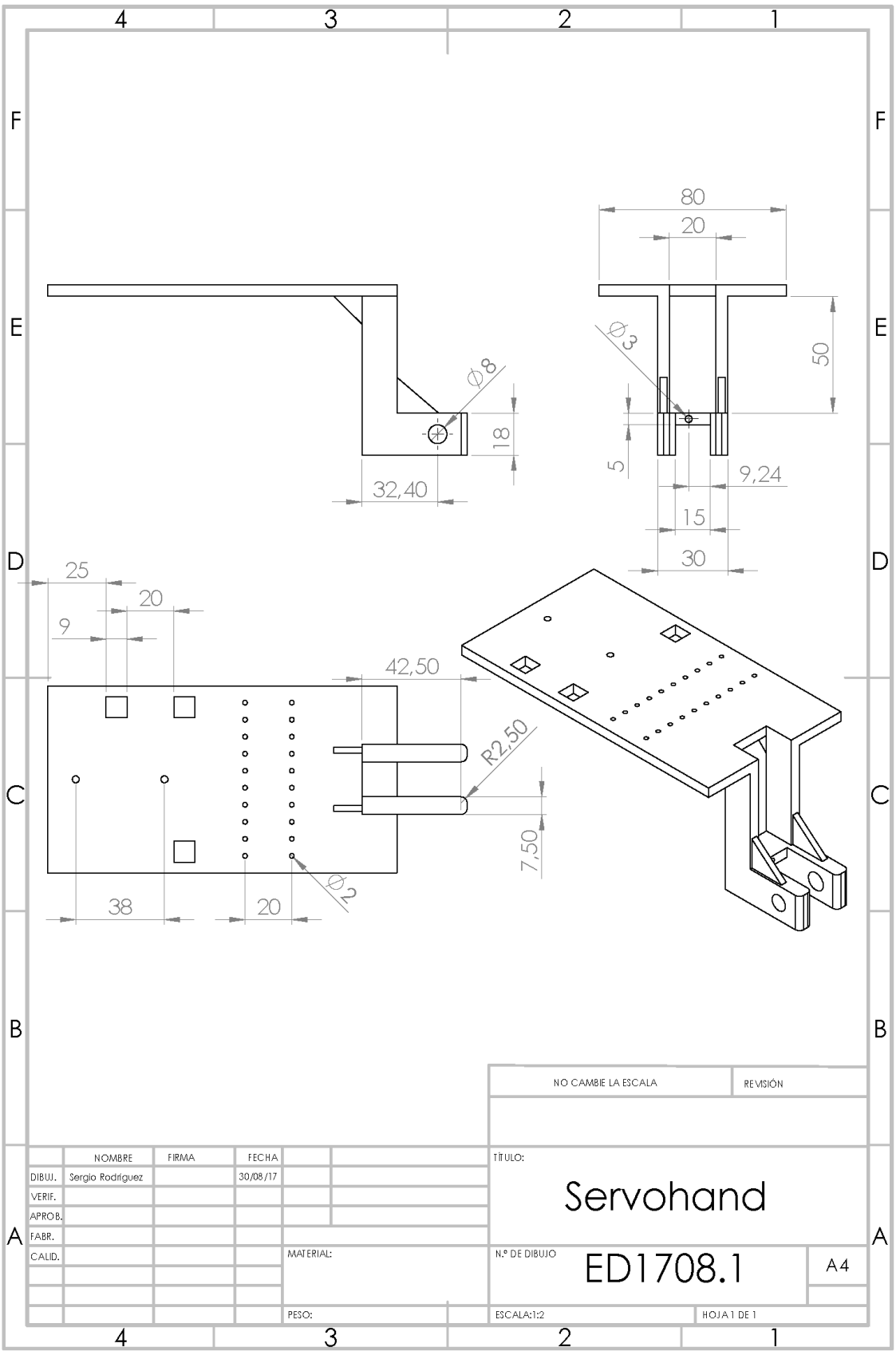
A4

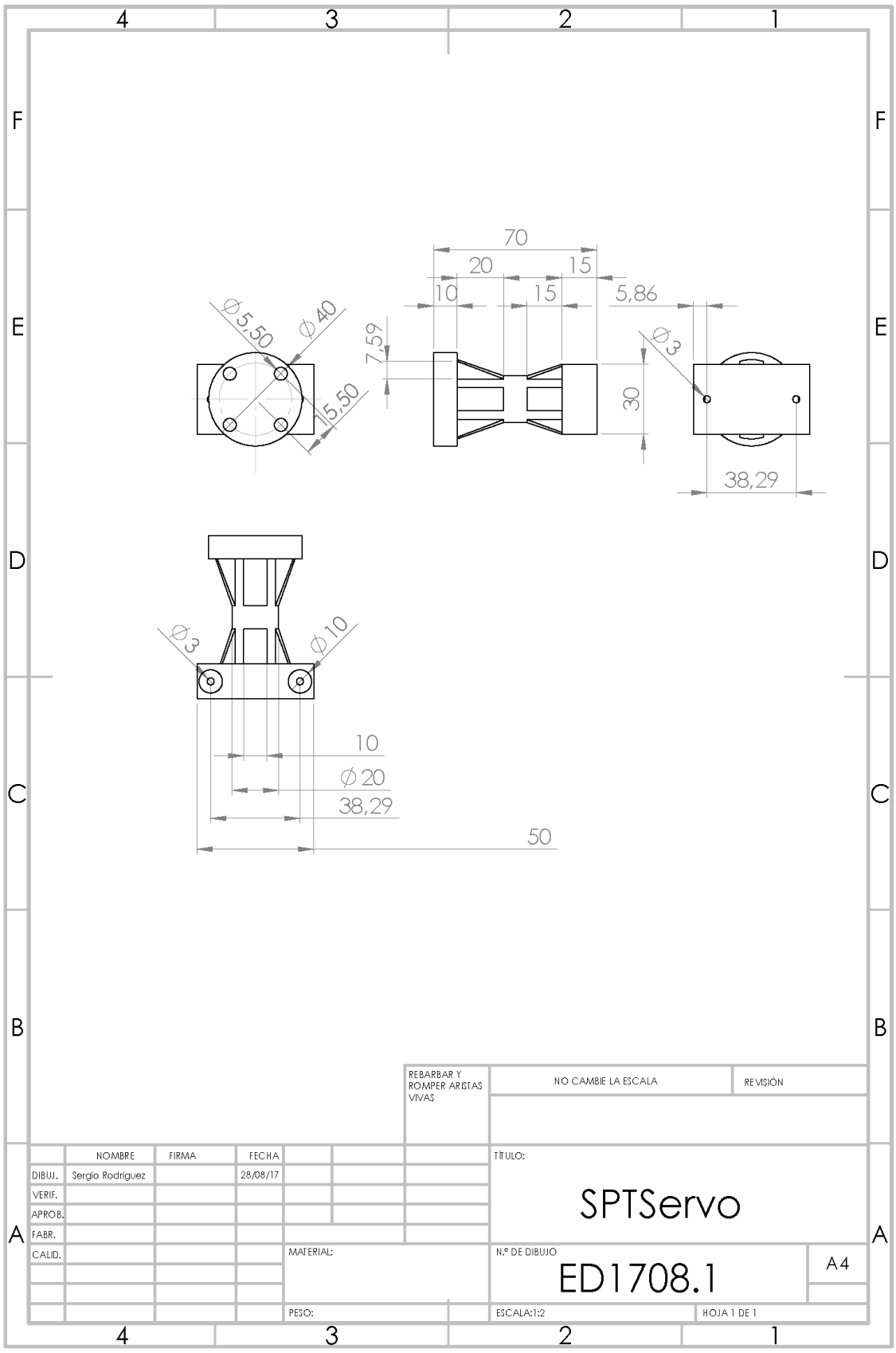
PESO:

ESCALA:1:2

HOJA 1 DE 1


4
3
2
1
F
E
D
C
B
A
Ø2
Ø33
28,09
Ø2,50
4
6
1,98
R9,50
R3,98
5,53
REBARBAR Y ROMPER ARISTAS VIVAS
NO CAMBIE LA ESCALA
REVISIÓN 4
NOMBRE
FIRMA
FECHA
DIBUJ. Sergio Rodriguez 08/08/17
VERIF.
APROB.
FABR.
CALID.
TÍTULO:
Agarre Motor
MATERIAL:
PLA
N.º DE DIBUJO
ED1708.4
A4
PESO:
ESCALA:1:1
HOJA 1 DE 1


4
3
2
1
F
E
D
C
B
A
80
20
50
Ø3
Ø8
18
32,40
5
9,24
15
30
25
20
9
42,50
R2,50
7,50
Ø2
38
20
NO CAMBIE LA ESCALA
REVISIÓN
NOMBRE
FIRMA
FECHA
DIBU.
Sergio Rodríguez
30/08/17
VERIF.
APROB.
FABR.
CALID.
MATERIAL:
PESO:
TÍTULO:
Servohand
N.º DE DIBUJO
ED1708.1
A4
ESCALA:1:2
HOJA 1 DE 1


4
3
2
1
F
E
D
C
B
A
70
20
15
10
15
5,86
Ø5,50
Ø40
7,59
15,50
Ø3
30
38,29
Ø3
Ø10
10
Ø20
38,29
50
REBARBAR Y ROMPER ARISTAS VIVAS
NO CAMBIE LA ESCALA
REVISIÓN
NOMBRE
FIRMA
FECHA
DIBUJ.
Sergio Rodriguez
28/08/17
VERIF.
APROB.
FABR.
CALID.
TÍTULO:
SPTServo
MATERIAL:
N.º DE DIBUJO
ED1708.1
A4
PESO:
ESCALA:1:2
HOJA 1 DE 1

SOFTWARE DE CONTROL ARDUINO UNO

```
#include <Servo.h>

Servo thumbservo, indexservo, middleservo, ringservo,
pinkyservo;  // crea un objeto de control para el servo


    int posI = 0, posM = 0, posR=0, posP=0, posT=0; //
guarda la posición del servo
    int posI_C = 0, posM_C = 0, posR_C=0, posP_C=0,
posT_C=0;
    int Pos=0;
    int Dedo;
    int Velocidad=100;


    void setup()
    {
      thumbservo.attach(3);  // PWM del dedo pulgar en pin 5
      indexservo.attach(5); //PWM del dedo índice en pin 6
      middleservo.attach(6);//PWM del dedo medio en pin 9
      ringservo.attach(9);//PWM del dedo anular en pin 10
      pinkyservo.attach(10); //PWM del dedo meñique en pin
11
      Serial.begin(57600);


      indexservo.write(90); // manda a la posición inicial
todos los dedos
      middleservo.write(90);
      ringservo.write(90);
      pinkyservo.write(90);
      thumbservo.write(90);
    }

    void loop(){
      while (Serial.available()>0){
        Dedo=Serial.read();
        switch (Dedo){
          case 'I': posI=Serial.parseInt();break;
          case 'M': posM=Serial.parseInt();break;
          case 'R': posR=Serial.parseInt();break;
          case 'P': posP=Serial.parseInt();break;
          case 'T': posT=Serial.parseInt();break;
          case 'V': Velocidad=Serial.parseInt();break;
      }
      Serial.println(thumbservo.read());
      }
      posI_C=Calcula_posicion(posI,posI_C);
      posM_C=Calcula_posicion(posM,posM_C);
      posR_C=Calcula_posicion(posR,posR_C);
```

```
51.       posP_C=Calcula_posicion(posP,posP_C);
52.       posT_C=Calcula_posicion(posT,posT_C);
53.       indexservo.write(posI_C); // manda a la posición
   inicial todos los dedos
54.       middleservo.write(posM_C);
55.       ringservo.write(posR_C);
56.       pinkyservo.write(posP_C);
57.       thumbservo.write(posT_C);
58.       delay(Velocidad);
59.       }
60.
61.       int Calcula_posicion(int Pos, int Pos_Cal){
62.
63.           if (Pos!=Pos_Cal){
64.             if(Pos<Pos_Cal) Pos_Cal--;
65.             if(Pos>Pos_Cal) Pos_Cal++;
66.             }
67.           return Pos_Cal;}
```

```
function varargout = Control_Inmoov(varargin)
% CONTROL_INMOOV MATLAB code for Control_Inmoov.fig
%      CONTROL_INMOOV, by itself, creates a new CONTROL_INMOOV or raises the existing
%      singleton*.
%
%      H = CONTROL_INMOOV returns the handle to a new CONTROL_INMOOV or the handle to
%      the existing singleton*.
%
%      CONTROL_INMOOV('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CONTROL_INMOOV.M with the given input arguments.
%
%      CONTROL_INMOOV('Property','Value',...) creates a new CONTROL_INMOOV or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Control_Inmoov_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Control_Inmoov_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Control_Inmoov

% Last Modified by GUIDE v2.5 26-Jul-2017 12:58:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Control_Inmoov_OpeningFcn, ...
                   'gui_OutputFcn',  @Control_Inmoov_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
%--------------------------------------------------------------------------%
%-----------------------------------Initilization--------------------------%
%--------------------------------------------------------------------------%

% --- Executes just before Control_Inmoov is made visible.
function Control_Inmoov_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Control_Inmoov (see VARARGIN)

% Choose default command line output for Control_Inmoov
handles.output = hObject;
handles.Ha_conectado=0;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Control_Inmoov wait for user response (see UIRESUME)
% uiwait(handles.Control);


% --- Outputs from this function are returned to the command line.
function varargout = Control_Inmoov_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


function Slider_indice_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Slider_indice (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
% --- Executes during object creation, after setting all properties.
% --- Executes during object creation, after setting all properties.

function Slider_corazon_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Slider_corazon (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes during object creation, after setting all properties.
function Slider_anular_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Slider_anular (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
```

```
end

% --- Executes during object creation, after setting all properties.
function Slider_menique_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Slider_menique (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes during object creation, after setting all properties.
function Slider_pulgar_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Slider_pulgar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes during object creation, after setting all properties.
function Menu_puertos_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Menu_puertos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function Menu_velocidad_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Menu_velocidad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%--------------------------------------------------------------------------%
%-----------------------------------Callback-------------------------------%
%--------------------------------------------------------------------------%

% --- Executes on slider movement.
```

```
function Slider_indice_Callback(hObject, eventdata, handles)
% hObject    handle to Slider_indice (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Posicion=get(hObject,'Value');
fwrite(handles.Serial,strcat('I',int2str(Posicion)))
set(handles.Text_mensajes,'String',strcat('Se ha cambiado la posicion INDICE a --
>',int2str(Posicion)));
pause(0.8);
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% --- Executes on slider movement.
function Slider_corazon_Callback(hObject, eventdata, handles)
% hObject    handle to Slider_corazon (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Posicion=get(hObject,'Value');
fwrite(handles.Serial,strcat('M',int2str(Posicion)))
set(handles.Text_mensajes,'String',strcat('Se ha cambiado la posicion CORAZON a --
>',int2str(Posicion)));
pause(0.8);
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% --- Executes on slider movement.
function Slider_anular_Callback(hObject, eventdata, handles)
% hObject    handle to Slider_anular (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Posicion=get(hObject,'Value');
fwrite(handles.Serial,strcat('R',int2str(Posicion)))
set(handles.Text_mensajes,'String',strcat('Se ha cambiado la posicion ANULAR a --
>',int2str(Posicion)));
pause(0.8);
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% --- Executes on slider movement.
function Slider_menique_Callback(hObject, eventdata, handles)
% hObject    handle to Slider_menique (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Posicion=get(hObject,'Value');
fwrite(handles.Serial,strcat('P',int2str(Posicion)))
set(handles.Text_mensajes,'String',strcat('Se ha cambiado la posicion MENIQUE a --
>',int2str(Posicion)));
pause(0.8);
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% --- Executes on slider movement.
function Slider_pulgar_Callback(hObject, eventdata, handles)
% hObject    handle to Slider_pulgar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
Posicion=get(hObject,'Value');
fwrite(handles.Serial,strcat('T',int2str(Posicion)))
set(handles.Text_mensajes,'String',strcat('Se ha cambiado la posicion PULGAR a --
>',int2str(Posicion)));
pause(0.8);
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% --- Executes on button press in Boton_abrir.
function Boton_abrir_Callback(hObject, eventdata, handles)
% hObject    handle to Boton_abrir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
   set(handles.Slider_pulgar,'Value',10);
   set(handles.Slider_indice,'Value',10);
   set(handles.Slider_corazon,'Value',10);
   set(handles.Slider_anular,'Value',10);
   set(handles.Slider_menique,'Value',10);
   Posicion=10;
   fwrite(handles.Serial,strcat('M',int2str(Posicion)))
   fwrite(handles.Serial,strcat('T',int2str(Posicion)))
   fwrite(handles.Serial,strcat('I',int2str(Posicion)))
   fwrite(handles.Serial,strcat('R',int2str(Posicion)))
   fwrite(handles.Serial,strcat('P',int2str(Posicion)))
    set(handles.Text_mensajes,'String','Se ha ABIERTO la mano');

% --- Executes on button press in Boton_cerrar.
function Boton_cerrar_Callback(hObject, eventdata, handles)
% hObject    handle to Boton_cerrar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
   set(handles.Slider_pulgar,'Value',170);
   set(handles.Slider_indice,'Value',170);
   set(handles.Slider_corazon,'Value',170);
   set(handles.Slider_anular,'Value',170);
   set(handles.Slider_menique,'Value',170);
   Posicion=170;
   fwrite(handles.Serial,strcat('M',int2str(Posicion)))
   fwrite(handles.Serial,strcat('T',int2str(Posicion)))
   fwrite(handles.Serial,strcat('I',int2str(Posicion)))
   fwrite(handles.Serial,strcat('R',int2str(Posicion)))
   fwrite(handles.Serial,strcat('P',int2str(Posicion)))
    set(handles.Text_mensajes,'String','Se ha CERRADO la mano');
% --- Executes on selection change in Menu_puertos.
function Menu_puertos_Callback(hObject, eventdata, handles)
% hObject    handle to Menu_puertos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns Menu_puertos contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from Menu_puertos

% --- Executes on selection change in Menu_velocidad.
```

```
function Menu_velocidad_Callback(hObject, eventdata, handles)
% hObject    handle to Menu_velocidad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
var=get(hObject,'Value');
switch var
    case 1
        fwrite(handles.Serial,'V100');
        set(handles.Text_mensajes,'String','Se ha cambiado la velocidad a BAJA');
    case 2
        fwrite(handles.Serial,'V40');
        set(handles.Text_mensajes,'String','Se ha cambiado la velocidad a MEDIA');
    case 3
        fwrite(handles.Serial,'V20');
        set(handles.Text_mensajes,'String','Se ha cambiado la velocidad a ALTA');
end

% Hints: contents = cellstr(get(hObject,'String')) returns Menu_velocidad contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from Menu_velocidad

% --- Executes on button press in Boton_conectar.
function Boton_conectar_Callback(hObject, eventdata, handles)
% hObject    handle to Boton_conectar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Estado=get(hObject,'Value');
if Estado==1
   Posicion_puerto=get(handles.Menu_puertos,'Value');
   Var=get(handles.Menu_puertos,'String');
   set(hObject,'BackgroundColor','red');
   set(hObject,'String','Desconectar');
   set(handles.Slider_pulgar,'Enable','on');
   set(handles.Slider_indice,'Enable','on');
   set(handles.Slider_corazon,'Enable','on');
   set(handles.Slider_anular,'Enable','on');
   set(handles.Slider_menique,'Enable','on');
   set(handles.Menu_velocidad,'Enable','on');
   set(handles.Boton_cerrar,'Enable','on');
   set(handles.Boton_abrir,'Enable','on');
   set(handles.Reposo_Boton,'Enable','on');
   handles.Serial=serial(char(Var(Posicion_puerto)),'baudrate',57600);
   fopen(handles.Serial);
   set(handles.Text_mensajes,'String','Se ha conectado correctamente');
   handles.Ha_conectado=1;
   guidata(hObject, handles);
else
   set(hObject,'BackgroundColor','green');
   set(hObject,'String','Conectar');
   set(handles.Slider_pulgar,'Enable','off');
   set(handles.Slider_indice,'Enable','off');
   set(handles.Slider_corazon,'Enable','off');
   set(handles.Slider_anular,'Enable','off');
   set(handles.Slider_menique,'Enable','off');
   set(handles.Menu_velocidad,'Enable','off');
```

```
   set(handles.Boton_cerrar,'Enable','off');
   set(handles.Boton_abrir,'Enable','off');
    set(handles.Reposo_Boton,'Enable','off');
   fclose(handles.Serial);
   set(handles.Text_mensajes,'String','Se ha desconectado correctamente');
end
guidata(hObject, handles);




% Hint: get(hObject,'Value') returns toggle state of Boton_conectar


% --- Executes when user attempts to close Control.
function Control_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to Control (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
if handles.Ha_conectado==1
delete(handles.Serial);
end
delete(hObject);


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
foto=imread('uah.jpg');
image(foto);
axis off;
guidata(hObject, handles);

% Hint: place code in OpeningFcn to populate axes1


% --- Executes on button press in Reposo_Boton.
function Reposo_Boton_Callback(hObject, eventdata, handles)
 set(handles.Slider_pulgar,'Value',90);
   set(handles.Slider_indice,'Value',90);
   set(handles.Slider_corazon,'Value',90);
   set(handles.Slider_anular,'Value',90);
   set(handles.Slider_menique,'Value',90);
   Posicion=90;
   fwrite(handles.Serial,strcat('M',int2str(Posicion)))
   fwrite(handles.Serial,strcat('T',int2str(Posicion)))
   fwrite(handles.Serial,strcat('I',int2str(Posicion)))
   fwrite(handles.Serial,strcat('R',int2str(Posicion)))
   fwrite(handles.Serial,strcat('P',int2str(Posicion)))
```

7 PRESUPUESTO

En la siguiente tabla se recoge un presupuesto del coste total de este proyecto

7.1 COSTES MATERIALES

	Concepto	Coste /Unidad	Unidades	Coste total
Hardware	Ordenador	600,00€	1	600,00€
	Impresora 3D	550,00€	1	550,00€
	Arduino UNO R3	25,00€	1	25,00€
	PCB	10,00€	1	10,00€
	Material electrónico	15,00€	1	15,00€
	Bobina de 10 M PLA	25,00€	1	25,00€
Software	Matlab R2016B (Versión Educación)	500,00 €	1	500,00 €
	Windows 10	0,00 €	1	0,00 €
	Microsoft Office 365	150,00 €	1	150,00 €
		Total		**1.875,00 €**

Tabla 7-1Costes materiales:

7.2 COSTES TOTALES

Los costes totales incluyen los costes materiales, profesionales y el IVA.

Concepto	**Costes totales**
Costes materiales	1.875,00€
Tasas profesionales	5.000,00€
Subtotal	**6.875,00 €**
IVA (21%)	1.443,75 €
TOTAL	**8.318,75 €**

Tabla 7-2: Costes totales

8 BIBLIOGRAFÍA

[1] «Acerca de la mano robótica Shadow Dexterous,» [En línea]. Available: http://www.robotnik.es/manos-roboticas/mano-robotica-shadow-dexterous/.

[2] Z. Xu y E. Todorov, «Acerca de la mano robotica de la universidad de Washington,» [En línea]. Available: https://spectrum.ieee.org/automaton/robotics/medical-robots/biomimetic-anthropomorphic-robot-hand.

[3] «Sitio del Programa Delfin,» [En línea]. Available: https://www.programadelfin.com.mx/.

[4] G. Langevin, «Mano Inmoov,» [En línea]. Available: http://inmoov.fr/.

[5] «Página oficial de SolidWorks,» [En línea]. Available: http://www.solidworks.es/.

[6] «Página oficial Arduino,» [En línea]. Available: https://www.arduino.cc/.

[7] «Acerca del robot IRB120,» [En línea]. Available: http://new.abb.com/products/robotics/es/robots-industriales/irb-120.

[8] «Libreria "Servo.h",» [En línea]. Available: https://www.arduino.cc/en/Reference/Servo.

[9] A. Gutiérrez Corbacho, Desarrollo de una interfaz para el control del robot IRB120 desde Matlab, 2014.

[10] M. L. Fernández Iglesias, Diseño y fabricación de garras para un IRB120, Sevilla, 2017.

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR