TRABAJO DE FIN DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

DISEÑO E IMPLEMENTACIÓN DE UNA PINZA SERVOCONTROLADA

Autor: **Diego Antolí Soler**

Tutor: **Ranko Zotovic Stanisic**

Curso Académico: 2018-19

RESUMEN

El objetivo de este proyecto consiste en el diseño y puesta en marcha de las funciones básicas de una pinza robótica de bajo coste con la finalidad que pueda ser usada como prótesis biónica en minusválidos para realizar las principales actividades del día a día.

Para ello, se diseñará e imprimirá en 3D la pinza y sus componentes; a continuación, se diseñará toda la electrónica necesaria, se programará el *software* necesario en el microcontrolador *Discovery STM32F429I* y se procederá al montaje de esta.

Finalmente, se realizarán los controles de posición, velocidad y fuerza para que la pinza sea capaz de seguir las órdenes que se le envíen con la mayor precisión posible y utilizando un único servomotor de bajo coste al cual se elimina la electrónica interna para diseñar dichos controles.

Palabras Clave: Pinza robótica, STM32F4, control, PID, prótesis, sensores, servomotor

RESUM

L'objectiu d'aquest projecte consisteix en el disseny y posada en marxa de les funcions bàsiques d'una pinça robòtica de baix cost amb la finalitat que puga ser utilitzada com a pròtesi biònica pels minusvàlids per a realitzar les principal activitats del dia a dia.

Per a això, es dissenyarà i imprimirà en 3D la pinça y els seus components; a continuació, es dissenyarà tota la electrònica necessària, es programarà el *software* necessari al microcontrolador *Discovery STM32F429I* i es procedirà al muntatge d'aquesta.

Finalment, es realitzaran els controls de posició, velocitat y força per a que la pinça siga capaç de seguir les ordres que se li envien amb la major precisió possible y emprant un únic servomotor de baix cost al qual s'elimina l'electrònica interna per a dissenyar aquests controls.

Paraules Clau: Pinça robòtica, STM32F4, control, PID, sensors, servomotor

ABSTRACT

The objective of this project is the design and implementation of the basic functions of a low-cost robotic gripper in order that it can be used as a bionic prosthesis for disabled people in order to carry out the main day to day activities.

For this purpose, the clamp and its components will be designed and printed in 3D; next, all the necessary components will be designed, then the necessary software will be programmed in the *Discovery STM32F429I* and the assembly will proceed.

Finally, the position, speed and force controls will be programmed so that the robotic gripper will be able to follow the orders as accurately as possible and using a single low-cost servomotor. The internal electronics of the servomotor will be eliminated.

Keywords: Robotic gripper, STM32F4, control, PID, prosthesis, sensors, servomotor

Índice

Índice de figuras

Índice de tablas

Glosario:

- **ABS:** Acrilonitrilo butadieno estireno.
- **Control P:** Control proporcional.
- **Control PD:** Control proporcional-derivativo.
- **Duty Cycle:** Ciclo de trabajo.
- **EEG:** Electroencefalograma.
- **EMG:** Electromiografía
- **FSR:** *(Force Sensing Resistor)* sensor de fuerza resistivo.
- **GPIO:** *(General Purpose Input-Output)* entrada-salida de propósito general.
- **N:** Newton.
- **PCB:** *(Printed Circuit Board)* circuito impreso.
- **PLA:** Ácido poliláctico.
- **PWM:** *(Pulse Modulation Width)* modulación de ancho de pulso
- **TFG:** Trabajo de Fin de Grado.
- **V:** Voltio.

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

DISEÑO E IMPLEMENTACIÓN DE UNA PINZA SERVOCONTROLADA

MEMORIA

Autor: **Diego Antolí Soler**

Tutor: **Ranko Zotovic Stanisic**

Curso Académico: 2018-19

MEMORIA

1. Objetivo y Justificación

1.1. Objetivo general

El objetivo de este proyecto consiste en el diseño, impresión 3D e implementación de una pinza servocontrolada de bajo coste que pueda servir como ayuda en las tareas del día a día de una persona minusválida.

Cabe destacar que este proyecto es una de mejora del Proyecto de Final de Máster realizado por Jefferson Fernando Camacho Muñoz en el año 2016 [1].

1.2. Objetivos específicos

- Diseñar una pinza de tamaño reducido que pueda sostener objetos de tamaño medio (botellas, bolígrafos, gafas, cubiertos, etc.) con un rango de pesos entre 0 y 500 gramos.
- Impresión 3D de la estructura de la pinza.
- Implementación de todo el software en una única tarjeta controladora (*STM32F429I*).
- Mantener el coste de fabricación por debajo de 200 euros.
- Fabricar las piezas con materiales reciclables o que se puedan reutilizar.
- Diseñar los controladores de posición, velocidad y fuerza para el servomotor lo más simples posibles.
- Implementar sensores de fuerza que detecten la presión ejercida sobre los objetos.

1.3. Justificación

A lo largo de los años, la tecnología ha ido avanzando de tal manera que ha permitido la creación de dispositivos de ayuda a las personas discapacitadas, siendo estos cada vez más pequeños, precisos y duraderos. El principal problema de estos nuevos dispositivos radica en los altos precios que estos tienen, por lo que no toda la población con alguna discapacidad puede disponer de uno de ellos.

Es por esto por lo que se plantea el siguiente proyecto, debido a que se persigue el objetivo de conseguir diseñar una pinza robótica que cumpla con las necesidades básicas al menor precio y con la mayor durabilidad posible. Para ello, se busca un diseño simple y la utilización de la menor cantidad de componentes electrónicos con la finalidad de reducir las posibles averías.

Finalmente, esto permitiría que este tipo de tecnología fuera asequible a toda la población con este tipo de necesidades, permitiendo incluso su llegada a países subdesarrollados en los que la proliferación de enfermedades e infecciones incrementan el número de amputaciones de brazos.

1.4. Motivación

El motivo que justifica la elección personal de este proyecto radica en que en verano de 2018 imprimí en 3D y ensamblé una mano robótica llamada *InMoov* (siguiendo el TFG de Pablo Ropero Giralda [2]) la cual posee cinco dedos que son capaces de moverse de manera independiente mediante servomotores. Al acabar este proyecto, comprobé que su funcionamiento era simple y que me gustaría mejorarlo. Fue cuando me propuse mejorar esta mano robótica, pero como hubiera sido demasiado complejo el desarrollo de la mano con cinco motores independientes se simplificó en el desarrollo de la pinza servocontrolada que se describe a lo largo de este proyecto.

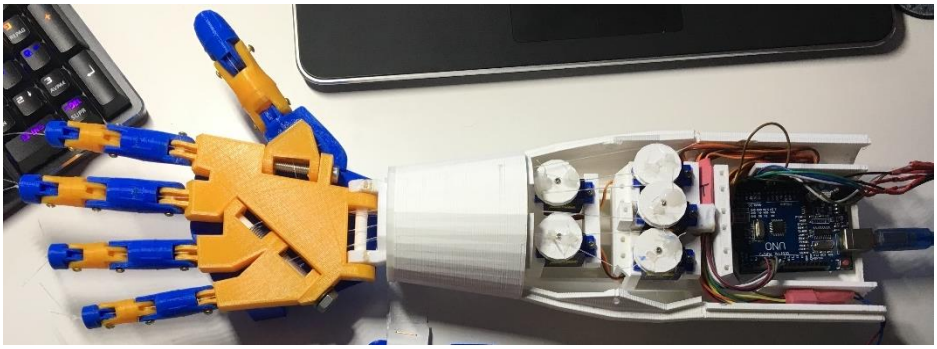


Figura 1. Proyecto personal mano InMoov

1.5. Metodología

La metodología de trabajo a seguir para el correcto desarrollo de este proyecto es la siguiente:

1. En primer lugar, se determinan los parámetros de diseño sobre los cuales se tendrá que fabricar la pinza. Estos parámetros son, entre otros, rango de pesos, tamaños y tipos de objetos a sujetar; o, fuerza y par a ejercer por el servomotor. También se eligen los sensores y el servomotor adecuados para cumplir con estos parámetros y, finalmente se comparan diferentes tipos de pinzas analizando sus ventajas e inconvenientes.
2. En segundo lugar, se procede al diseño 3D de todas las piezas que compondrán el prototipo de la pinza. Estas piezas se ensamblan mediante el software 3D para comprobar que todas ellas encajan correctamente, y el movimiento de apertura y cierre es el deseado.
3. En tercer lugar, una vez hechas las mejoras pertinentes al diseño, se procede a la impresión 3D del prototipo. Una vez impreso, este se ensambla. En el caso que hubiera algún problema estructural con el montaje de la pinza se puede recurrir al paso número 2.
4. En cuarto lugar, y en paralelo con los pasos dos y tres, se procede al montaje de los diferentes circuitos que contendrán toda la electrónica (puente en H, amplificadores operacionales, motores, etc.) que permite el funcionamiento de la pinza.
5. En quinto lugar, se programará todo el software (interrupciones, ADC's, Timers, señal PWM, etc.) en el microcontrolador *Discovery STM32F429I* y se conectará con los circuitos montados en el paso cuatro, realizando las pruebas pertinentes para comprobar el correcto funcionamiento del conjunto.
6. En sexto lugar, se monta el motor y los sensores sobre la pinza.

7. En séptimo lugar, se procede a programar los controles de posición, velocidad y fuerza.
8. Finalmente, en octavo lugar, se realizan pruebas y simulaciones de funcionamiento de los diferentes controles.

En caso de ser necesario, se rediseñará/reemplazará todo aquello que no cumpla con los objetivos expuestos en este proyecto.

Esta metodología se resume en la siguiente figura:

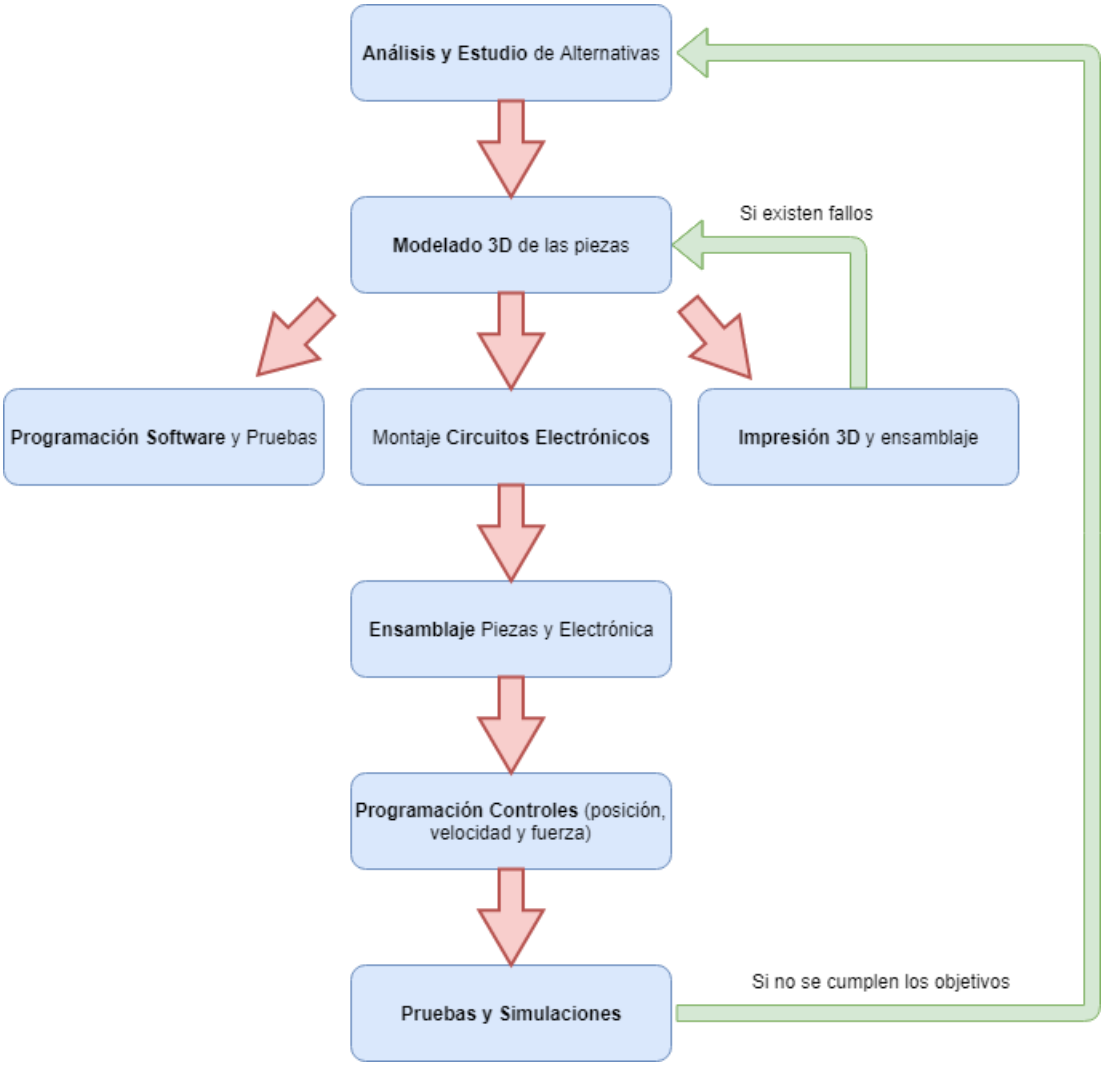


Figura 2. Metodología empleada para la realización del proyecto

2. Estado del arte

2.1. Introducción. Historia de las prótesis.

Los primeros indicios de las primeras prótesis robóticas datan de la época de las pirámides. Desde entonces, el campo de la protésica se ha convertido en un claro ejemplo del empeño de las personas por encontrar soluciones a los problemas que surjan [3, 4].

Los egipcios destacaron por hacer ser los pioneros en el campo de la protésica, fabricando los diferentes tipos de prótesis hechos de diferentes tipos de fibras, siendo estas muy simples y rudimentarias, sin ningún tipo de tecnología.

En la Edad Media, apenas hubo avances en este campo más allá de las típicas patas de palo o el gancho de la mano. La mayoría de las prótesis que se fabricaban en la época no eran ni siquiera funcionales puesto que en la mayoría de los casos se utilizaban para disimular las heridas o las deformidades, y en la guerra se colocaban prótesis cuya única utilidad era sujetar un escudo o una lanza. Cabe destacar que solamente los ricos eran capaces de costear el alto precio de las prótesis más avanzadas.

Durante el Renacimiento, las prótesis se elaboraban principalmente con acero, hierro, cobre y madera. En esa época se consiguió construir la primera mano de hierro tecnológicamente avanzada. Esta era capaz de moverse mediante una serie de mecanismos de liberación y resortes, como la que se muestra en la Figura 3.

Figura 3. Prótesis del Renacimiento

Entorno a finales del 1500, el cirujano y barbero del ejército francés Ambroise Paré, también conocido como el padre de la cirugía de amputación y del diseño protésico moderno, inventó un dispositivo similar a una pata de palo, pero con la mejora de que esta podía flexionarse por la parte de la rodilla, como se observa en la siguiente imagen (Figura 4).

Figura 4. Prótesis Francesa

Durante la Guerra Civil Estadounidense, la cantidad de amputaciones se incrementó muchísimo, por lo que se llegaron a fabricar prótesis con casi cualquier tipo de material disponible. Fue Gustav Hermann quien empezó a usar aluminio en sus prótesis para hacerlas más ligeras en lugar de acero debido al elevado peso de este.

No fue hasta principios de los años 70 cuando se empezaron a ver intentos de aplicar la robótica en el campo de las prótesis.

2.2. Principales consideraciones de las prótesis robóticas actualmente

Los avances en robótica industrial de los últimos años han permitido ampliar los campos de aplicación de esta hacia horizontes como el de las prótesis robóticas, también conocidas como prótesis biónicas.

La robótica, al estar relacionada con gran parte de campos de la tecnología (electrónica, informática, mecánica y control) permite suplir o solucionar la mayor parte de los problemas físicos de las personas. La Tabla 1 representa la estrecha relación entre las diferentes partes constituyentes de una persona y un robot:

Persona	Robot
Cerebro	Microprocesador – Ordenador
Cuerpo	Estructura
Músculos	Motores
Sentidos	Sensores

Tabla 1. Símil Persona-Robot

Fue a partir de los años 70 cuando se realizaron las primeras aplicaciones de robótica dentro del campo de la discapacidad, tomándose señales mioeléctricas como señales de control sobre el propio paciente.

En ese momento, se determinó que este tipo de prótesis robóticas no serían capaces de realizar todas las tareas requeridas por un usuario con discapacidad, por lo que se concluyó que este tipo de tecnología tenía que estar enfocada para suplir al menos las necesidades más básicas, como, por ejemplo, en el caso de una prótesis de mano, coger objetos tales como vasos o comida, dejando de lado tareas más complicadas como por ejemplo pasar las hojas de un libro.

Por otra parte, en el diseño de una prótesis robótica también existe la parte estética que siempre suele estar reñida con su funcionalidad. Esta problemática junto con los ruidos generador por los motores o el tamaño de las baterías son factores que a lo largo de los años se han ido subsanando, haciendo las prótesis silenciosas y lo más discretas posible, como se observa en la Figura 5:

Figura 5. Prótesis moderna de estilo realista

2.3. Fases en el desarrollo de una prótesis robótica

Para el correcto funcionamiento de una prótesis robótica, esta debe contener como mínimo las siguientes fases:

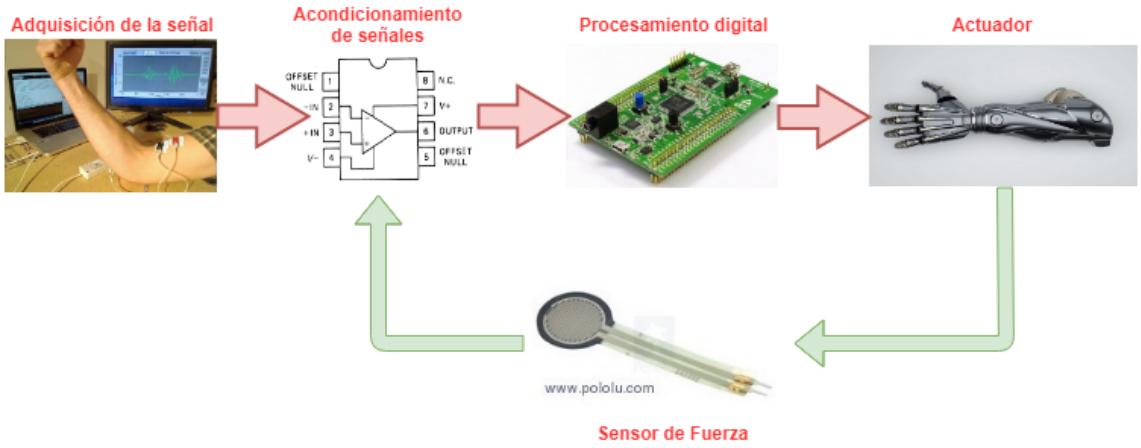


Figura 6. Fases en el desarrollo de una prótesis robótica

2.3.1. Adquisición de la Señal

Las dos formas más comunes para adquirir señales de un cuerpo humano son por electromiografía o a través de impulsos cerebrales.

2.3.1.1. Electromiografía

La electromiografía (EMG) consiste en la adquisición de señales eléctricas producidas por un músculo durante su contracción y su relajación. Para obtener estas señales basta con colocar una serie de electrodos sobre el músculo el cual se desean obtener los impulsos eléctricos. La siguiente imagen (Figura 7) ilustra los electrodos colocados sobre el bíceps:

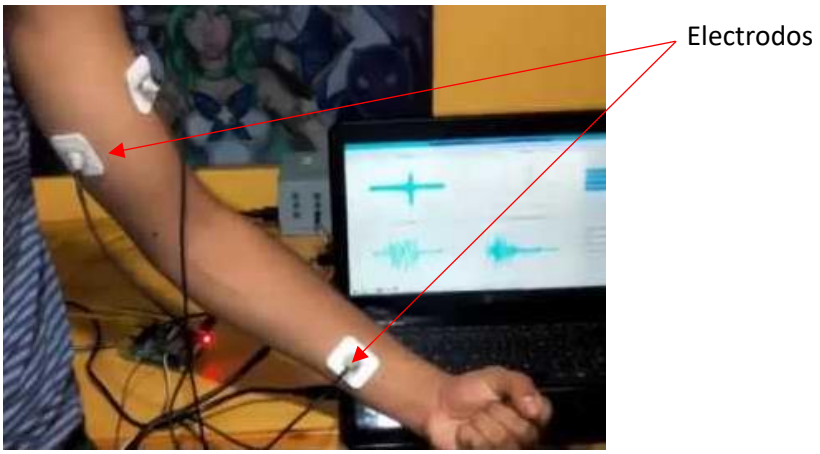


Figura 7. Captación de señales electromiográficas

2.3.1.2. Impulsos Cerebrales – EEG

La electroencefalografía o EEG se basa en el registro de la actividad bioeléctrica cerebral. Se realiza mediante una serie de electrodos que registran las corrientes eléctricas que se forman entre las neuronas del cerebro. Los electrodos son colocados como se muestran en la Figura 8:

Figura 8. Captación de señales por electroencefalografía[1]

2.3.2. Procesamiento Digital

La utilización de un microcontrolador en una aplicación de robótica es esencial, puesto que se trata del cerebro de todo el sistema, encargándose de recibir las lecturas de los sensores, y tomar decisiones para accionar los actuadores en función de su programación interna.

Actualmente, existen una gran cantidad de microcontroladores que con el paso de los años han reducido considerablemente su tamaño y su precio, y han aumentado su potencia de procesamiento. Esto permite su implementación en cualquier tipo de dispositivo. Los microprocesadores más conocidos son *Arduino, STMicroelectronics* y *RaspberryPi.*

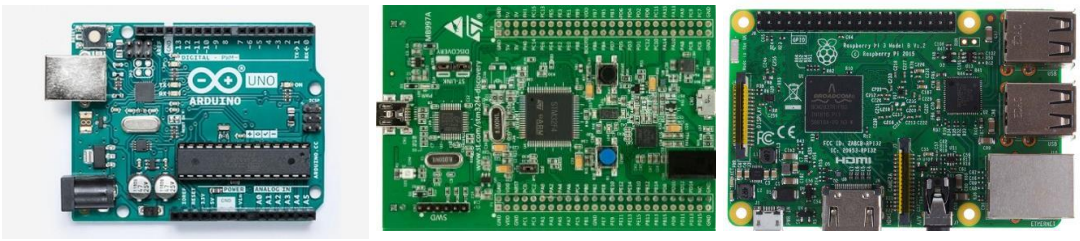


Figura 9. Tipos de microcontroladores: Arduino, STM, Raspberry

2.3.3. Actuadores – Motores

La mayor parte de los actuadores suelen ser motores eléctricos, pero en algunos casos también se utilizan pistones con accionamiento hidráulico. Los motores eléctricos más utilizados son los servomotores, puesto que estos están formados por un motor DC y una caja de engranajes que se encarga de transformar la velocidad en par, lo que le permite ubicarse en cualquier posición y mantenerse estable en dicha posición ante algún tipo de perturbación. La mayoría de los servomotores tienen un rango de giro limitado, normalmente 180 grados. Algunas de las principales ventajas de los servomotores son su pequeño tamaño y peso; y un consumo energético reducido por lo que son ideales para su implementación sobre una prótesis robótica.

[1] https://www.webconsultas.com/pruebas-medicas/electroencefalograma-eeg-12529

2.3.4. Sensores de Fuerza

Uno de los sensores más simples que se puede utilizar en el campo de la protésica es el sensor de fuerza resistivo (Figura 10), el cual varía su resistencia según la presión que se aplique sobre su área de funcionamiento. A mayor fuerza, menor será su resistencia.

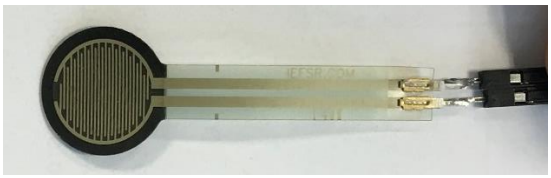


Figura 10. Sensor de fuerza resistivo

2.3.4.1. Tecnología SynTouch

En cuanto a la sensorización, la detección de fuerzas en una mano robótica es muy importante puesto que no se puede ejercer la misma fuerza para sujetar un bolígrafo o un huevo, puesto que este último podría romperse si se ejerce una fuerza excesiva sobre él.

Para ello, la empresa americana *SynTouch* [4] (líderes mundiales en Machine Touch®) ha desarrollado el producto *BioTac.* Se trata de un sensor táctil de altísima precisión con una detección táctil similar a la humana. Esta tecnología es capaz de detectar fuerza, vibraciones e incluso la temperatura simulando el tacto humano. Como se puede observar en la Figura 11, el sensor táctil *BioTac* está compuesto por un núcleo rígido donde se ubica toda la electrónica. Esta a su vez, está rodeada por una piel elástica llena de un líquido, siendo similar a la punta de un dedo humano.

Gracias a este tipo de sensor, el usuario que lo lleva incorporado en su prótesis robótica es capaz de sujetar todo tipo de objetos independientemente de su estructura, permitiéndole coger objetos frágiles sin miedo a que estos sean aplastados por la pinza.

Figura 11. Dedos biónicos con tecnología BioTac

2.3.5. Materiales de Construcción

Actualmente, las prótesis robóticas están construidas con los siguientes materiales [5]:

- Materiales termoplásticos: como por ejemplo el polietileno y el propileno, los cuales se pueden modelar fácilmente con la aplicación de calor. Se suelen utilizar para fabricar conexiones protésicas y componentes estructurales.
- Siliconas: se suele utilizar como relleno para encajes.
- Metales: como por ejemplo el aluminio y el titanio, dejando de lado el acero por ser más pesado. Se suelen utilizar como elementos estructurales y para la construcción de codos y rodillas.
- Espumas, fundas, etc.: se utilizan con la finalidad de generar un acabado cosmético de la prótesis. En la Figura 12, se muestra como en la actualidad se pueden conseguir unos acabados ultra realistas.

Figura 12. Diferentes tipos de prótesis hiperrealistas

2.3.5.1. Impresión 3D

Cabe destacar que en algunas ocasiones se utiliza la impresión 3D para fabricar determinadas piezas o componentes. Esto es debido a que este novedoso método de impresión de piezas con plástico permite fabricar toda clase de piezas estructurales o mecánicas.

Este método no se suele utilizar en los modelos finales de las prótesis porque es menos estético y realista que las siliconas que se emplean para los recubrimientos de estas. En cambio, sí que se suele utilizar bastante la impresión 3D cuando se está diseñando una prótesis, para la impresión de prototipos, ya que sin necesidad de maquinaria ni moldes de alto coste se puede conseguir un prototipo con el que hacer las pruebas pertinentes previas al modelo final.

Algunos de los modelos más conocidos de manos robóticas impresas en 3D son los siguientes:

- InMoov[2]: no se trata únicamente de una mano robótica, sino de un robot humanoide de código abierto impreso en 3D y controlado mediante Arduino. InMoov fue creado por el francés Gaël Langevin en 2011. Este robot se creó con la finalidad de ser un proyecto de aprendizaje de impresión 3D y robótica.

  En las siguientes figuras, se puede apreciar el robot InMoov (Figura 13) y un detalle de la mano robótica (Figura 14):

[2] http://inmoov.fr/

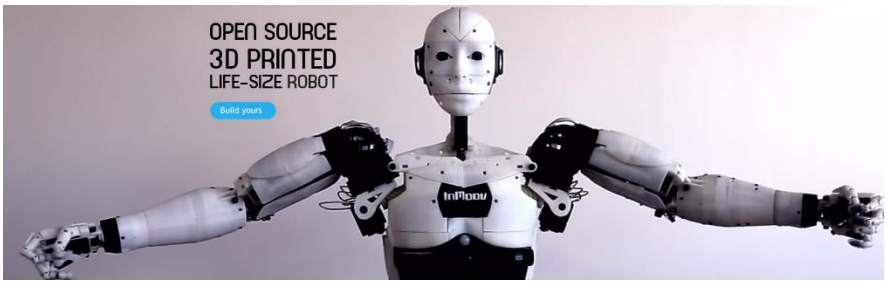


Figura 13. Humanoide InMoov

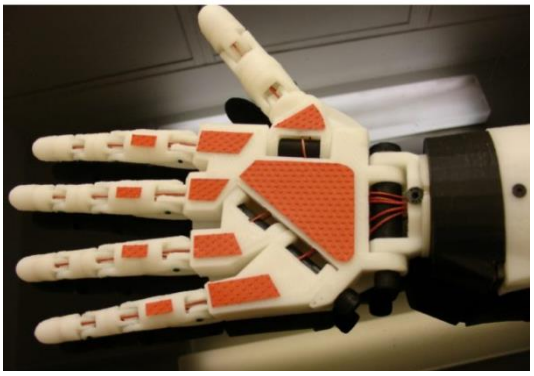


Figura 14. Mano InMoov

- YouBionic[3]: Se trata de un concepto futurista de una mano robótica, creada por el italiano Federico Ciccarese a finales de 2016. Al igual que la anterior, está totalmente impresa en 3D con el añadido de que los dedos de la mano robótica son accionados con las señales recibidas por unos sensores resistivos colocados en los dedos del sujeto, con lo que se consigue imitar con bastante precisión los movimientos de los dedos. En la siguiente figura, se muestra uno de los modelos de mano robótica creado por Ciccarese:

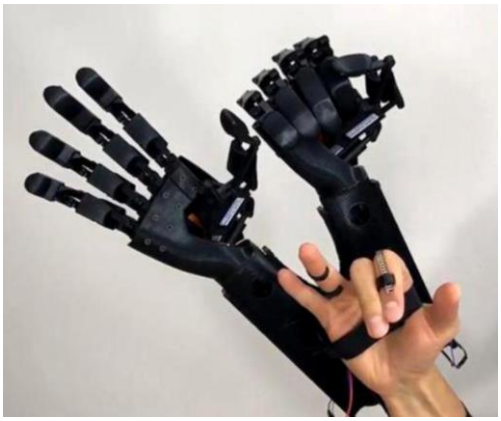


Figura 15. Manos de YouBionic

[3] https://www.youbionic.com/

2.3.6. Tipos de Manos Robóticas

Existen multitud de modelos de prótesis de manos robóticas, por lo que en este apartado se muestran las que en cierto modo han determinado el camino a seguir de las prótesis más modernas. Se pueden destacar las siguientes:

- Cyberhand: esta mano posee cinco dedos y seis motores, uno de los cuales se encarga del giro de muñeca (Figura 16).

Figura 16. Mano Cyberhand

- Sensorhand: esta mano se caracteriza por ser más rápida que la anterior, poseyendo únicamente tres dedos a modo de pinza (Figura 17).

Figura 17. Mano Sensorhand

- I-Limb: esta mano posee cinco dedos, y puede funcionar mediante señales electromiográficas (Figura 18).

Figura 18. Mano I-Limb

- DLR Hand II: esta mano consta de cuatro dedos, con cuatro articulaciones y cuatro grados de libertad. Además, es flexible y agarra objetos con facilidad (Figura 19).

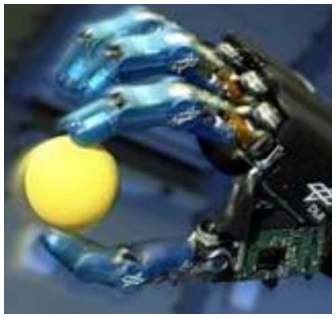


Figura 19. Mano DLR Hand II

- Shadow Hand: esta mano posee 5 dedos y usa un actuador neumático por lo que sus movimientos son suaves. Este tipo no es muy útil debido a que el sistema completo pesaría y ocuparía demasiado volumen (Figura 20).

Figura 20. Mano Shadow Hand

2.3.7. Alimentación - Baterías

El tipo de batería que más se emplea en la actualidad es la batería de ion de litio (Li-Ion). Sus principales características son:

- Ligeras.
- Elevada capacidad energética.
- Resistencia a la descarga (Reducido efecto memoria).
- Elevado número de ciclos de carga.

Figura 21. Batería ion de litio

Estas baterías también poseen algunos defectos:

- Rápida degradación.
- Sensibilidad a temperaturas elevadas.

El funcionamiento inadecuado de estas baterías (Figura 21) puede provocar su destrucción por inflamación o incluso pueden llegar a explotar, por lo que hay que trabajar con ellas bajo unas condiciones controladas.

3. Especificaciones de diseño

- El diseño de la pinza debe permitir que se puedan sujetar objetos con un diámetro similar al de una botella de 1 L (en torno a 100 mm de diámetro). También se deberán poder sujetar objetos de pequeño tamaño como cubertería, gafas, botes, etc.
- La pinza debe imprimirse en 3D, por lo que su diseño ha de ser el más simple posible y adaptado a las limitaciones de la impresión 3D.
- El prototipo ha de imprimirse en plástico PLA por ser de origen vegetal (almidón de maíz o caña de azúcar) y biodegradable.
- El peso de la estructura junto con los componentes electrónicos y el microprocesador no debe superar 1 Kg.
- Coste de fabricación y ensamblaje del prototipo inferior a 200 euros.
- Colocación de sensores de presión tanto en las puntas de los dedos como en la parte interior de estos.
- El servomotor encargado del movimiento de la pinza debe otorgar el par suficiente como para sujetar los objetos con firmeza.
- Los controles de posición, velocidad y fuerza deben ser lo más simples posibles con la finalidad de no sobrecargar el microprocesador.

4. Desarrollo del proyecto

4.1. Software necesario para la elaboración del proyecto

Para la realización de este proyecto es necesario el siguiente software informático:

- *Autodesk Fusion 360* (diseño, modelado y ensamblado 3D)
- *Simplify 3D* (Impresión 3D)
- *Proteus 8* (elaboración de esquemas eléctricos)
- *Keil uVision 5* (Programación de microcontroladores)
- *Matlab R2018* (tratamiento y representación gráfica de datos)

4.2. Desarrollo de la pinza

4.2.1. Objetos susceptibles de ser agarrados por la pinza

En la siguiente lista se muestra una serie de objetos que la pinza debería poder agarrar para cumplir con las especificaciones básicas:

- Cubertería
- Vasos
- Bolígrafos
- Botellas de al menos 500 mL
- Gafas
- Botes
- Latas
- Platos

4.2.2. Comparativa de los diferentes modelos de pinza

Tras determinar el tipo de objetos que la pinza tiene que sujetar, se realiza una comparativa de dos diseños de pinzas que podrían servir para esta aplicación:

En la Figura 22, se tiene un tipo de pinza simple el diseño de la cual es muy simple y además su construcción es robusta, pero su principal defecto radica en el pequeño rango de objetos que esta sería capaz de sujetar puesto que al tener solamente dos dedos con pequeñas superficies de contacto se requeriría de una gran fuerza para sostener un objeto, por lo que este tipo de pinza no es el adecuado.

Figura 22. Modelo Pinza simple[4]

[4] https://www.thingiverse.com/thing:969447

Por tanto, se va a emplear el diseño de la pinza creada por Jefferson Fernando Camacho Muñoz en su TFG [1], ya que como se muestra en el siguiente croquis (Figura 23), esta dispone de mayor ángulo de apertura junto con cuatro de dedos de grandes superficies las cuales permitirían un mejor agarre de los objetos sin realizar grandes fuerzas.

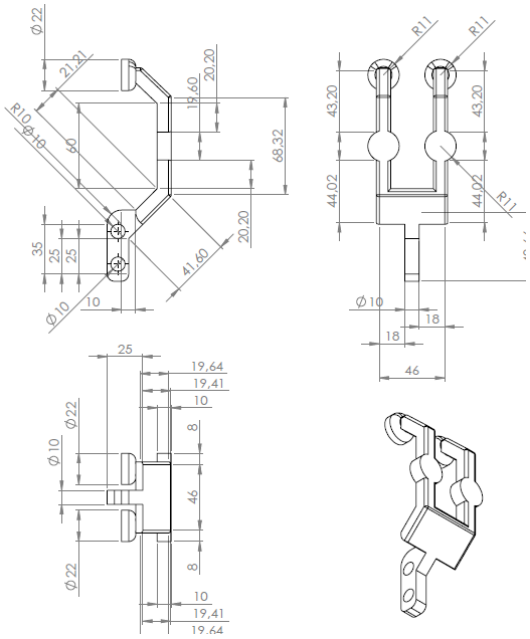


Figura 23. Planos Pinza por utilizar

4.2.3. Diseño CAD de los diferentes componentes

Para el diseño de la pinza, se parte del diseño elegido en el apartado 4.2.2. A partir del plano de la Figura 23, se realizan algunas mejoras tanto estéticas como estructurales en el modelado de las diferentes piezas.

El software por emplear para realizar el diseño CAD de las piezas es *Autodesk Fusion 360* debido a su multitud de funcionalidades (modelado 3D, simulación, ensamblaje, elaboración de planos normalizados, etc.).

El prototipo está formado por las siguientes piezas:

- Estructura dedos
- Soporte sensor superior
- Cara inferior de unión
- Cara superior de unión
- Eje engranaje motor
- Eje engranaje libre
- Eje final de carrera
- Varillas roscadas y tuercas
- Soporte servomotor
- Caja electrónica

En el apartado 8 (Planos), se encuentran todos los planos acotados de las piezas diseñadas.

4.2.3.1. Estructura Dedos

Se utilizan dos unidades las cuales forman la estructura de agarre de la pinza. Como se puede observar en la Figura 24, se ha dejado un desnivel en la parte de los dedos para colocar los sensores sobre las zonas circulares junto con el material antideslizante para mejorar el agarre sobre los objetos. Además, se ha dejado hueco el interior de la parte de los dedos para poder pasar todo el cableado de los sensores por dentro de estos huecos.

Figura 24. Diseño Estructura Dedos

4.2.3.2. Soporte Sensor Superior

Sobre esta pieza se colocarán los sensores de los dedos situados en los extremos de la pinza, por lo que son necesarias cuatro piezas de este tipo (Figura 25). Esta posee al igual que la pieza anterior un desnivel donde se colocará el sensor fuerza y el material antideslizante.

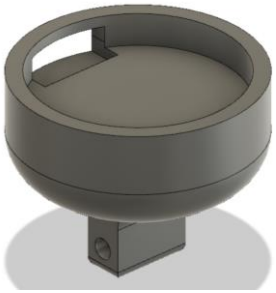


Figura 25. Diseño Soporte Sensor Superior

Esta pieza no es fija, sino que tiene cierto ángulo de movimiento lo cual le permite adaptarse al objeto que se quiera sujetar (Figura 26).

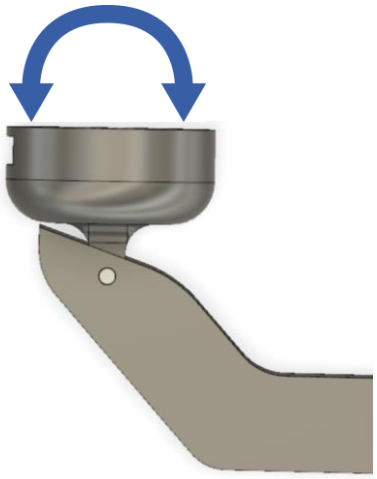


Figura 26. Representación del movimiento de Soporte Sensor Superior

4.2.3.3. Cara Inferior de Unión

La función de esta pieza (Figura 27) junto con su análoga superior, es sujetar el servomotor y la estructura de los dedos.

Como se observa, esta pieza posee dos salientes perforados ubicados en los laterales de la pieza con la finalidad de albergar el servomotor. Los orificios del centro de la pieza corresponden con los huecos donde se ubicarán los ejes de rotación de la pinza.

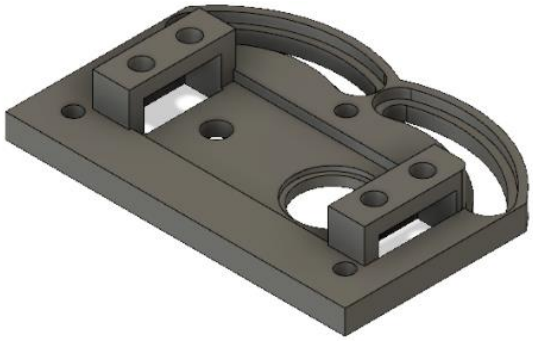


Figura 27. Diseño Cara Inferior de Unión

El conjunto ensamblado del servomotor junto con la cara inferior de unión se muestra en la Figura 28.

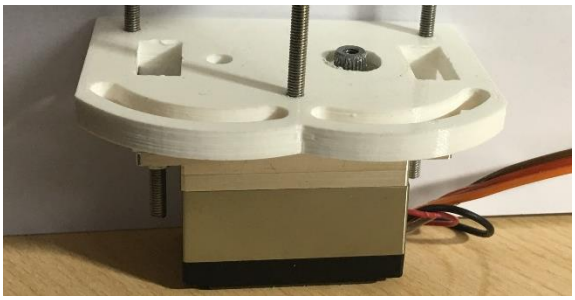


Figura 28. Conjunto Servomotor y Cara Inferior de Unión

4.2.3.4. Cara Superior de Unión

La funcionalidad de esta pieza es la misma que la de la pieza anterior, sujetar firmemente todo el conjunto de la pinza. El resultado se muestra en la Figura 29.

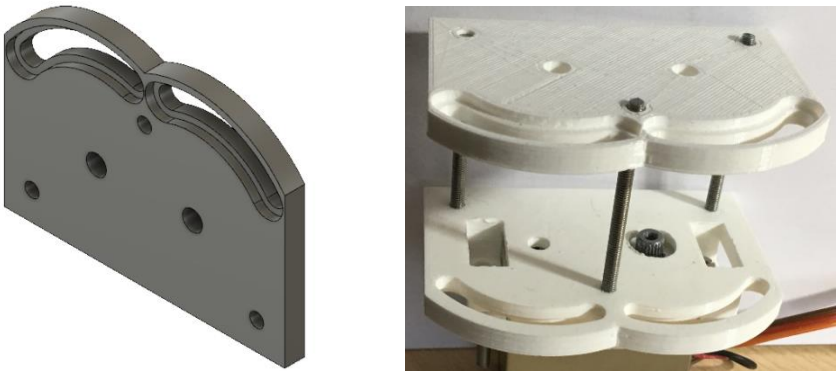


Figura 29. Diseño Cara Superior de Unión y ensamblado junto con Cara Inferior de Unión

4.2.3.5. Eje Engranaje Motor, eje engranaje libre y eje final de carrera

Como se observa en la Figura 30, la pieza está formada por varios cilindros de diferentes diámetros, facilitando así que cada pieza encaje en su lugar. Por ejemplo, el diámetro de mayor tamaño encaja con uno de los orificios inferiores de la pieza "Estructura Dedos" formando así el eje de rotación accionado por el engranaje conectado al servomotor.

Figura 30. Diseño Eje Engranaje Motor

La siguiente pieza (Figura 31), es muy similar a la anterior, y su finalidad es la misma pero esta encaja en la segunda "Estructura Dedos" y su movimiento de rotación estará determinado por la transmisión de movimiento de un engranaje al contiguo.

Figura 31. Diseño Eje Engranaje Libre

Finalmente, esta última pieza (Figura 32) de menor altura que las dos anteriores, se coloca en uno de los agujeros de la parte inferior de "Estructura Dedos" y su finalidad es la accionar un final de carrera cuando la pinza llegue a su máxima apertura.

Figura 32. Diseño Eje Final de Carrera

Al ensamblar el conjunto de piezas anterior, el resultado es el que se muestra a continuación (Figura 33).

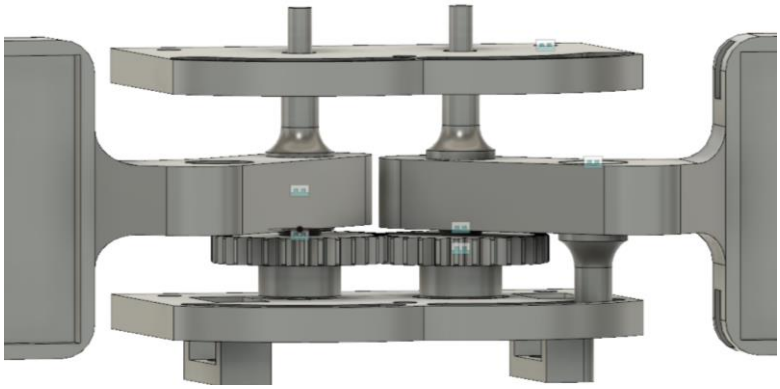


Figura 33. Ensamblado de los ejes junto con Estructura Dedos

4.2.3.6. Varillas roscadas y tuercas

Con la finalidad de poder unir "Cara Superior de Unión" y "Cara Inferior de Unión" se utilizan tres varillas roscadas de acero zincado de métrica M3 cortadas a cuatro centímetros; y catorce tuercas de la misma métrica para asegurar una correcta fijación de las diferentes piezas. El resultado es el que se muestra en la siguiente en la Figura 34.

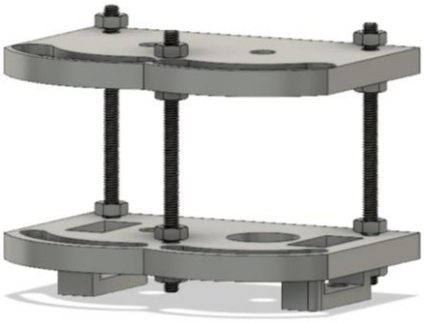


Figura 34. Ensamblado de las varillas roscadas junto con los soportes superior e inferior

4.2.3.7. Soporte servomotor

La pieza que se muestra a continuación (Figura 35) no forma parte de la estructura de la pinza, pero se ha diseñado con la finalidad de sujetar el prototipo de la pinza por la base del servomotor, permitiendo de esta manera realizar las pruebas de funcionamiento de manera más sencilla.

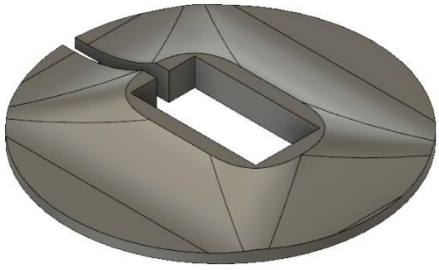


Figura 35. Diseño Soporte Servomotor

4.2.3.8. Caja electrónica

Esta pieza ha sido diseñada con la finalidad de albergar en su interior todos los componentes electrónicos y circuitos necesarios para el funcionamiento del sistema, así como el cableado y el microcontrolador. Esta se ubica en la parte trasera de la pinza, en lo que sería el antebrazo de la persona discapacitada. En la siguiente figura (Figura 36) se puede observar su diseño:

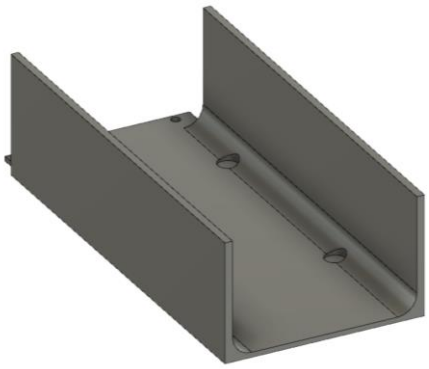


Figura 36. Diseño Caja Electrónica

A la pieza de la Figura 36, se le añaden una serie de piezas para cubrir su parte superior, una de las cuales albergará los dos botones con los cuales se controla la interfaz del microcontrolador (Figura 37), otra simplemente actúa a modo de embellecedor (Figura 38) y la última sujetará el microcontrolador a la "Caja Electrónica" (Figura 39).

Figura 37. Diseño Tapa Botones

Figura 38. Diseño Tapa Superior

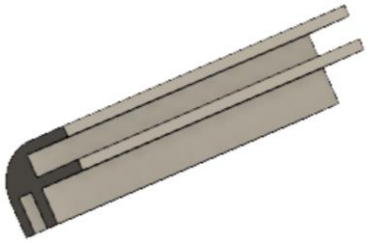


Figura 39. Diseño Laterales Superiores

Una vez ensamblado este conjunto de piezas el resultado es el que se muestra a continuación:

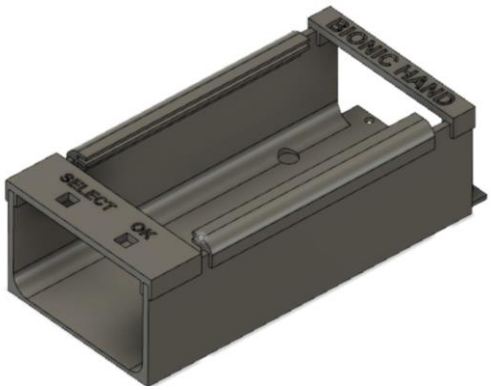


Figura 40. Ensamblado de las piezas que componen Caja Electrónica

Como se observa en la figura anterior, el hueco que se queda en la parte superior será para colocar el microcontrolador, y tener acceso a la pantalla en la cual se visualizará la interfaz del sistema.

4.2.4. Ensamblado del modelo 3D

Empleando también el software *Autodesk Fusion 360* se ha realizado el ensamblado de todas las piezas diseñadas con la finalidad de ver fácilmente errores estructurales antes de proceder a la impresión 3D.

Adicionalmente, se han activado las propiedades físicas en dicho software permitiendo así las colisiones entre las diferentes partes móviles del prototipo. Esto permite comprobar que la pinza alcanza todas las posiciones deseadas y que las ruedas dentadas cumplen su función de transmisión de movimiento.

El resultado del ensamblado se observa en la Figura 41:

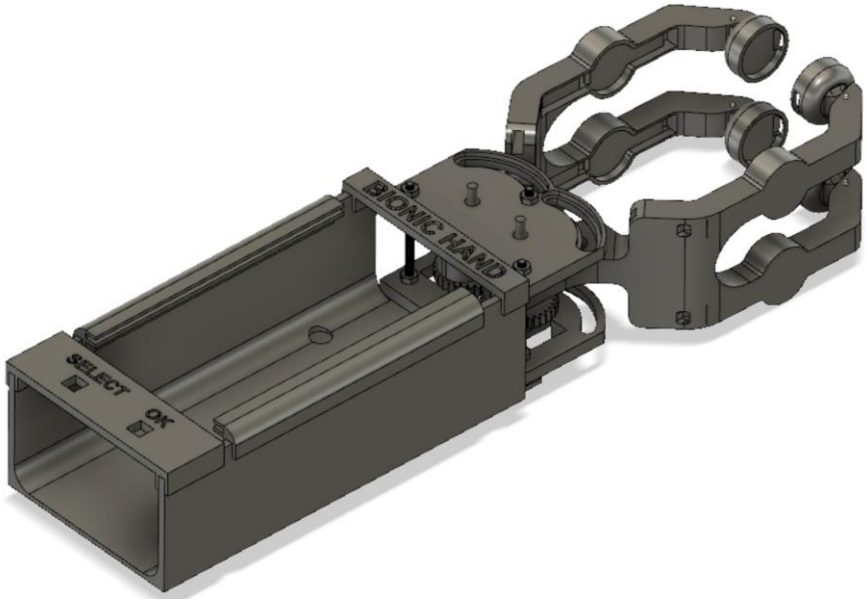


Figura 41. Ensamblado de la pinza completa

4.2.5. Impresión 3D

Para fabricar las piezas del prototipo de la pinza se ha optado por la impresión 3D puesto que con un coste muy bajo se puede crear casi cualquier tipo de objeto en poco tiempo.

4.2.5.1. Impresora 3D a utilizar

En la actualidad, existen una gran cantidad de modelos de impresoras 3D, desde gamas sencillas de uso personal hasta modelos de uso industrial con una alta precisión.

En este caso, puesto que no se requiere una alta precisión, se ha optado por realizar la impresión de las piezas en una impresora 3D de bajo coste. Se trata del modelo *Geeetech Prusa i3 Pro-W* (Figura 42), una impresora que, pese a su bajo coste, imprime con buena calidad todo tipo de materiales y tiene un área de impresión más que suficiente para imprimir todas las piezas del prototipo. Uno de sus principales defectos es que para obtener una buena calidad de impresión se ha de imprimir a una velocidad inferior a otros modelos, por lo que el tiempo de impresión será mayor.

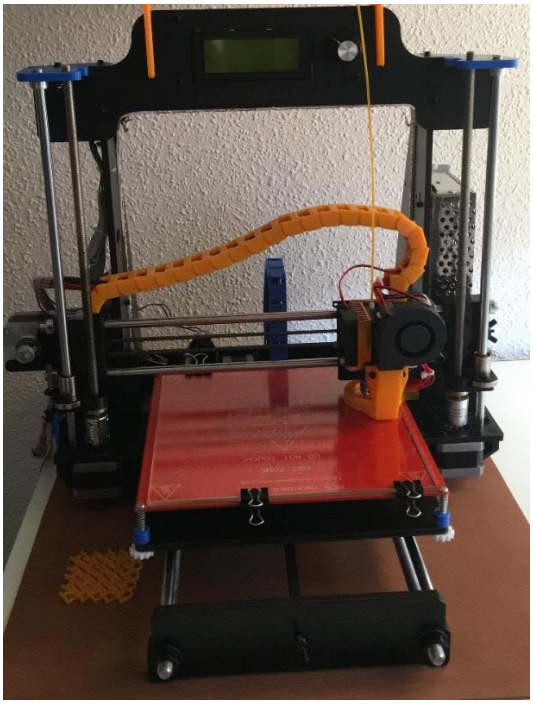


Figura 42. Impresora 3D empleada

4.2.5.2. Filamento empleado

Como se ha comentado en el apartado anterior, este modelo de impresora es capaz de utilizar una gran variedad de filamentos, entre los cuales se destacan PLA, ABS, PETG, Filaflex, etc. En la Tabla 2 se comparan las ventajas e inconvenientes de los diferentes tipos de plásticos que se pueden emplear para realizar la impresión de las piezas 3D:

Tipo de Filamento	Ventajas	Inconvenientes
PLA	- Fácil extrusión - Termoplástico biodegradable - Baja temperatura de impresión - Barato	- No flexible - No resistente a altas temperaturas
ABS	- Larga vida útil - Resistente a altas temperaturas - Alta dureza	- Deformaciones cuando se enfrían - Emisión de humos tóxicos - Difícil extrusión
PETG	- Más flexible que el PLA - Alta durabilidad	- Se raya fácilmente
Filaflex	- Alta flexibilidad - Alta durabilidad	- Difícil extrusión

Tabla 2. Comparativa filamentos impresión 3D

Tras esta comparativa, se ha decidido que, para imprimir las piezas del prototipo, se empleará PLA por ser fácil de extruir, barato y además ecológico. En el caso de que se decidiera imprimir piezas para la construcción de un modelo final de la pinza, se utilizaría ABS o PETG por tener una mayor durabilidad y resistencia al desgaste.

En el apartado 7.2 (Presupuesto) se recogen los tiempos y el precio de todas las impresiones 3D que se han llevado a cabo para fabricar la pinza.

4.2.5.3. Configuración básica para la impresión 3D

Como se ha comentado en el apartado anterior, para imprimir las piezas en 3D se va a utilizar PLA, en concreto, de color blanco.

Para que la impresora pueda imprimir el prototipo, se han de importar los modelos 3D a un programa de impresión 3D, en este caso, se utiliza *Simplify 3D*. En este programa se introducen todos aquellos parámetros necesarios para que la impresión se pueda llevar a cabo, entre los que se pueden destacar: temperatura de impresión (215 grados), altura de capa (0.2 mm), o el porcentaje de relleno del interior de la pieza y la creación de soportes, que se determinan según la forma y las dimensiones de la pieza que se quiera imprimir.

Por ejemplo, para imprimir la pieza “Estructura Dedos”, se utiliza un porcentaje de relleno del 40% (Figura 43. Ajustes de impresión con Simplify 3D) formando una estructura triangular la cual le confiere una gran resistencia a la pieza sin necesidad de imprimirla totalmente maciza, con el consiguiente ahorro de tiempo de impresión. Por otra parte, en este caso se utilizarán soportes (Figura 43) con la finalidad que el filamento al ser extruido no se derrame en las zonas que por su inclinación no tienen material debajo.

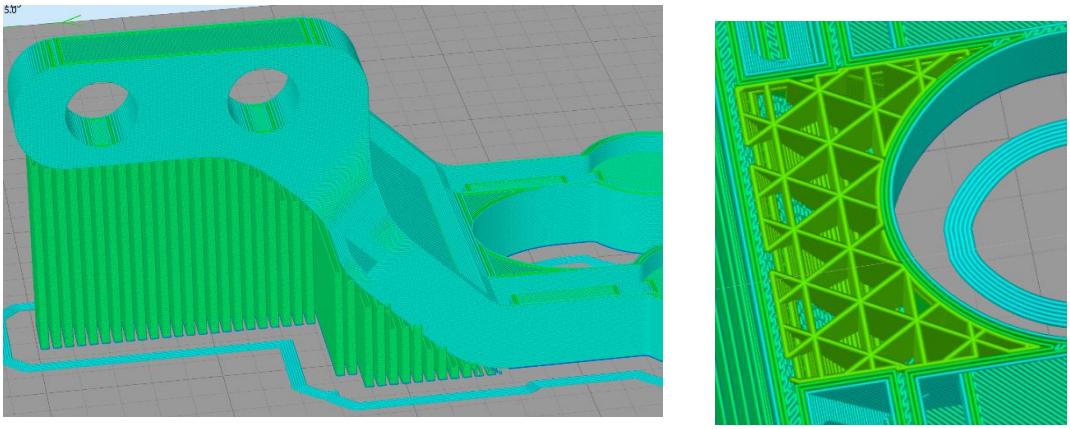


Figura 43. Ajustes de impresión con Simplify 3D

Tras realizar todas las impresiones, el resultado de la pinza impresa y ensamblada es el que se muestra a continuación:

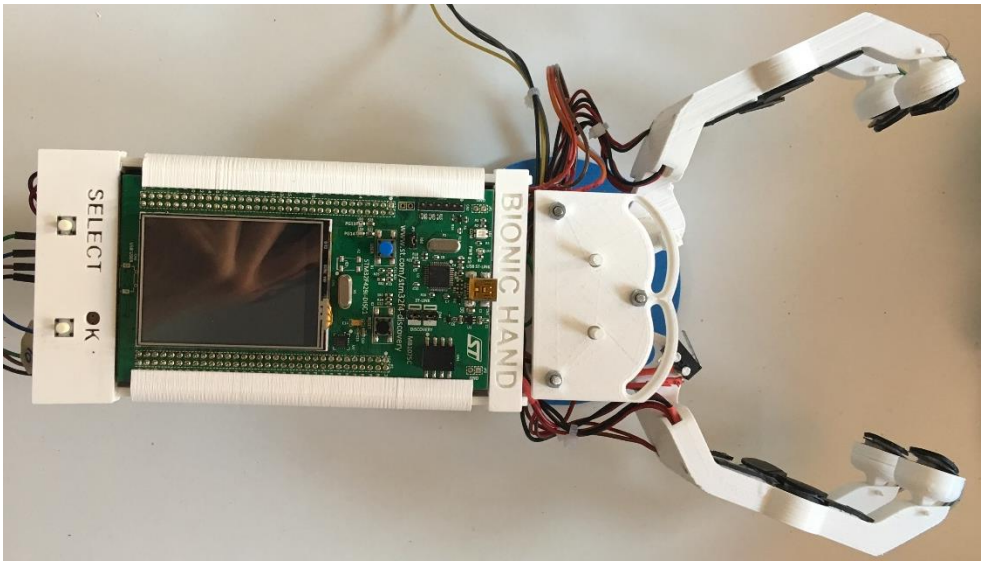


Figura 44. Pinza real completamente ensamblada

4.2.6. Material antideslizante

Con la finalidad que los objetos no resbalen cuando son agarrados por la pinza, se colocan sobre los dedos y alrededores unas láminas de caucho obtenidas de la cámara de una rueda de bicicleta. Estas láminas también ejercen una función de actuar como recubrimiento para los sensores para que estos no entren en contacto directo nunca con los objetos. En la Figura 45 se muestra el resultado tras la colocación de estas láminas.

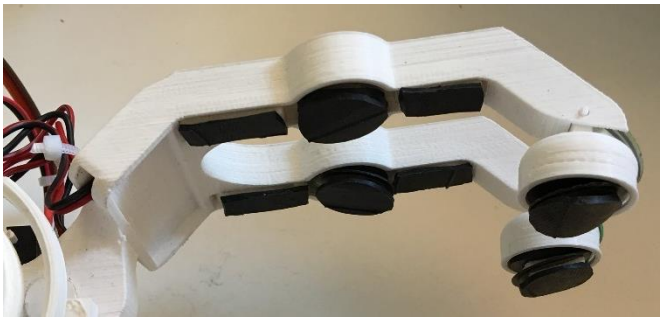


Figura 45. Detalle del material antideslizante

4.3. Desarrollo de la electrónica

La electrónica se desarrolla por separado en los bloques que se describen en los siguientes apartados para finalmente conectarse todo con el microcontrolador, tal y como se describe en el siguiente esquema (Figura 46), donde mediante flechas se indica lo que serán entradas o salidas del microcontrolador:

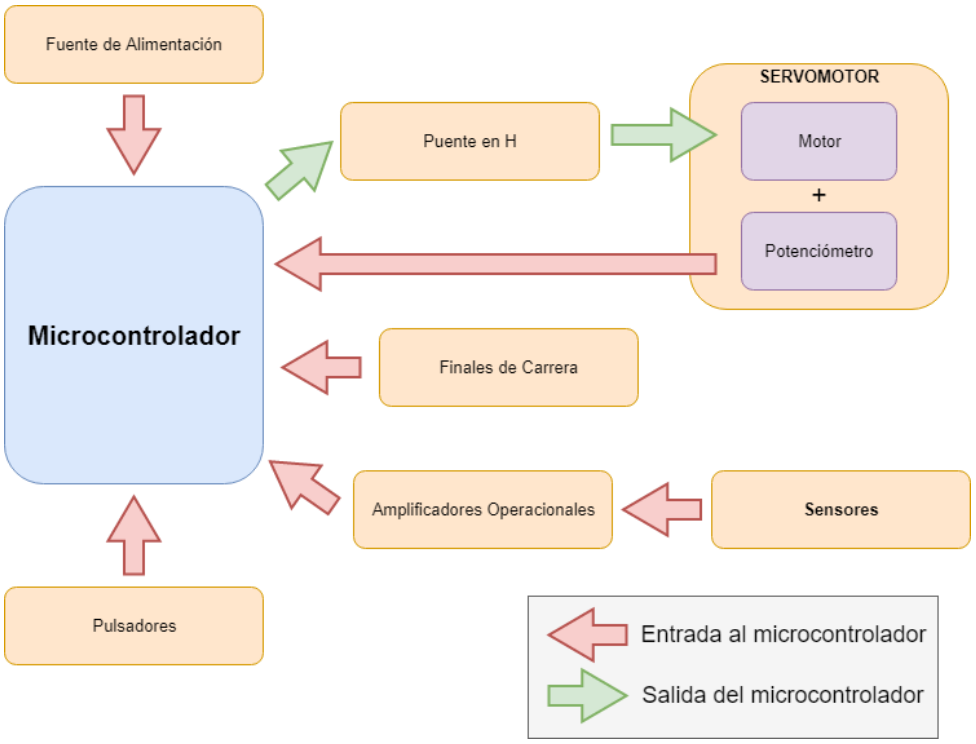


Figura 46. Representación general de la electrónica de la pinza

4.3.1. Servomotor y puente en H

Para los movimientos de apertura y cierre de la pinza, es necesario el uso de un servomotor el cual siga las órdenes recibidas del controlador.

4.3.1.1. Comparativa Servomotores

En la Tabla 3 se muestran las prestaciones de los servomotores más utilizados actualmente:

Modelo	Tensión (V)	Par (Kg·cm)	Rango	Peso (g)	Dimensiones (mm)
SG90	4.8 – 6	2.5	180 °	9	22.2 x 11.8 x 31
MG946R (TowerPro)	4.8 – 6	10.5 – 13	180 °	55	40.7 x 19.7 x 42.9
MG958 (TowerPro)	4.8 – 6	18 - 20	180 °	65	40.2 x 20.1 x 36.8
MG959 (TowerPro)	4.8 – 6	28 - 32	180 °	78	40.2 x 20.1 x 36.8

Tabla 3. Comparativa Servomotores

Como se observa en la tabla anterior, todos los modelos expuestos trabajan en el mismo rango de tensiones y rango de giro, por lo que su elección se basará en el peso y el par que sean capaces de desarrollar. Por ello, entre todas las opciones, se elige el servomotor *MG959*[5] fabricado por *TowerPro* por ser el que mayor par es capaz de desarrollar. Gracias a ello, la pinza podrá soportar los objetos con una mayor fuerza sin llevar el motor a sus límites de funcionamiento. Sus principales defectos son el peso, ya que este es mayor que los otros servomotores; y el precio (54 euros).

4.3.1.2. Eliminación de la electrónica interna

El servomotor por defecto trabaja en bucle cerrado utilizando la realimentación de la posición proveniente de su potenciómetro interno. La salida del potenciómetro es proporcional a la posición del eje del servomotor. Este funcionamiento se describe en el siguiente esquema:

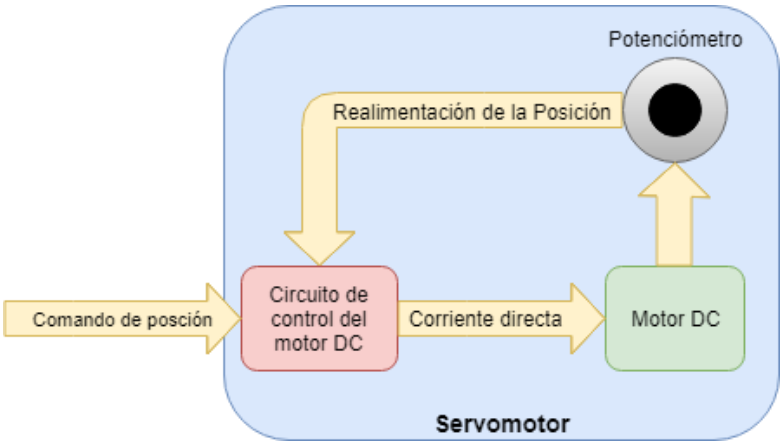


Figura 47. Representación del servomotor por defecto

Si se conecta el servomotor a un microcontrolador, el bucle cerrado seguirá estando dentro del servomotor por lo que desde la tarjeta se podrá determinar una posición a alcanzar, pero no habrá manera de confirmar si esa posición se ha alcanzado. El esquema que lo representa se muestra en la Figura 48:

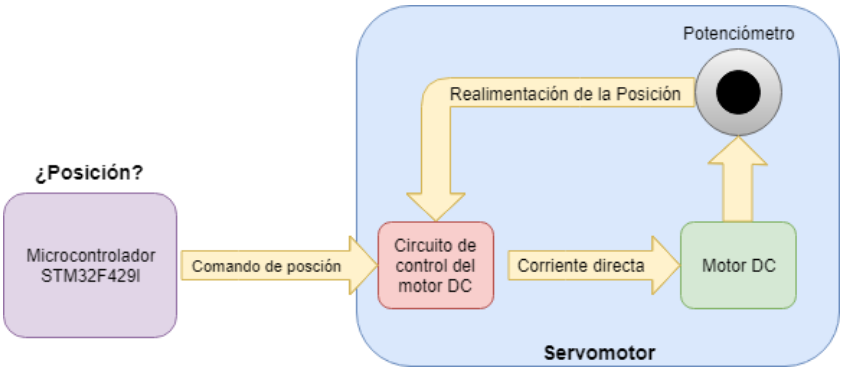


Figura 48. Representación del servomotor por defecto conectado con el microcontrolador

[5] https://www.towerpro.com.tw/product/mg959-2/

Por lo que para que sea el microcontrolador el que obtenga los datos del potenciómetro con los que determinar si se ha alcanzado o no la posición, se procede a eliminar el circuito de control del motor DC para posteriormente implementarlo en dicho microcontrolador como se muestra en la siguiente representación (Figura 49):

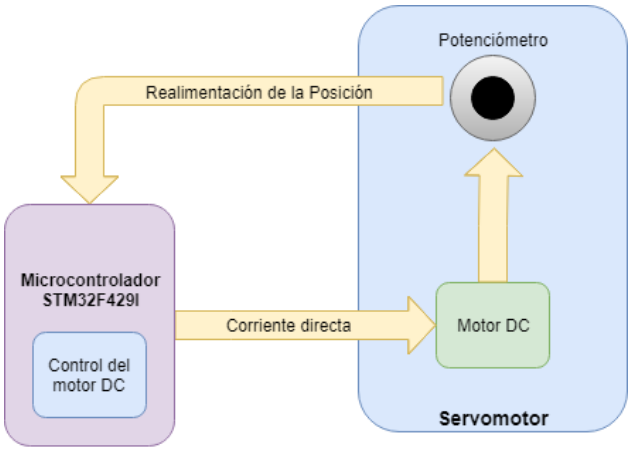


Figura 49. Representación estructura interna servomotor tras eliminar electrónica interna

Eliminación de la electrónica interna del servomotor MG959:

Con el fin de eliminar la placa electrónica del interior del servomotor dejando así únicamente el motor DC, el potenciómetro y el conjunto de ruedas dentadas que forman la transmisión, se siguen los siguientes pasos:

En primer lugar, se extrae la cubierta inferior del servomotor (Figura 50) eliminando los cuatro tornillos que lo sujetan, dejando al descubierto la placa controladora que se pretende eliminar.

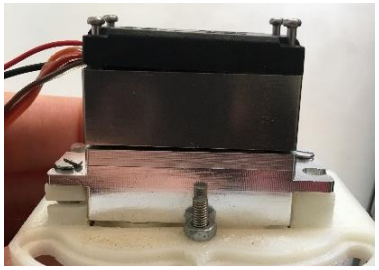


Figura 50. Servomotor antes de la eliminación de su electrónica interna

En segundo lugar, se extrae la placa controladora (Figura 51) y se cortan todos los cables soldados a esta. Con ayuda de un soldador, se desueldan los dos pines el motor.

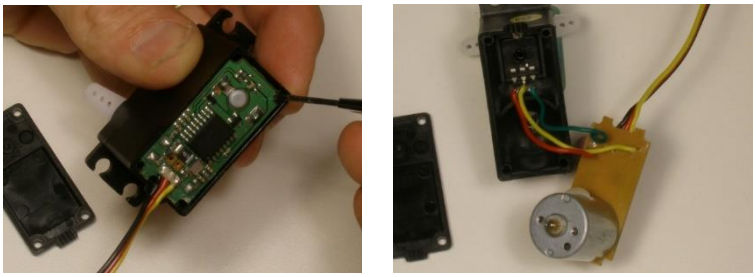


Figura 51. Eliminación de la placa electrónica del servomotor

En tercer lugar, una vez se ha eliminado la placa electrónica, se tendrán cinco pines a la vista, tres de ellos pertenecientes al potenciómetro, y los otros dos, al motor DC. A continuación, se suelda un cable de unos 20 cm a cada uno de estos pines (Figura 52).

Figura 52. Resultado del interior del servomotor tras eliminar la placa electrónica

En último lugar, se cierra el servomotor con su tapa dejando los cinco cables al aire. La tapa del servomotor se fija con los tornillos que se han extraido en el primer paso.

4.3.1.3. Puente en H

Como se ha eliminado la electrónica interna del servomotor, este actualmente no tiene forma de cambiar el sentido de giro, ni tampoco variar su velocidad. Para solucionar esto, se le añade un puente en H, es decir, un integrado *L293B*[6], el cual permitirá variar el sentido de giro y también la velocidad de movimiento a través de una señal PWM.

Las principales características del *L293B (*Figura 53*)* son:

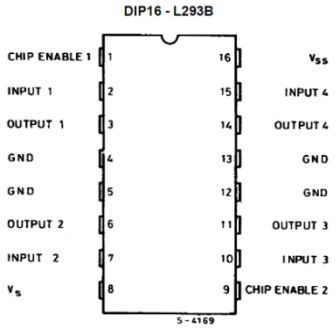


Figura 53. Esquema L293B

- V_s (tensión de alimentación): Min= V_{ss}; Máx= 36 V
- V_{ss} (tensión de alimentación lógica): Min= 4.5 V; Máx= 36 V
- V_i (tensión de entrada): Máx= 7 V
- Posibilidad de controlar 4 motores en una dirección o dos motores en ambas direcciones.

Dadas estas características del puente en H, en este proyecto y para el motor elegido, V_s y V_{ss} se alimentan a 5 V. Además, como únicamente se necesita conectar un servomotor, se utilizará la mitad del puente en H, por lo que en un futuro se podría añadir otro motor que controlaría por ejemplo el movimiento de la muñeca de la pinza

[6] https://www.st.com/resource/en/datasheet/l293b.pdf

4.3.1.4. Esquema eléctrico Servomotor + L293B

El esquema eléctrico que representa el conjunto Servomotor + *L293B* es el que se muestra en la siguiente figura:

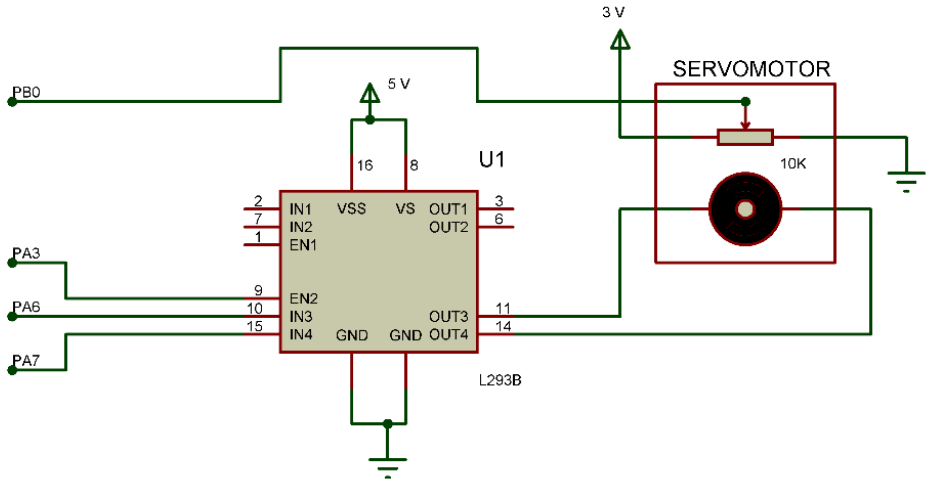


Figura 54. Esquema eléctrico Servomotor + L293B

Como se observa en la Figura 54, las salidas *OUT3* y *OUT4* se conectan a los polos del motor DC, mientras que las entradas *IN3* y *IN4* se conectan a los puertos *PA6* y *PA7* del microcontrolador, la función de los cuales es cambiar el sentido de giro del motor. El puerto *EN2* se conecta junto con *PA3* cuya función será la de recibir una señal *PWM* (Modulación de ancho de pulso) con la que se controlará la velocidad de giro del motor. Por otra parte, las salidas del potenciómetro se conectarán a alimentación positiva, a tierra, y la última salida se conecta con *PB0* que leerá la variación de tensión según el movimiento del potenciómetro.

Para que el motor pueda girar en un sentido u otro, o pararse, las entradas del integrado L293B tienen que estar en un estado alto o bajo según muestra la Tabla 4. Combinaciones de las entradas del puente en H

EN2	IN3	IN4	Función
L	X	X	Motor Parado
H	H	L	Giro Horario
H	L	H	Giro Antihorario
H	H	H	Paro Rápido Motor

Tabla 4. Combinaciones de las entradas del puente en H

L = Low (nivel lógico bajo) - H = High (nivel lógico alto) - X = No importa el estado

4.3.2. Sensores de Fuerza y Amplificadores Operacionales

4.3.2.1. Comparativa sensores de fuerza

Existen multitud de tipos de sensores capaces de detectar una fuerza cuando esta se aplica sobre ellos. A continuación, se muestran algunos de los tipos de sensores que se podrían implementar en una pinza de las características que ocupa este proyecto [6, pp. 298 -300]:

- Sensor Resistivo:

Los sensores de fuerza resistivos (Figura 55) o FSR son unos dispositivos que experimentan un decremento de su resistencia interna con un incremento de la fuerza aplicada sobre su superficie de detección.

En cuanto a sus ventajas, se puede destacar que el precio de estos es reducido en relación con los otros tipos de sensores, funcionan con tensiones de alimentación pequeñas y están disponibles en multitud de tamaños. Una de sus principales desventajas reside no están diseñados para detectar presiones con precisión, sino que está presión tendrá un rango de variación.

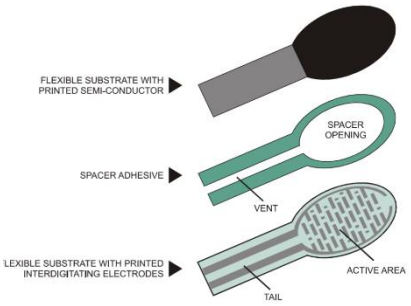


Figura 55. Sensores de Fuerza Resistivos

- Sensor Capacitivo:

Los sensores de fuerza capacitivos (Figura 56) basan su funcionamiento en el mismo principio que los resistivos, pero en este caso, cuando se aplica una fuerza sobre el sensor, se produce una variación de la capacitancia en vez de la resistencia como en el caso anterior.

En cuanto a sus ventajas, este tipo de sensores son de tamaño reducido y requieren unas tensiones de alimentación pequeñas. Su principal ventaja es el precio, puesto que se trata de un tipo de sensor menos común que el resistivo, y por tanto de mayor coste.

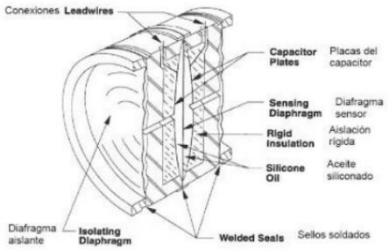


Figura 56. Sensores de Fuerza Capacitivos

- Célula de carga:

Una célula de carga (Figura 57) consiste en un transductor que convierte en una señal eléctrica medible la fuerza que se ha aplicado sobre la célula.

En cuanto a sus ventajas, destacan su alta presión y su bajo error. Sus desventajas se basan en su tamaño y su precio.

Figura 57. Célula de Carga

4.3.2.2. Amplificador operacional empleado

Tras el estudio de los diferentes tipos de sensores del apartado anterior, se ha decidido que se van a emplear sensores de fuerza resistivos, puesto que como se persigue la construcción de una pinza robótica de bajo coste, son los que más se adaptan por su precio reducido. Otra de las razones que justifican su elección se basa en su reducido tamaño en concreto su grosor 0.55 mm.

El modelo de sensor resistivo que se va a utilizar es el *FSR 402*[7] fabricado por *Interlink Electronics* el cual posee un rango de detección de fuerzas de 0.1 a 10 N.

Figura 58. Sensor de fuerza resistivo empleado

Para poder analizar los datos obtenidos por el sensor, será necesario un circuito cuya función será la conversión de fuerza a tensión. Para ello, se utiliza un divisor de tensión junto con un amplificador operacional como el que se muestra en la Figura 59.

La resistencia R_m se puede elegir para maximizar la sensibilidad de la fuerza deseada. En este caso, para una tensión de alimentación V^+ de 5 V, y con un rango deseado de la tensión de salida V_{out} entre 0 y 3 V (tensión máxima admitida a la entrada de los puertos de la tarjeta *STM32F429I*), se elige una R_m de valor 2 KΩ tras la realización de varias pruebas.

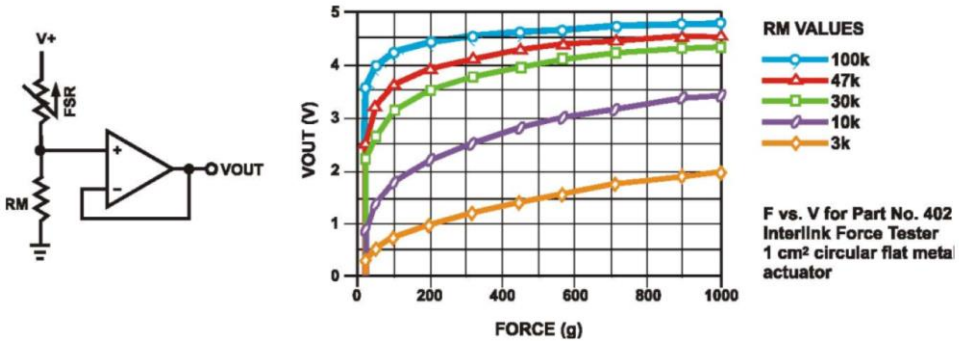


Figura 59. Comportamiento del sensor en función de RM

[7] https://www.sparkfun.com/datasheets/Sensors/Pressure/fsrguide.pdf

4.3.2.3. Esquema eléctrico sensor + LM358P

El esquema eléctrico que representa el conjunto sensor y circuito de amplificación es el que se muestra en la Figura 60 en el que una de las patillas del sensor está alimentado a 5 V y la otra se conecta junto a la resistencia de 2 KΩ a la entrada positiva del operacional.

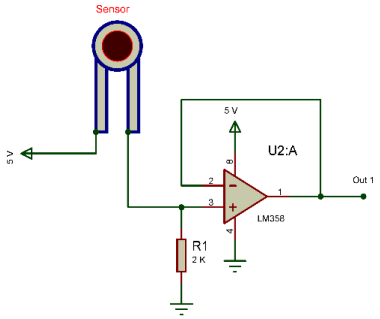


Figura 60. Esquema eléctrico Sensor + LM358P

El circuito anterior representa únicamente el esquema de conexiones de uno de los sensores, pero este proyecto requiere ocho sensores que deberán estar conectados de la misma manera, para ello el *Departamento de Automática y Sistemas* de la *UPV* ha cedido una placa PCB que dispone de cuatro *LM358P*[8] en los cuales se pueden conectar hasta 8 sensores. Esta placa PCB lleva también montado el circuito de la figura anterior con sus respectivas resistencias de 2 KΩ (1 KΩ + 1 KΩ). La siguiente imagen (Figura 61), muestra la PCB, mientras que la Tabla 5. Correspondencia de las entradas de la PCB muestra las conexiones que se deben realizar en cada entrada/salida. A estas entradas se conectan los ocho sensores, mientras que las salidas se conectan con el microcontrolador *STM32F429I.*

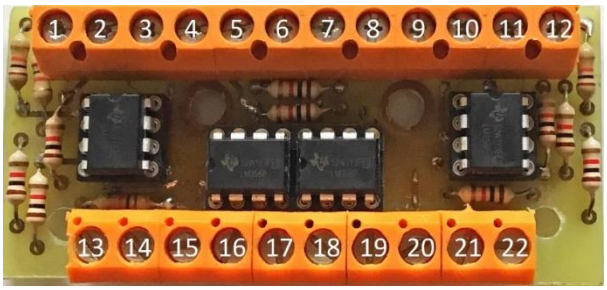


Figura 61. PCB amplificadores operacionales de los sensores

1 – GND	12 – Entrada Sensor 8
2 – Alimentación 5 V	13 – GND
3 – Entrada Sensor 1	14 – Alimentación 5 V
4 – Entrada Sensor 2	15 – Salida Sensor 1 – PB1
5 – Entrada Sensor 3	16 – Salida Sensor 2 – PC0
6 – Entrada sensor 4	17 – Salida Sensor 3 – PC1
7 – Alimentación 5 V	18 – Salida Sensor 4 – PC3
8 – Alimentación 5 V	19 – Salida Sensor 5 – PF3
9 – Entrada Sensor 5	20 – Salida Sensor 6 – PF4
10 – Entrada Sensor 6	21 – Salida Sensor 7 – PF5
11 – Entrada Sensor 7	22 - Salida Sensor 8 – PF6

Tabla 5. Correspondencia de las entradas de la PCB

[8] http://www.ti.com/lit/ds/symlink/lm358.pdf

4.3.2.4. Calibración del sensor para la conversión a fuerza de la tensión leída

El sensor FSR 402 ha sido sometido a una serie de pruebas con diferentes pesos para poder comprender su funcionamiento ante las variaciones de fuerza que se ejerzan sobre él. Para ello, se ha colocado el sensor en una superficie plana, tal y como se muestra en la Figura 63.

Figura 62. Diferentes pesos empleados en la calibración del sensor

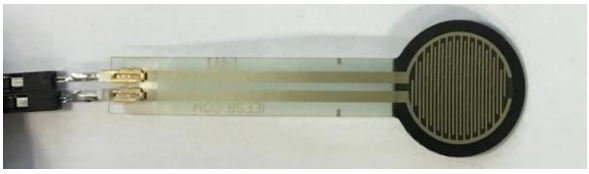


Figura 63. Sensor por calibrar sin recubrimiento

La primera prueba que se realiza sobre el sensor consiste en colocarle diferentes pesos (como los que se muestran en la Figura 62) entre 50 g y 500 g sin ningún tipo de recubrimiento entre el peso y el sensor, lo que ha provocado que no se haya podido tomar una lectura de datos correcta porque el peso no se puede aplicar correctamente al no haber recubrimiento.

A continuación, se coloca sobre el sensor una lámina de caucho de 1.5 mm obtenida de la cámara de una rueda de una bicicleta. Cabe destacar que se ha utilizado caucho para colocar encima del sensor debido a su elevado coeficiente de fricción ya que se mejorará el agarre de los objetos que se quieran sujetar.

Con este recubrimiento, se han hecho las mismas pruebas colocando diferentes pesos sobre el sensor y la lectura de datos en este caso ha sido correcta, aunque dependiendo de cómo se ejerza dicha fuerza, los sensores detectan fuerzas diferentes para un mismo peso. Para intentar aumentar la sensibilidad de dicho sensor se coloca una lámina de caucho del diámetro de la superficie de detección del sensor. Esta mejora ha hecho que con el mismo peso actuando sobre el sensor la tensión a la salida del operacional es mayor que en el caso anterior, aunque los sensores presentan un ruido bastante elevado.

En la Tabla 6. Datos obtenidos de la calibración de los sensores se muestran algunos de los datos que se han obtenido tras realizar las pruebas con diferentes pesos sobre el sensor:

Peso	Lectura 1	Lectura 2	Valor medio Lecturas
0	0	0	0
100	0,7	0,73	0,715
200	1,47	1,5	1,485
300	1,87	1,94	1,905
400	2,38	2,4	2,39
500	2,63	2,69	2,66
600	2,9	2,9	2,9

Tabla 6. Datos obtenidos de la calibración de los sensores

Con los datos obtenidos en la tabla anterior, se hace una representación gráfica de ellos (Figura 64) para poder modelizar una recta con la que obtener su ecuación. Esta ecuación indica la variación de tensión según la fuerza que se ejerza sobre el sensor por lo que se introducirá en la programación con la finalidad de estimar la fuerza que se está ejerciendo en cada momento.

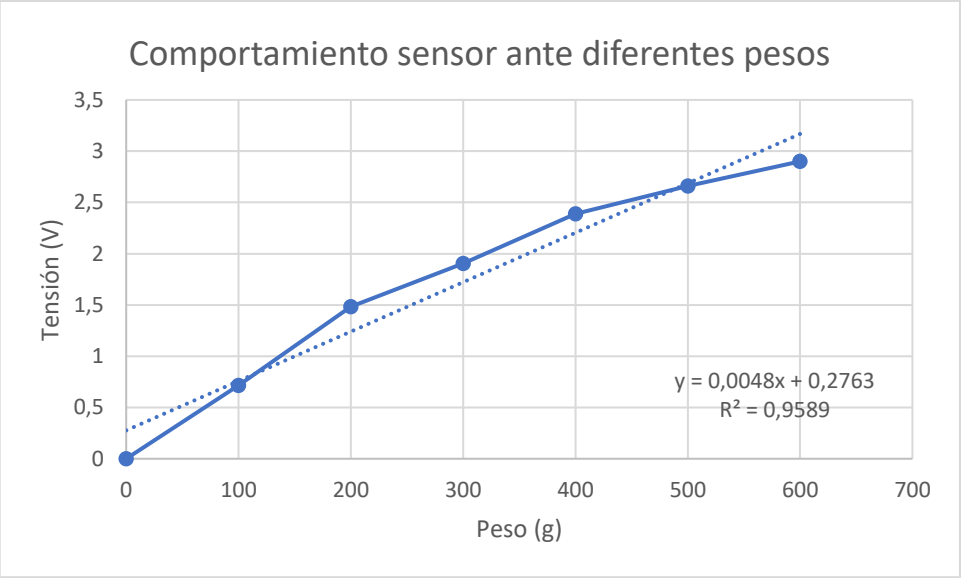


Figura 64. Comportamiento del sensor ante diferentes pesos

Tras realizar estas pruebas, se comprueba que el funcionamiento de este tipo de sensor no es muy bueno debido a los siguientes factores: no se obtienen los mismos datos si se ejerce una fuerza sobre el extremo que si se ejerce sobre su centro o sobre toda su superficie, y también, los datos obtenidos presentan una elevada variabilidad por lo que se estima que la ecuación para obtener la fuerza ejercida en función del peso tendrá un alto margen de error y será poco precisa.

4.3.3. Finales de Carrera

Con la finalidad de proteger el motor y el circuito ante posibles fallos, se ubican dos finales de carrera en las posiciones extremas de la pinza. Uno de ellos se activará si la pinza llega a su máximo cierre, y el otro se activará si esta llega a su máxima apertura tal como se muestra en la Figura 65.

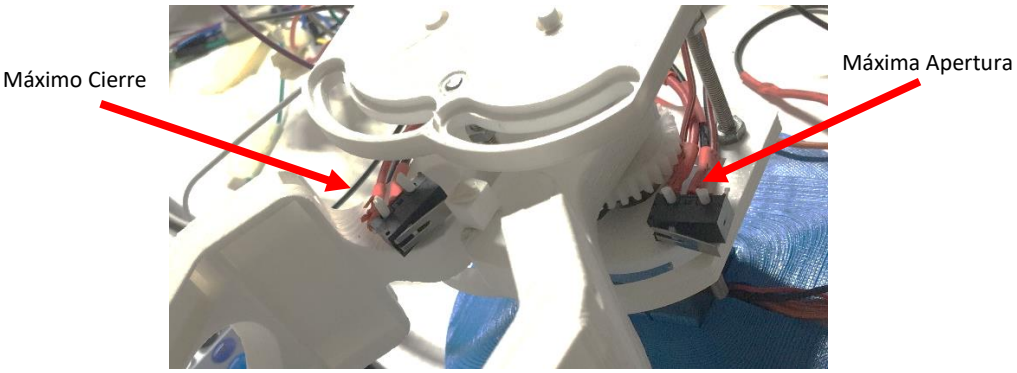


Figura 65. Ubicación de los finales de carrera

El esquema eléctrico de los finales de carrera se muestra a continuación:

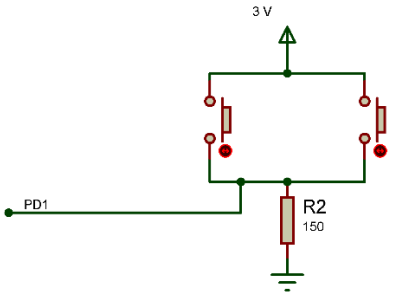


Figura 66. Esquema eléctrico de los finales de carrera

4.3.4. Fuente de Alimentación

La elección de una batería en una pinza robótica es muy importante, puesto que esta deberá tener un tamaño y peso reducido, otorgar suficiente carga para todo el día y ser segura puesto que va a ser llevada en el día a día de una persona.

Por estos motivos, se ha optado por utilizar una fuente de alimentación fija para fabricar el prototipo en vez de una batería que se implementaría en el modelo final. La fuente de alimentación que se emplea es una fuente de PC vieja de la cual se han utilizado las salidas de 5 y 3 V para alimentar los circuitos implementados en los apartados anteriores.

4.3.5. Pulsadores para la Interfaz

Con la finalidad de poder interactuar con la interfaz que se muestra en la pantalla de la tarjeta *STM32F429I* se emplean dos pulsadores que sirven para desplazarse entre menús y seleccionar las diferentes opciones. Estos pulsadores se han conectado a las entradas *PB10* y *PB12* del microcontrolador. El circuito electrónico que permite el funcionamiento de los pulsadores es el siguiente:

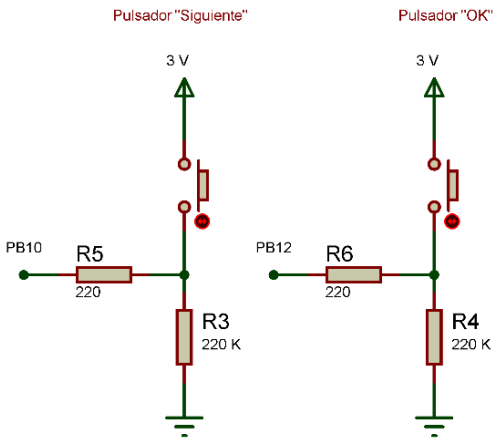


Figura 67. Esquema eléctrico pulsadores

4.3.6. Esquema eléctrico de todo el conjunto

Si se unen todos los circuitos y componentes electrónicos de los apartados anteriores, el circuito completo es el que se muestra a continuación:

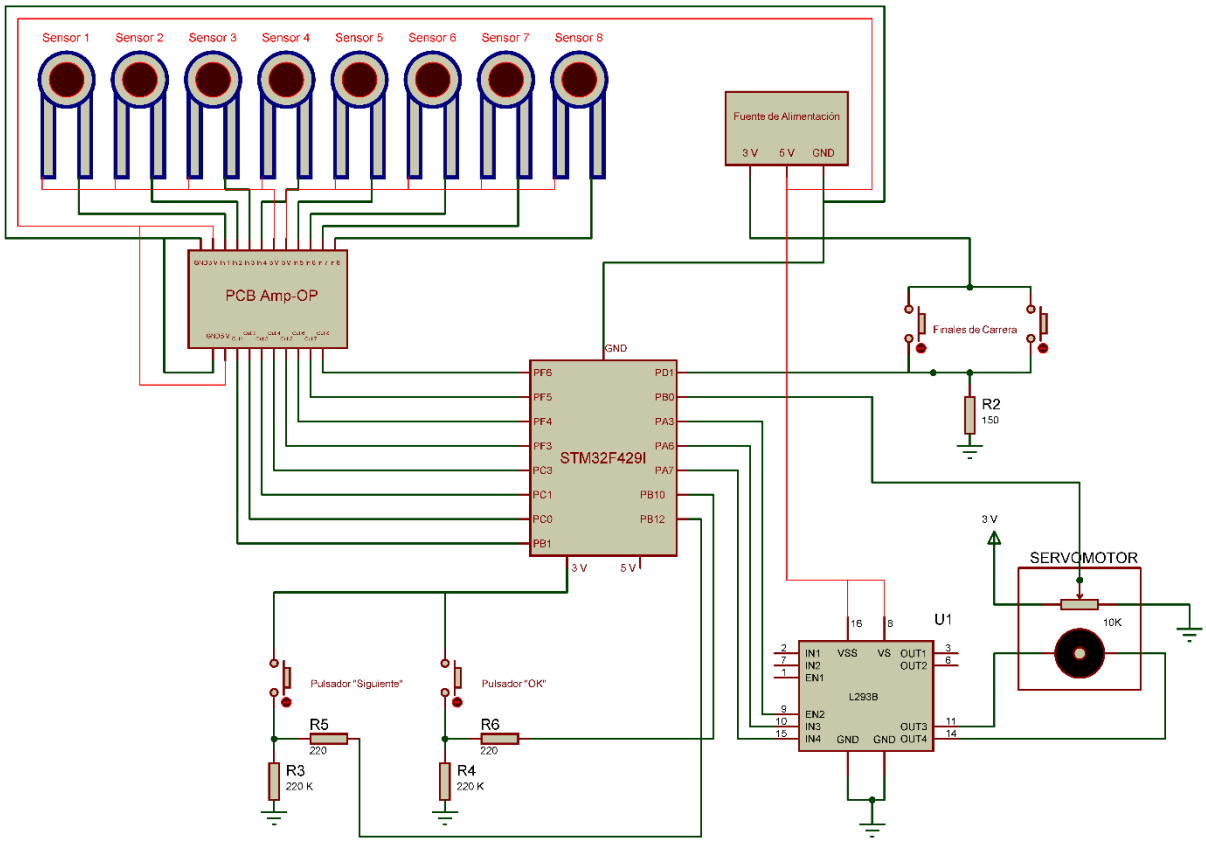


Figura 68. Esquema eléctrico del circuito completo de la pinza

La Figura 68 muestra una representación visual de como se ha realizado el montaje completo de toda la circuitería de la pinza, mientras que la Figura 69, muestra cómo se han soldado/montado todos los componentes sobre la pinza:

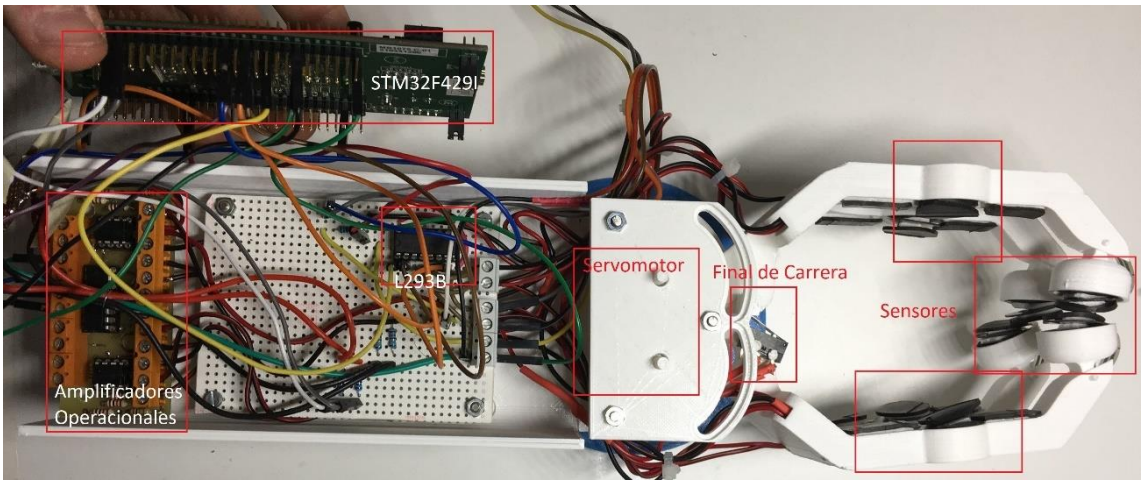


Figura 69. Circuito electrónico real montado sobre la pinza

En la Figura 69 se resalta la ubicación de los componentes más importantes mientras que en la Figura 70 se muestra el resultado final del montaje de la electrónica en el cual se deja a la vista la pantalla del microcontrolador y los botones de control de dicha pantalla, mientras que el resto de los componentes, quedan ocultos tras la tarjeta.

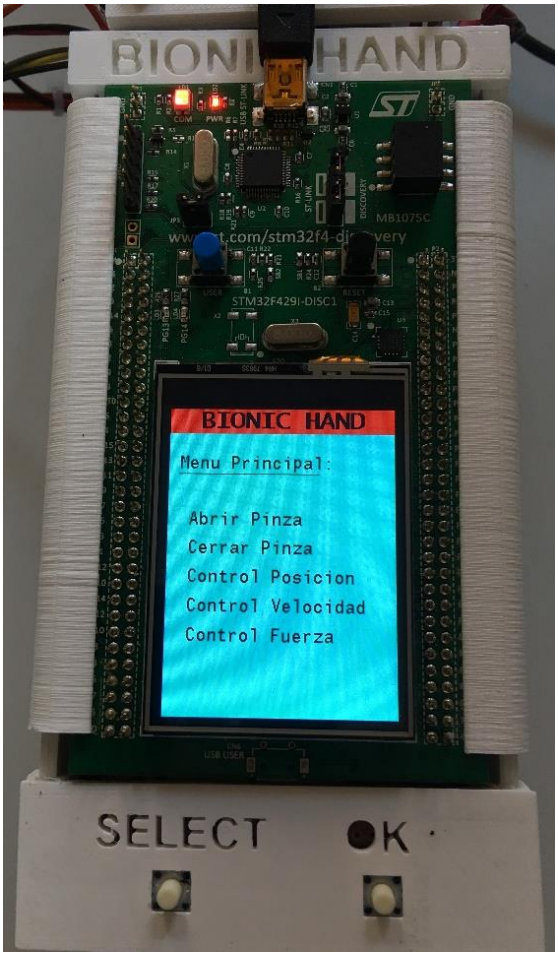


Figura 70. Electrónica visible tras el ensamblado completo de la pinza

4.4. Programación

4.4.1. Lenguaje de programación y software empleado

El lenguaje de programación a emplear para implementar todas las funciones sobre el microcontrolador es lenguaje *C*, debido a que este es relativamente sencillo de utilizar y aprender, ofreciendo una gran flexibilidad en la programación. Otra de las ventajas es que es el lenguaje utilizado por la mayor parte de los microcontroladores del mercado.

Por otra parte, el programa empleado para realizar la programación ha sido *Keil uVision 5* por ser un entorno preparado para la programación de microcontroladores

4.4.2. Microcontrolador. STM32F429I

4.4.2.1. Comparativa de los diferentes tipos de microcontroladores

Para la realización de este proyecto es necesario el uso de un microcontrolador que contenga toda la programación para el funcionamiento del sistema, por lo que en la siguiente tabla se realiza una comparativa de los microcontroladores más utilizados:

Microcontrolador	Ventajas	Inconvenientes
Arduino	- Bajo Coste - Fácil Programación - Código Abierto - Lenguaje #C	- Menor Potencia - Menos puertos disponibles
STMicroelectronics	- Elevada potencia - Lenguaje #C - Gran número de puertos disponibles - Gran número de funcionalidades	- Mayor complejidad de programación
Raspberry Pi	- Gran rendimiento - Dispone de puertos USB y microSD - Bajo consumo - Posibilidad de elegir el lenguaje de programación	- Según aplicación es posible que requiera un sistema de refrigeración

Tabla 7. Comparativa microcontroladores

4.4.2.2. Justificación del microcontrolador elegido

Tras la comparativa realizada en el apartado anterior, se decide que se va a emplear el microcontrolador *STM32F429I* debido a las siguientes razones:

- **Velocidad del reloj:** la velocidad de esta placa es unas 10 veces superior a la de un modelo común de *Arduino* por lo que en los controles se podrán tener tiempos de muestreo de 1 ms.
- **Número de pines:** esta placa dispone de más de 160 pines de entrada/salida con multitud de funciones: *Timers*, interrupciones*, ADC's, DAC's*, etc.
- **Interrupciones:** permiten el paro momentáneo del programa en ejecución si una de ellas se activa, por lo que puede ser muy útil como sistema de seguridad, como, por ejemplo, pueden ser implementadas cuando se acciona un final de carrera para parar un motor que ha alcanzado su posición extrema.

4.4.3. Pines empleados y funcionalidad

La tarjeta *STM32F429I* dispone de gran cantidad de pines útiles para ser programados según se desee. En referencia a este proyecto, se han programado los siguientes pines cuya funcionalidad se muestra a continuación:

- **PA3:** Salida PWM (*TIM2_CH4*) conectada al *ENABLE2* del *L293D.*
- **PA6:** Salida Digital conectada al *INPUT3* del *L293D.*
- **PA7:** Salida Digital conectada al *INPUT4* del *L293D.*
- **PBO:** Entrada Analógica (*A.DC2*) conectada a sensor de fuerza.
- **PB1:** Entrada Analógica (*ADC2*) conectada a sensor de fuerza.
- **PB10:** Entrada Digital conectada al circuito de *Pulsador Siguiente.*
- **PB12:** Entrada Digital conectada al circuito de *Pulsador OK.*
- **PCO:** Entrada Analógica (*ADC1*) conectada a sensor de fuerza.
- **PC1:** Entrada Analógica (*ADC1*) conectada a sensor de fuerza.
- **PC3:** Entrada Analógica (*ADC1*) conectada a sensor de fuerza.
- **PD1:** Entrada Digital (*Interrupción EXTI1_IRQn*) conectada a los finales de carrera.
- **PF3:** Entrada Analógica (ADC3) conectada a sensor de fuerza.
- **PF4:** Entrada Analógica (ADC3) conectada a sensor de fuerza.
- **PF5:** Entrada Analógica (ADC3) conectada a sensor de fuerza.
- **PF6:** Entrada Analógica (ADC3) conectada a sensor de fuerza.

Para usar la pantalla LCD integrada en el microcontrolador se utiliza la librería *tm_stm32f4_ili9341* creada por Tilen Majerle [7] la cual para hacer funcionar la pantalla necesita que no se utilicen los siguientes pines:

- **PC2:** Pantalla LCD (*chip select for SPI).*
- **PD12:** Pantalla LCD (*Reset LCD).*
- **PD13:** Pantalla LCD (*Data/Command Register*).
- **PF7:** Pantalla LCD (*SPI Clock).*
- **PF8:** Pantalla LCD (*Output from LCD to SPI).*
- **PF9:** Pantalla LCD *(SPI Master Output).*

Finalmente, la tarjeta dispone de dos botones, uno azul (se puede programar), uno negro (botón *Reset*) y dos leds:

- **PA0:** Botón Azul (*Interrupción EXTI0_IRQn*) utilizado para hacer pruebas de funcionamiento de la pinza.
- **PG13:** Led Verde el cual se ilumina cuando se ejecutan correctamente los diferentes controles programados.
- **PG14:** Led Rojo el cual se ilumina cuando se ejecutan correctamente los diferentes controles programados.

4.4.4. Programación básica: *ADC, Timers, Interrupciones y PWM*

En primer lugar, antes de empezar con la programación, se ha de soldar el puente *SB9* de la tarjeta STM32F429I para poder así imprimir datos mediante *printf ()* en la pantalla de *debug* de *Keil uVision 5* [8].

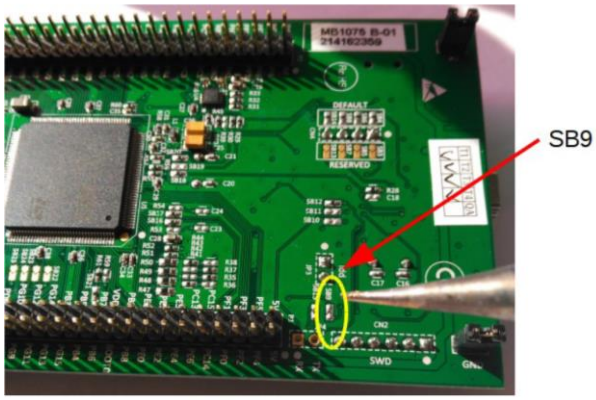


Figura 71. Detalle del puerto del microcontrolador a soldar

A continuación, para no tener que crear un proyecto desde cero en *Keil uVision 5,* se ha recurrido a los proyectos y librerías creados por Tilen Majerle [7], en concreto un proyecto que dispone de una librería que preconfigura la pantalla LCD del microcontrolador.

El siguiente paso consiste en la programación de las diferentes herramientas que permitirán el tratamiento de las señales recibidas o a enviar, como pueden ser *ADC's, Timers, Interrupciones, Delays, PWM,* etc.

4.4.4.1. Programación de los ADC

Un *ADC* consiste en la conversión de una señal analógica en una señal digital, es decir, en código binario. En este caso, los conversores *ADC* utilizados tienen una resolución de 12 bits, es decir, 2^{12} valores distintos (entre 0 y 4095).

Para este proyecto, son necesarios nueve convertidores *ADC*, ocho para los sensores de fuerza y uno para la lectura del potenciómetro ubicado en el servomotor. Estas nueve entradas analógicas se reparten para su conversión entre los tres *ADC (ADC1, ADC2, ADC3)* que dispone la tarjeta *STM32F429I.*

Para ello, se creado el fichero *InicializacionEntradasAnalogicas.c* donde se declaran todas aquellas funciones que permiten el funcionamiento de los ADC. En el Anexo I: Código programación microcontrolador, se encuentra el código de todos los ficheros que se han programado en este proyecto.

En primer lugar, se ha creado la función "*analog_inicializar(void)"* en la cual se declaran entradas analógicas los nueve pines, y se inician los tres *ADC* que se van a emplear. En este caso, el *ADC1* manejará los puertos *PC0, PC1 y PC3;* el *ADC2* manejará los puertos *PB0 y PB1;* y el *ADC3* manejará los puertos *PF3, PF4, PF5 y PF6.*

Una vez inicializadas todas las entradas analógicas con sus respectivos *ADC*, se han creado una serie de funciones cuya misión es realizar la conversión de bits a voltios o grados según la entrada analógica:

- ***int leer_valor_PXX (void)*** (siendo PXX cualquiera de las entradas analógicas declaradas que se muestran en la Figura 72): esta función se encarga de leer el valor en bits de una de las entradas analógicas. A su salida devuelve dicho valor.

```
//ENTRADAS ANALÓGICAS CORRESPONDIENTES A ADC1
int32_t leer_valor_PC0(void){      //Sensor Fuerza
int32_t leer_valor_PC1(void){      //Sensor Fuerza
int32_t leer_valor_PC3(void){      //Sensor Fuerza
//ENTRADAS ANALÓGICAS CORRESPONDIENTES A ADC3
int32_t leer_valor_PF3(void){      //Sensor Fuerza
int32_t leer_valor_PF4(void){      //Sensor Fuerza
int32_t leer_valor_PF5(void){      //Sensor Fuerza
int32_t leer_valor_PF6(void){      //Sensor Fuerza
//ENTRADA ANALÓGICA CORRESPONDIENTE A ADC2
int32_t leer_valor_PB0(void){      //POTENCIÓMETRO MOTOR
int32_t leer_valor_PB1(void){      //Sensor Fuerza
```

Figura 72. Funciones de lectura del valor de los diferentes sensores

- ***float leer_valor_PB0_grados (void):*** esta función se encarga de convertir el valor en bits tomado por *leer_valor_PB0* a grados. Esta conversión se realiza mediante la siguiente expresión (1)

$$valorPB0_{grados} = \frac{leervalor_{PB0} - 330}{-1.7857} \qquad (1)$$

Esta ecuación se obtiene realizando la siguiente experiencia con el servomotor:

En primer lugar, se traza sobre un papel una serie de líneas cada 10 grados tal y como se muestra en la Figura 73. A continuación, se coloca el extremo de la pinza sobre los 0 grados y se mide su valor en bits. Después, se coloca el extremo de la pinza sobre la posición de 70 grados y se vuelve a medir la posición en bits.

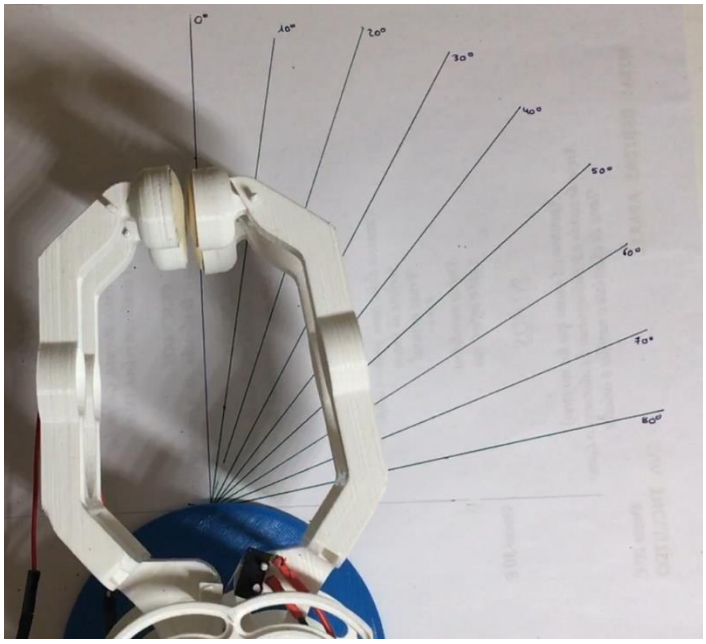


Figura 73. Hoja con medidas de grados

En segundo lugar, se representan en una gráfica (Figura 74) los datos anteriores y se calcula su pendiente (2):

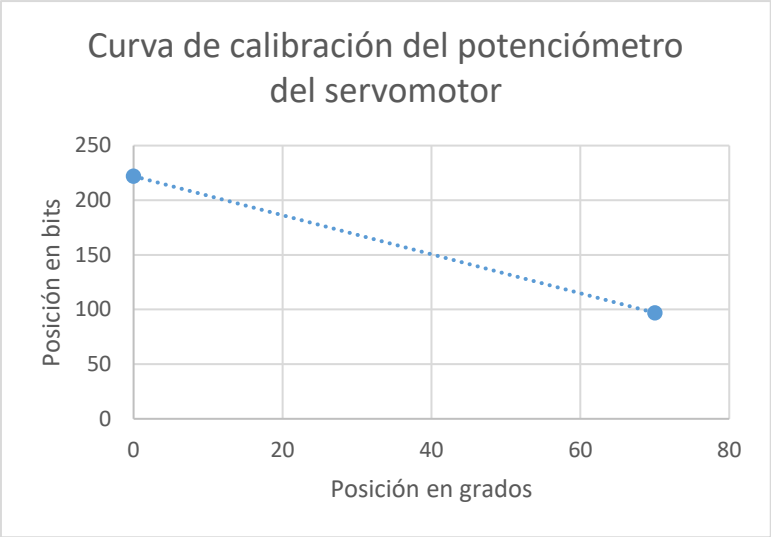


Figura 74. Curva de calibración del potenciómetro del servomotor

$$pendiente = \frac{posbits_{final} - posbits_{inicial}}{posgrados_{final} - posgrados_{inicial}} = \frac{97 - 222}{70 - 0} \quad (2)$$

$$pendiente = -1.7857\ bits/grado \quad (3)$$

Por tanto, el valor de la pendiente obtenida en la anterior ecuación se coloca en el denominador de la siguiente ecuación (4) donde el valor de -330 bits pertenece a la posición de 0 grados, es decir, la pinza completamente cerrada.

$$valorPB0_{grados} = \frac{leervalor_{PB0} - 330}{-1.7857} \quad (4)$$

- ***float leer_valor_PXX_voltios (void):*** (siendo PXX cualquiera de las entradas analógicas declaradas que se muestran en la Figura 75) esta función se encarga de convertir el valor en bits tomado por *leer_valor_PXX* a voltios. Esta conversión se realiza mediante la expresión que se muestra en la ecuación (5).

```
//Conversiones a voltios de los sensores de presión
float leer valor PC0 voltios(void){
float leer valor PC1 voltios(void){
float leer valor PC3 voltios(void){
float leer valor PF3 voltios(void){
float leer valor PF4 voltios(void){
float leer valor PF5 voltios(void){
float leer valor PF6 voltios(void){
float leer valor PB1 voltios(void){
```

Figura 75. Funciones conversoras a voltios

$$valorPXX_{grados} = \frac{leervalor_{PXX} \cdot 2.9}{4096} \quad (5)$$

La ecuación anterior se ha obtenido realizando una regla de 3, en la que se tienen como datos que 2.9 V es la tensión máxima que puede recibir la entrada analógica y la resolución del ADC es de 12 bits (4096 valores distintos). Por tanto:

$$2.9\ V \rightarrow 4096 \quad (6)$$

$$valorPXX_{grados} \rightarrow\ leervalor_{PXX} \quad (7)$$

Con esta simple regla de tres, se obtiene la ecuación (8) que transforma a grados el valor leído por el ADC.

$$valorPXX_{grados} = \frac{leervalor_{PXX} \cdot 2.9}{4096} \quad (8)$$

4.4.4.2. Programación del PWM y del motor

Para que el motor DC que se encuentra dentro del servomotor pueda girar en ambas direcciones y también variar su velocidad, se han creado dos ficheros: el primero *InicializacionMotor.c* que se encarga del control del sentido de giro; y el segundo *InicializacionPWM.c* cuya función es variar la velocidad de giro del motor.

InicializacionMotor.c

Este fichero está compuesto por 4 funciones, el funcionamiento de las cuales se describe a continuación:

- ***void InicializacionMotor (void)***: esta función se encarga de la inicialización de los pines PA6 y PA7 como salidas digitales.

- ***void MotorCierrePinza (void)***: cuando se llama a esta función, el pin PA6 pasa a estado alto, mientras que PA7 pasa a estado bajo, provocando así el movimiento del motor en el sentido de cierre de la pinza.

- ***void MotorAperturaPinza (void)***: cuando se llama a esta función, el pin PA7 pasa a estado alto, mientras que PA6 pasa a estado bajo, provocando así el movimiento del motor en el sentido de apertura de la pinza.

- ***void MotorParado (void)***: cuando se llama a esta función, los pines PA6 y PA7 pasan a estado bajo, provocando así el paro del motor.

InicializacionPWM.c

El *PWM* (Pulse Modulation Width) consiste en la modulación del ancho de pulso que se utiliza para variar el nivel de tensión de un pin mediante una señal cuadrada con un ciclo de trabajo variable. La señal en estado alto tiene un valor de 3 V y en estado bajo un valor de 0 V.

Como se muestra en la Figura 76, a mayor el ciclo de trabajo que se aplique sobre la señal, los pulsos por periodo tendrán un mayor ancho de pulso, y por tanto el valor promedio de la tensión será mayor. Por ejemplo, como se observa en la figura, si se aplica un ciclo de trabajo del 25 % el ancho de los pulsos será menor por lo que la tensión promedio, también será menor, mientras que, si se aplica un ciclo de trabajo del 75 %, el ancho del pulso será notablemente mayor, con lo que el valor promedio de la tensión también será mayor y, por tanto, mayor velocidad.

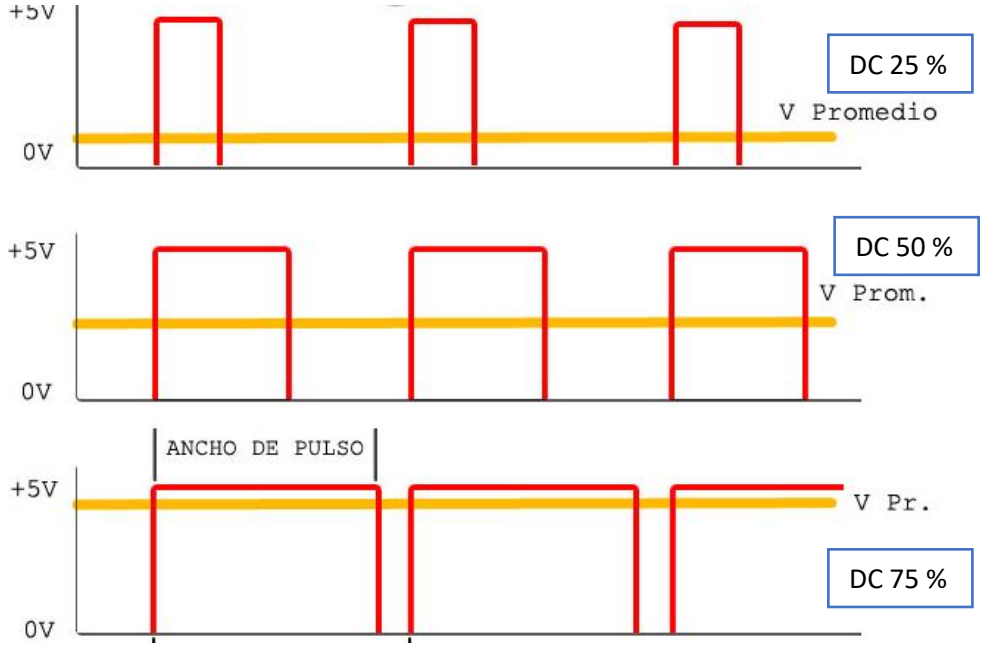


Figura 76. Representación de diferentes ciclos de trabajo

Para conseguir esta modulación del pulso se modifican algunos parámetros de las siguientes funciones creadas por Tilen Majerle:

- ***void InicializacionGPIOA3 (void):*** en esta función, se declara *PA3* como una salida en modo alternativo (*Alternative Function*). También se asigna un reloj a *PA3*.

- ***void TM_TIMER_Init (void):*** en esta función se introduce el valor de la velocidad del reloj interno de la tarjeta y se inicia dicho reloj.

- ***void TM_PWM_Init (uint32_t variablePWM):*** esta función está diseñada para introducirle un valor de *variablePWM* entre 0 (motor parado) y 9000 (velocidad máxima).

Tras realizar esta programación y montar el circuito del motor junto con el puente en H tal y como se describe en el apartado 4.3.1, se realizan pruebas con diferentes valores de *PWM* sobre el motor ya ensamblado en la estructura de la pinza, y se comprueba que el motor necesita un valor mínimo de *PWM* para empezar a moverse, debido a las pérdidas del motor y las fricciones que se generan con las transmisiones mecánicas de la pinza. Este valor mínimo para provocar el movimiento es 2000, es decir, que necesita un 22 % del valor de *PWM* para poder mover la estructura de la pinza.

4.4.4.3. Programación de Interrupciones

Las interrupciones son muy útiles para parar momentáneamente la ejecución del programa si sucede alguna acción de importante relevancia. En el caso de este proyecto, se hace uso de una interrupción que se activa cuando uno de los finales de carrera, colocados en las posiciones extremas de la pinza, es activado.

Para la programación de la interrupción, se crea el fichero *InicializacionInterrupción.c* que se encarga de configurar el pin *PD1* como entrada digital, y también de inicializar la interrupción. Para activar esta interrupción hay que tener en cuenta que el manejador de interrupciones funciona por líneas, es decir, en este caso se activará *EXTI_Line1* la cual está conectada a todos los pines que lleven el número 1 (*PA1, PB1, PC1*, etc.), por lo que si se quisiera añadir otra interrupción, se tendría que activar el manejador de otra línea diferente a la 1 tal y como se muestra en la Tabla 8. Manejador de Interrupciones[9].

Irq	Manejador (*Handler*)	Descripción
EXTI0_IRQn	EXTI0_IRQHandler	Manejador para los pines conectados a la línea 0
EXTI1_IRQn	EXTI1_IRQHandler	Manejador para los pines conectados a la línea 1
EXTI2_IRQn	EXTI2_IRQHandler	Manejador para los pines conectados a la línea 2
EXTI3_IRQn	EXTI3_IRQHandler	Manejador para los pines conectados a la línea 3
EXTI4_IRQn	EXTI4_IRQHandler	Manejador para los pines conectados a la línea 4
EXTI09_5_IRQn	EXTI9_5_IRQHandler	Manejador para los pines conectados de la línea 5 a la 9
EXTI015_10_IRQn	EXTI15_10_IRQHandler	Manejador para los pines conectados de la línea 10 a la 15

Tabla 8. Manejador de Interrupciones

Para finalizar con la programación de la interrupción, se crea la función *void EXTI1_IRQHandler (void)* dentro del fichero *stm32f4xx_it.c* en el que cuando se produzca una interrupción, el motor se pare y se encienda el led rojo durante un segundo indicando que la pinza ha llegado a su posición máxima tanto en apertura como en cierre.

9 https://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf

4.5. Diseño de los controladores del sistema

Uno de los objetivos principales de este proyecto consiste en el funcionamiento de la pinza mediante tres tipos de control: control de posición, control de velocidad y control de fuerza. En los siguientes apartados se describe la metodología empleada para realizar estos controles:

4.5.1. Control de Posición

El objetivo que se persigue en este apartado consiste en que el control de la pinza sea capaz de alcanzar una posición determinada tanto en apertura como en cierre con el mínimo error posible. El esquema que describe el funcionamiento de este control se muestra en la Figura 77 en la que el sensor de realimentación es el potenciómetro que se encarga de detectar la posición del motor a cada instante.

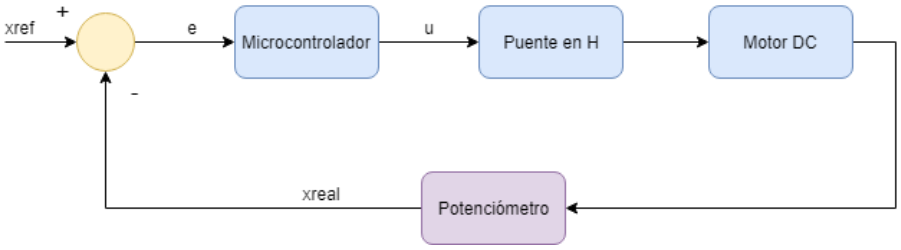


Figura 77. Bucle cerrado del control de Posición

El controlador, en este caso el microcontrolador, se encarga del cálculo del control a realizar en función de la comparación entre la señal real (obtenida por el potenciómetro) y la señal de referencia (introducida por el usuario). Si todos los elementos que se encargan del control de la pinza fueran ideales, la pinza debería arrancar el movimiento con la velocidad definida en la programación (aceleración infinita) hasta llegar a la posición deseada donde debería parar instantáneamente (velocidad cero y deceleración infinita). Esto no es físicamente posible debido a la inercia. Para ello, se ha programado un generador de trayectoria trapezoidal.

4.5.1.1. Generador de Trayectoria Trapezoidal

El generador de trayectoria consiste en la simulación del camino a seguir por la pinza en función de la posición de referencia que se le introduzca. Esto se consigue con un proceso que se divide en tres fases:

- **Primera fase:** se inicia el movimiento del motor con aceleración constante hasta alcanzar la velocidad deseada.
- **Segunda fase:** el motor se mueve a velocidad constante (aceleración nula).
- **Tercera fase:** se inicia la deceleración del motor con una aceleración constante y negativa hasta llegar a la posición de referencia indicada por el usuario.

En la siguiente figura se muestran estas tres fases de manera gráfica:

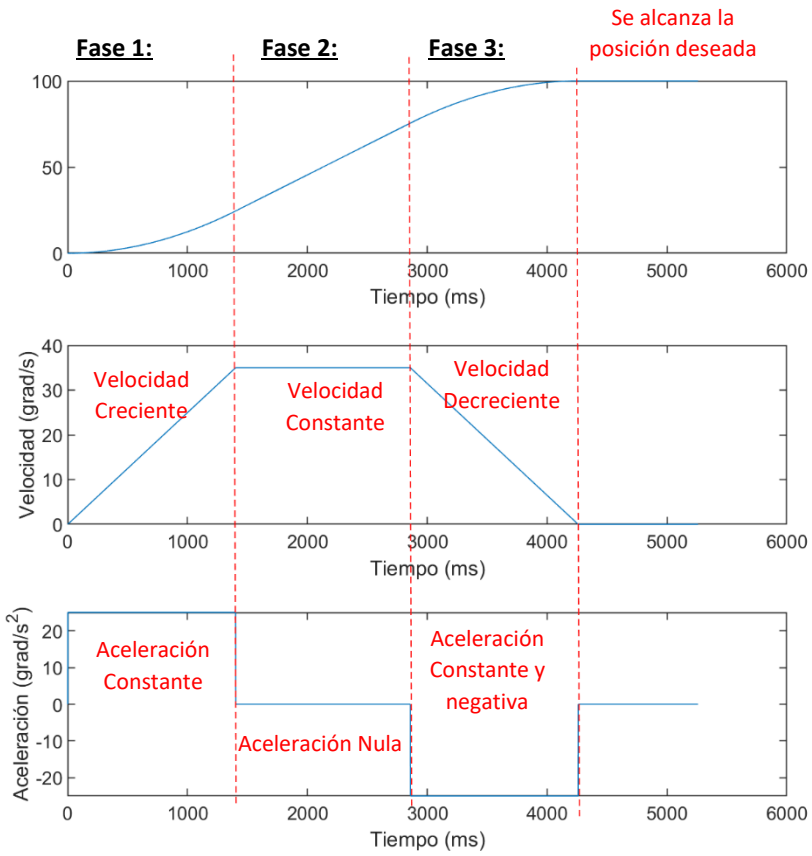


Figura 78. Representación del generador de trayectorias

4.5.1.2. Programación del Control de Posición

Para implementar tanto el generador de trayectorias como el controlador PID en el microcontrolador se programa un fichero cuyo nombre es *ControlPosicion.c* que alberga la función *void GeneradorTrayectoria (float Pos_cm)* a la cual se le introduce la posición en centímetros que se quiere alcanzar. La programación se realiza siguiendo los siguientes pasos:

a. **Introducción de las constantes** que el generador de trayectorias toma como referencia para generar dicha trayectoria. Estas constantes son la posición actual *"qr1"*, la posición a alcanzar *"qr2"*, la velocidad expresada en grados/segundo *"qpr"*, la aceleración expresada en grados/segundo^2 *"qsec"* y el tiempo de muestreo expresado en segundos *"tm"*.

   Los valores de "*qpr*" y *"qsec"* se obtienen de manera iterativa buscando el funcionamiento óptimo para el tipo de motor elegido; el valor de *"qr1"* se obtiene de la función *leer_valor_PB0_grados;* el valor de *"tm"* se ha fijado en 0.001 segundos; y el valor de *"qr2"* se transforma del valor en cm introducido por el usuario al inicio de la función, a grados mediante una de las siguientes ecuaciones (9):

$$qr2 = \frac{\sin^{-1}\left(\frac{Pos_{cm}}{12.5*2}\right)\cdot 360}{2\cdot\pi} \; o \; qr2 = \frac{\sin^{-1}\left(\frac{Pos_{cm}-6}{8.67*2}\right)\cdot 360}{2\cdot\pi} \qquad (9)$$

El hecho de utilizar la primera ecuación o la segunda se basa en el hecho de si se decide realizar el control de posición con los dedos extremos de la pinza o con los interiores, respectivamente. En la Figura 79 se aclara la ubicación de los dedos extremos y de los dedos interiores:

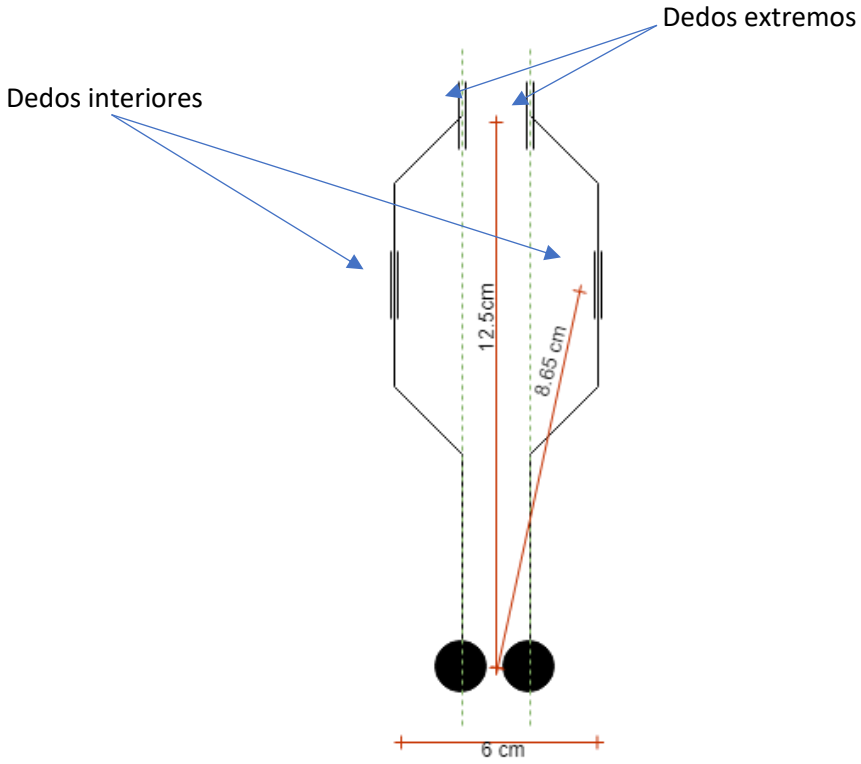


Figura 79. Esqueleto de la pinza

Las ecuaciones anteriores se obtienen realizando las siguientes operaciones:

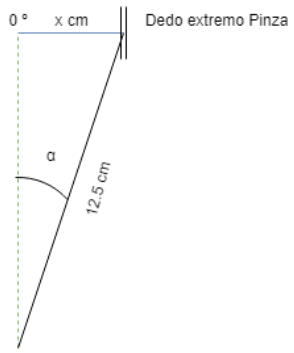


Figura 80. Ángulo entre el dedo superior y el origen

Para obtener la ecuación (9) en primer lugar, se busca el valor de α para los dedos extremos cuya distancia al eje de giro es de 12.5 cm. Por tanto:

$$x = 12.5 \cdot \sin \alpha \qquad (10)$$

El valor de *x* corresponde con la distancia entre el eje central de la pinza y el dedo extremo de uno de los lados. Como se pretende obtener la distancia total entre los dos dedos extremos, se multiplica por dos la ecuación anterior, dando como resultado (10):

$$x = 2 \cdot 12.5 \cdot \sin \alpha \qquad (11)$$

A continuación, puesto que se quiere obtener el ángulo total de apertura, se despejará α y pasará a llamarse *qr2*. Adicionalmente, como el ángulo que se obtiene está en radianes se hace la conversión pertinente para obtener este resultado en grados (12).

$$qr2 = \frac{\sin^{-1}\left(\frac{Pos_{cm}}{12.5 * 2}\right) \cdot 360}{2 \cdot \pi} \quad (12)$$

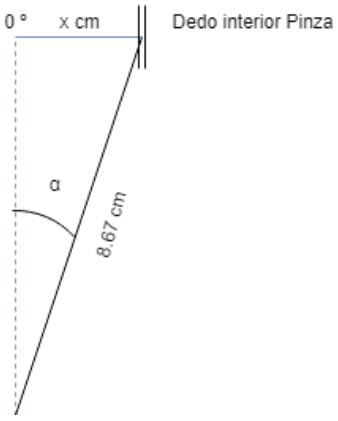


Figura 81. Ángulo entre dedo interior y origen

Finalmente, para obtener la segunda expresión de la ecuación (9), se sigue el mismo procedimiento anterior, pero teniendo en cuenta que la distancia entre los dedos interiores cuando la pinza está cerrada, es de 6 cm. Por tanto, se obtiene la ecuación de la distancia en centímetros entre los dos dedos interiores:

$$x = (2 \cdot 8.67 \cdot \sin\ \alpha) + 6 \quad (13)$$

Por último, se despeja α y se convierte el resultado a radianes siguiendo el procedimiento explicado en el caso anterior. Se obtiene la ecuación (14).

$$qr2 = \frac{\sin^{-1}\left(\frac{Pos_{cm} - 6}{8.67 * 2}\right) \cdot 360}{2 \cdot \pi} \quad (14)$$

b. **Comprobación de la posición actual para comprobar si se tiene que cerrar o abrir la pinza hasta la posición deseada.**
c. **Inicio del bucle de control:** este bucle se recorre cada 1 ms durante todo el tiempo de simulación.
d. **Ejecución del generador de trayectoria:** dividido en tres fases según el tiempo como se explica en el apartado 4.5.1.1
e. **Ejecución del control programado:** se programa un control Proporcional y uno Proporcional – Derivativo tal como se explica en los apartados 4.5.1.3 y 4.5.1.4 respectivamente
f. **Repetición del bucle**: si todavía no se ha alcanzado el tiempo total de simulación, se vuelve a recorrer el bucle.
g. **Fin de proceso**: una vez se ha alcanzado el tiempo total de simulación y el generador de trayectoria ha recorrido todas sus fases, se para el motor y se da por finalizado el control de posición. Adicionalmente se muestran por pantalla los datos de la posición final alcanzada para ver si esta se ha alcanzado o por el contrario se debe ajustar el controlador.

En el siguiente diagrama, se pueden observar de manera gráfica los pasos descritos en este apartado:

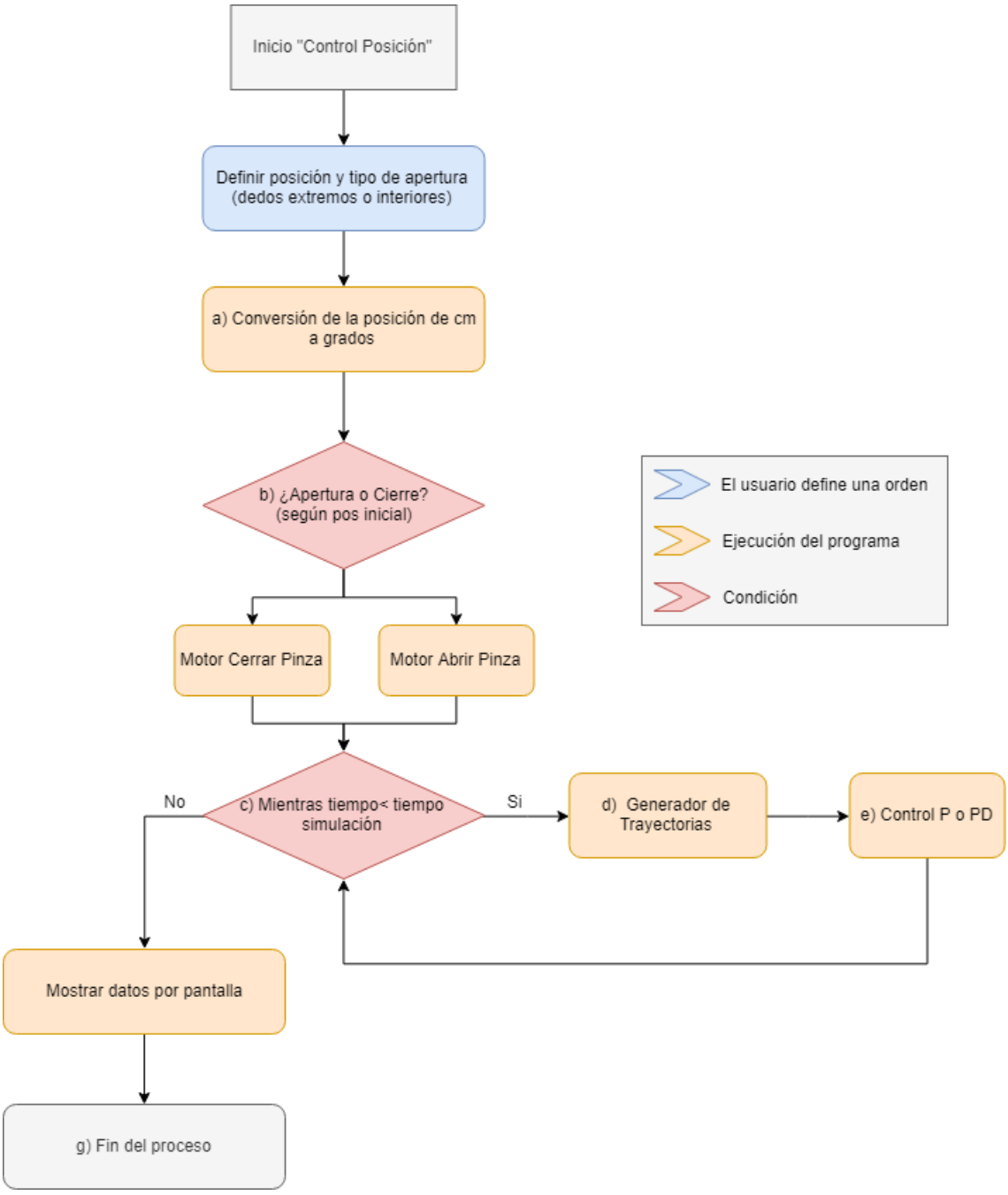


Figura 82. Diagrama funcionamiento Control de Posición

4.5.1.3. Control Proporcional (P)

El control proporcional se realiza de la siguiente manera:

En primer lugar, se calcula el error de posición (*e*) (15), siendo *q* el valor de la posición de referencia calculado en cada instante por el generador de trayectorias y q_{real} el valor de la posición del motor obtenido por la realimentación del potenciómetro en cada instante.

$$e = q - q_{real} \tag{15}$$

A continuación, se obtiene el valor de la acción de control (*u*)(16) a aplicar sobre el motor, siendo k_p el valor de la constante proporcional la cual se ajustará para conseguir un control lo más preciso posible. Cabe destacar, que se ha limitado *u* a valores únicamente de signo positivo puesto que el motor no funciona correctamente con valores negativos.

$$u = k_p \cdot e \tag{16}$$

Finalmente, este control se repite cada 1 ms hasta que se alcanza el tiempo total de simulación. En el apartado 5.1 se realizan las pruebas pertinentes con este tipo de control.

4.5.1.4. Control Proporcional – Derivativo (PD)

También se ha creado un controlador PD debido a que normalmente el control P no es capaz por si solo de alcanzar la posición deseada. Para ello, en primer lugar, al igual que en el control P se calcula el error de posición(17).

$$e = q - q_{real} \tag{17}$$

En segundo lugar, se va a estimar la velocidad del motor en cada instante. Para ello, se resta la posición actual del motor (q_{real}) con la posición del motor en el instante anterior (q_{real_ant}) y se divide el resultado por el valor del tiempo de muestreo (0.001 segundos) dando como resultado v_1 (18).

$$v_1 = \frac{q_{real} - q_{real_{ant}}}{t_m} \tag{18}$$

A continuación, puesto que el motor presenta un alto ruido dificultando el cálculo de la velocidad en cada instante, se ha implementado un filtro de tercer orden como el que se muestra en la ecuación (19), siendo v_{1ant1} y v_{1ant2} las velocidades un instante antes y dos instantes antes respectivamente. A los valores de a_0, a_1 y a_2 se les han asignado valores de 0.7, 0.2 y 0.1 respectivamente, siendo igual a 1 la suma de estos tres valores(20).

$$v_2 = a_0 \cdot v_1 + a_1 \cdot v_{1_{ant1}} + a_2 \cdot v_{1_{ant2}} \tag{19}$$

Donde

$$a_0 + a_1 + a_2 = 1 \tag{20}$$

Finalmente, se actualizan los valores de los instantes anteriores y, al igual que el control P, este control se repite cada 1 ms hasta alcanzar el tiempo total de simulación. En el apartado 5.1 se realizan las pruebas pertinentes con este tipo de control.

4.5.2. Control de Velocidad

Para realizar el control de velocidad de la pinza el sensor de realimentación del control debería ser un tacómetro el cual calcularía la velocidad en cada instante, como se muestra en la Figura 83. Bucle cerrado control de velocidad. En el caso del motor utilizado para el desarrollo de este proyecto, se dispone únicamente como sensor de realimentación el potenciómetro que obtiene la posición del motor en cada instante.

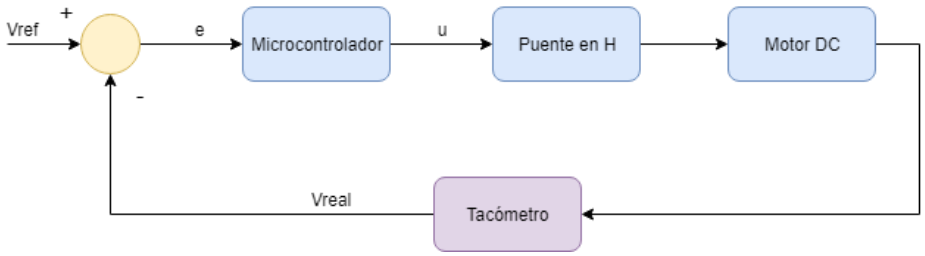


Figura 83. Bucle cerrado control de velocidad

Como alternativa al proceso anterior, se utilizan los datos de posición obtenidos por el potenciómetro para obtener la velocidad en cada instante. Para ello se emplea la siguiente ecuación(21), donde pos_rad_k y pos_rad_{k-1} son las posiciones de la pinza (actual y anterior) expresadas en grados, t_s corresponde al tiempo de muestreo (1 ms) y w_k es la velocidad angular expresada en *grados/ms*.

$$w_k = \frac{pos\_grad_k - pos\_grad_{k-1}}{Ts} \qquad (21)$$

Mediante la expresión anterior, se ha realizado una simulación de apertura de la pinza con una acción de control constante durante 1 segundo con la finalidad de observar la velocidad en cada instante. Los resultados obtenidos en una simulación real son los siguientes:

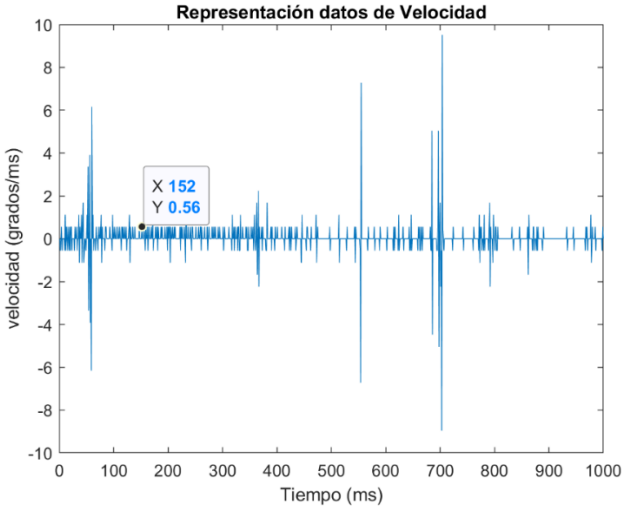


Figura 84. Representación gráfica datos de velocidad

Como se observa en la Figura 84, existe una gran cantidad de ruido en la lectura de la velocidad, por lo que este método de obtención de datos de velocidad queda descartado. Como alternativa, se ha optado por realizar la misma simulación que en el caso anterior, pero en vez de representar gráficamente los datos de la velocidad en cada instante, se representarán los datos de posición en cada instante con la finalidad de comprobar si para un valor constante de acción de control, se mantiene la velocidad constante en todo el recorrido.

Para ello se ha realizado una simulación de 400 ms de duración y 1 ms de tiempo de muestreo con un *Duty Cycle* del 50 % (en torno a 1.36 V de acción de control sobre el motor). Esta simulación se representa en la Figura 85 y como se observa, el desplazamiento para la acción de control aplicada se mantiene constante a lo largo de toda la simulación, con lo que se puede deducir que la velocidad también se mantendrá constante. En cuanto al ruido, sigue existiendo, pero la magnitud de este es reducida a excepción de algún dato de posición anómalo.

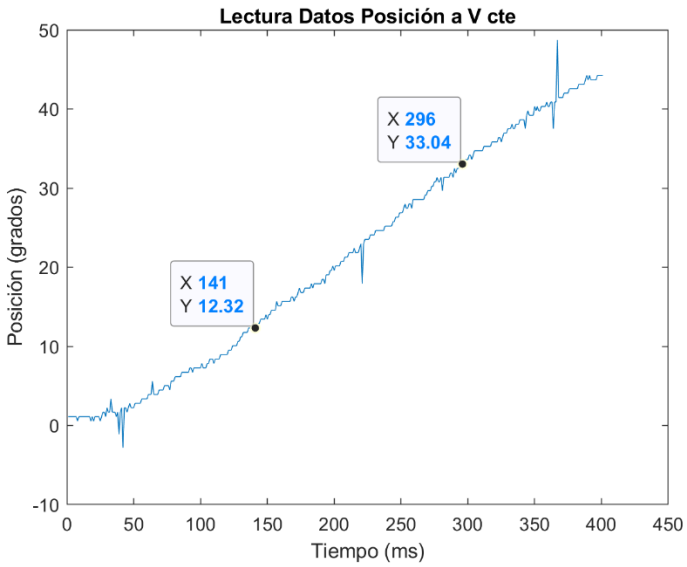


Figura 85. Representación gráfica datos posición a velocidad constante

Para comprobar si la velocidad se mantiene constante a lo largo del recorrido, se divide la diferencia del espacio recorrido entre los dos puntos marcados de la figura entre el tiempo invertido entre ambos instantes como se describe en la ecuación (22).

$$w\left(\frac{grados}{ms}\right) = \frac{pos_{final} - pos_{inicial}}{tiempo_{final} - tiempo_{incial}} = \frac{33.04 - 12.32}{296 - 141} = 0.134\frac{grados}{ms} \qquad (22)$$

Esta misma operación se ha realizado en diferentes puntos de la gráfica (evitando la zona de aceleración del motor entre 0 y 100 ms), obteniendo los siguientes resultados:

Posición Inicial (grados)	Posición Final (grados)	Tiempo Inicial (ms)	Tiempo Final (ms)	Velocidad Angular (grados/ms)
7.28	39.2	104	349	0.130
26.32	38.64	248	342	0.131
8.4	19.6	114	200	0.130

Tabla 9. Datos velocidad con ciclo de trabajo del 50%

Como se puede comprobar en la Tabla 9, la velocidad se mantiene prácticamente constante en (unos 0.13 grados/ms) todo el recorrido para una acción de control constante. Este mismo procedimiento se repite para diferentes valores del *Duty Cycle*, siendo los resultados obtenidos los que se muestran en la Tabla 10:

% *Duty Cycle*	Velocidad (grados/ms)
25	0.085
50	0.131
75	0.179
100	0.205

Tabla 10. Datos Velocidades con diferentes ciclos de trabajo

A continuación, se ha creado dentro del fichero *ControlVelocidad.c* la función *void ControlVelocidad (int sentido, int a_control, int impr_pantalla)*, a la cual se le introducirá el sentido y velocidad a la que se quiere que se desplace la pinza, y también se introduce si se quiere que se imprima por pantalla la velocidad de movimiento. Esta velocidad se introducirá a través del valor del *Duty Cycle* (a elegir por el usuario entre 25 %, 50 %, 75 % o 100 %). Esta función realizará el movimiento en el sentido y velocidad seleccionada y devolverá por pantalla la velocidad calculada entre dos instantes intermedios para comprobar si esta velocidad se corresponde con las velocidades calculadas en la tabla anterior. Estas pruebas se realizan en el apartado 5.2.

En el siguiente diagrama de bloques se muestra de manera gráfica el funcionamiento del control de velocidad:

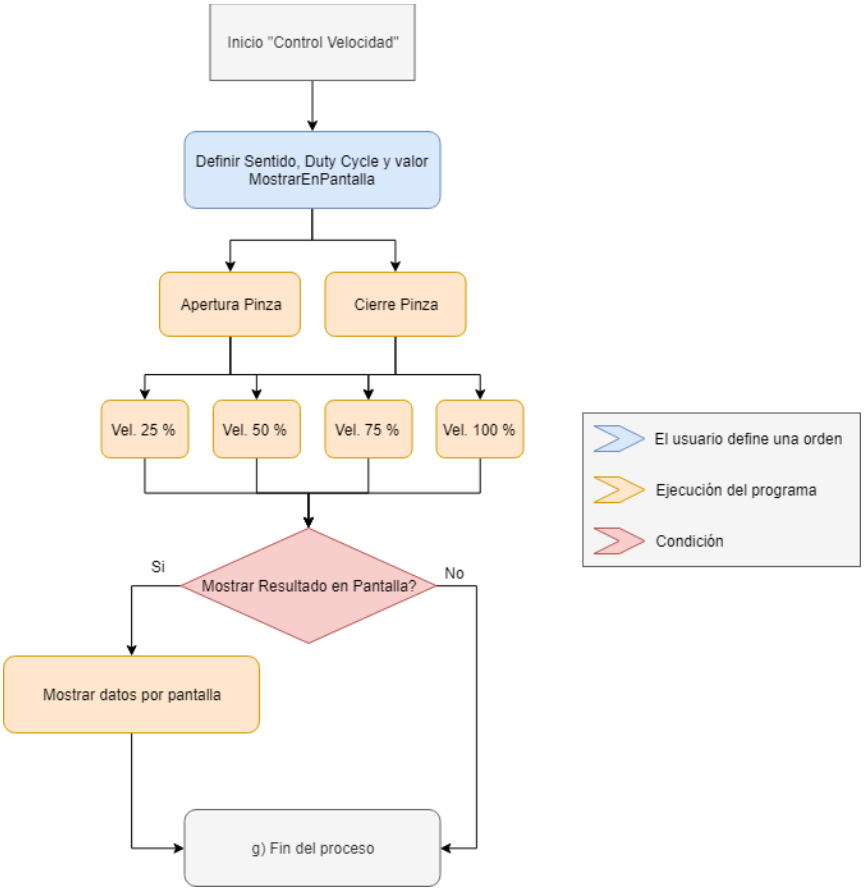


Figura 86. Diagrama funcionamiento Control Velocidad

4.5.3. Control de Fuerza

El objetivo que se persigue en este apartado consiste en que el control permita a la pinza agarrar objetos con una fuerza determinada. En este caso, el sensor de realimentación del bucle cerrado serán los sensores de fuerza tal y como se muestra en la Figura 87:

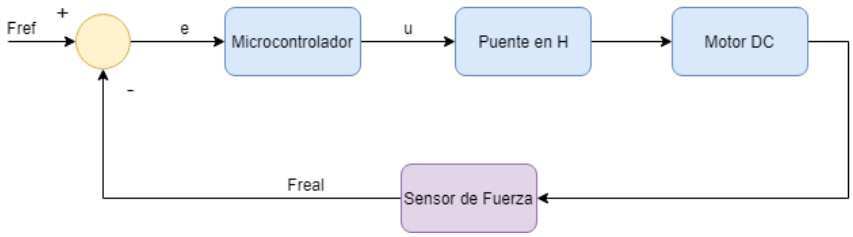


Figura 87. Bucle cerrado Control de Fuerza

Para la realización del control de fuerza, se han utilizado los 8 sensores de fuerza descritos en el apartado 4.3.2. Estos se han ubicado en ambas partes de la pinza tanto en sus extremos como en su parte interior (Figura 88) para poder así detectar fuerzas con los dedos extremos de la pinza (objetos de menor tamaño) y con los interiores (objetos de mayor tamaño).

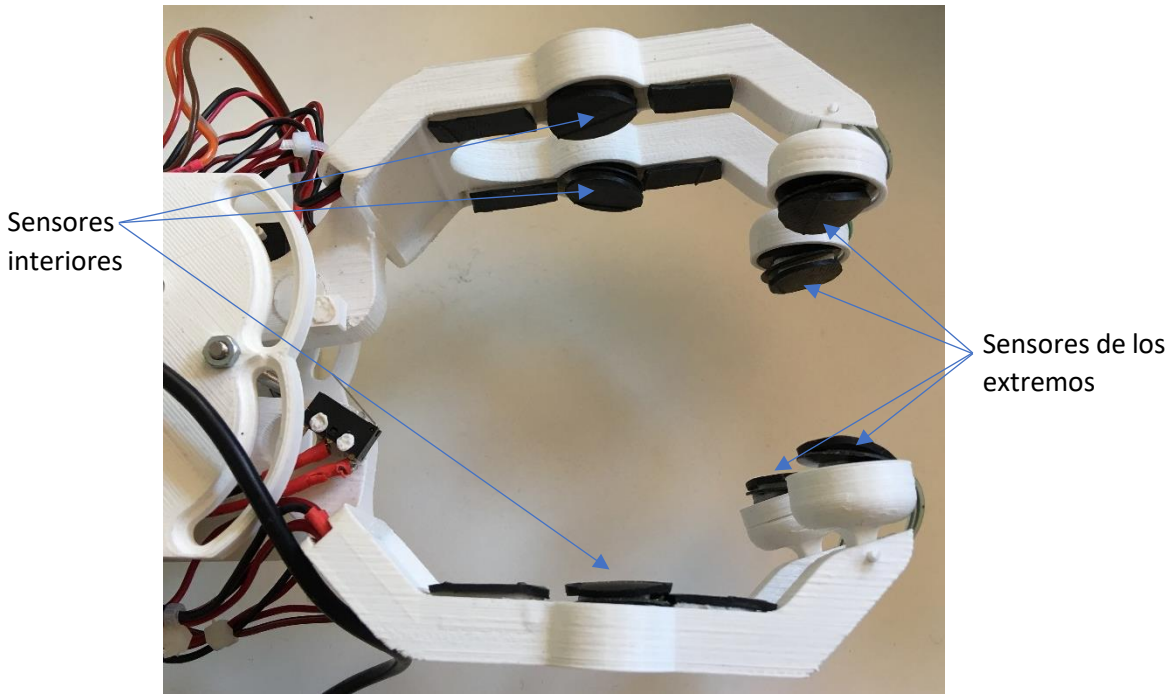


Figura 88. Ubicación de los sensores sobre la pinza

Para conseguir el agarre con realimentación de fuerza, en primer lugar, se tiene que transformar la tensión leída por los sensores a una unidad de fuerza, en este caso Newtons. Para ello, se emplea la ecuación de primer grado de la recta obtenida en el apartado 4.3.2.4 tras realizar las pruebas de calibración de los sensores. Esta expresión y su transformación se muestran en las siguientes ecuaciones. En la primera, se muestra la ecuación de la recta (23), en la segunda se calcula el valor de la masa (24), mientras que, en la tercera, esta masa se transforma en fuerza (25).

$$y = 0.0048 \cdot x + 0.2763 \qquad (23)$$

$$masa = \frac{leer_{valor_{PCo_{voltios}}} - 0.2763}{0.0048} \quad (24)$$

$$Freal = \frac{masa \cdot 9.81}{1000} \quad (25)$$

Una vez obtenida la ecuación que transforma los voltios de los sensores a unidades de fuerza, se crea el fichero *ControlFuerza.c* que contiene dos funciones: void ControlFuerzaExterior (float f_ref, float u_ref) y *void ControlFuerzaInterior (float f_ref, float u_ref)* a las cuales se les introduce el valor de la fuerza a ejercer y un valor de la fuerza que se tendrá que ejercer como referencia a seguir. Ambas funciones realizan el mismo control de fuerza, pero cada una se ejecuta en función de si se quiere ejercer fuerza en los sensores de los extremos o con los interiores.

Estas funciones se dividen en dos fases:

1ª Fase: consiste en el acercamiento de la pinza a velocidad reducida y constante hasta que uno de los sensores detecta contacto. Para ello, se han realizado pruebas con los sensores para establecer la fuerza umbral que indicará el contacto de al menos uno de los sensores con el objeto a agarrar. Esto se ha programado dentro de un bucle *while* en el que la condición para salir de dicho bucle (1ª fase) es que $f_{medida} > umbral$.

2ª Fase: una vez superada la fase anterior se inicia la segunda fase la cual consiste en la programación del control de fuerza. Para ello, en primer lugar, se programa un filtro cuya finalidad es suavizar la señal de fuerza recibida por los sensores debido al alto ruido que se genera cuando estos detectan fuerza. El filtro implementado es el que se muestra en la siguiente ecuación (26), donde f_{medida} es la fuerza captada por uno de los sensores, $f_{medida} \cdot z^{-1}$ y $f_{medida} \cdot z^{-2}$ son las fuerzas del sensor en el instante anterior, y dos instantes anteriores, respectivamente, dando lugar a $f_{estimada}$ siendo esta la fuerza filtrada.

$$\hat{f} = a_0 \cdot f_{medida} + a_1 \cdot f_{medida} \cdot z^{-1} + a_2 \cdot f_{medida} \cdot z^{-2} \quad (26)$$

La suma de los valores de a_0, a_1 y a_2 debe ser igual a uno (27):

$$a_0 + a_1 + a_2 = 1 \quad (27)$$

Donde (28):

$$a_0 > a_1 > a_2 \quad (28)$$

A continuación, se obtiene el valor de la derivada de la fuerza estimada obtenida en la ecuación (29) la cual será necesaria para realizar el control de posición:

$$\dot{\hat{f}} = \hat{f} - \dot{\hat{f}} \cdot z^{-1} \quad (29)$$

El siguiente paso consiste en el cálculo de la acción de control *u* la cual se puede obtener con un control PID (30) o con un control PD (31) con prealimentación de la fuerza de referencia:

1) Control PID:

$$u = k_p \cdot (f_{ref} - \hat{f}) - k_v \cdot \dot{\hat{f}} + k_i \cdot \int (f_{ref} - \hat{f}) dt \quad (30)$$

2) Control PD con prealimentación:

$$u = k_p \cdot (f_{ref} - \hat{f}) + k_v \cdot \left(\dot{f_{ref}} - \dot{\hat{f}}\right) + F_{ref} \quad (31)$$

Para realizar el control en este caso se elige el control PD con prealimentación debido a que tras varias pruebas con el control PID, este no funcionaba de manera correcta. En el apartado 5.3 se realizan las pruebas del control PD de fuerza.

En la siguiente figura se muestra de forma esquemática el proceso que lleva a cabo la función *ControlFuerza:*

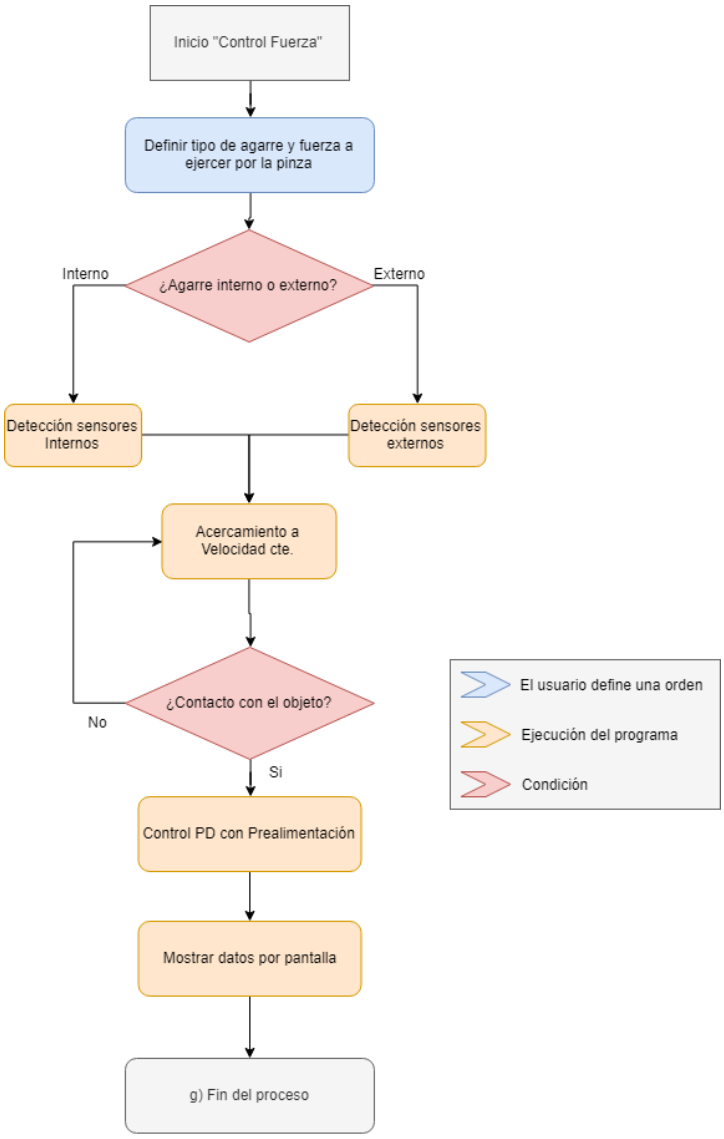


Figura 89. Diagrama funcionamiento Control de Fuerza

4.6. Interfaz

Con la finalidad de crear un prototipo de pinza robótica lo más funcional posible, se ha implementado una interfaz gráfica aprovechando la presencia de una pantalla LCD integrada en la tarjeta *STM32F429I.* La utilización de esta pantalla permitiría al usuario futuro interactuar con las diferentes funcionalidades de la pinza de manera sencilla e intuitiva. Es por ello por lo que, para controlar todas las funcionalidades de la pantalla, solamente son necesarios dos botones (Figura 90), uno para cambiar entre las diferentes opciones *"SELECT"* y el otro para confirmar la acción a ejecutar *"OK".*

Figura 90. Ubicación de los botones que controlan la interfaz

La interfaz está diseñada con diferentes submenús elegibles a partir de un menú principal entre los cuales el usuario se puede mover pulsando sobre *"Atrás"* sin la necesidad de reiniciar la tarjeta tras cada opción seleccionada.

Para comprender el funcionamiento de la interfaz, el siguiente esquema muestra el esqueleto del programa:

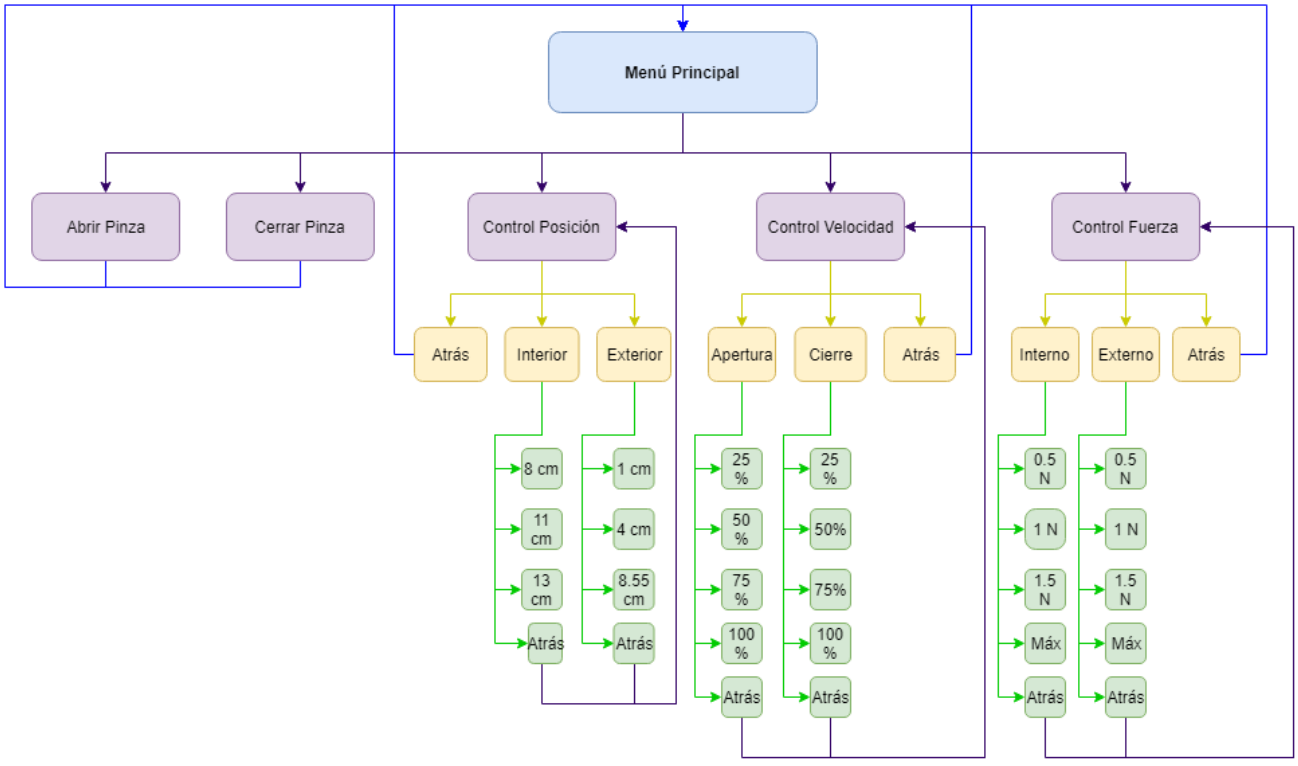


Figura 91. Diagrama de funcionamiento de la interfaz

A continuación, se muestran en imágenes los distintos menús de la interfaz:

Menú Principal:

Figura 92. Interfaz Menú Principal

La Figura 92 muestra el menú principal en el que se puede seleccionar cualquiera de las opciones. El punto rojo indica la ubicación del cursor, el cual se desplaza entre las opciones pulsando el botón *"SELECT"*. Para aceptar la selección se tiene que pulsar el botón *"OK"*.

Abrir Pinza y Cerrar Pinza:

Figura 93. Ejecución Abrir Pinza y Cerrar Pinza

Las opciones *"Abrir Pinza"* y *"Cerrar Pinza"* (Figura 93) no disponen de submenú, pero cuando se seleccionan, se ejecuta el proceso y sale por pantalla un mensaje de *"OK"* confirmando que el proceso ha sido realizado.

Control de Posición:

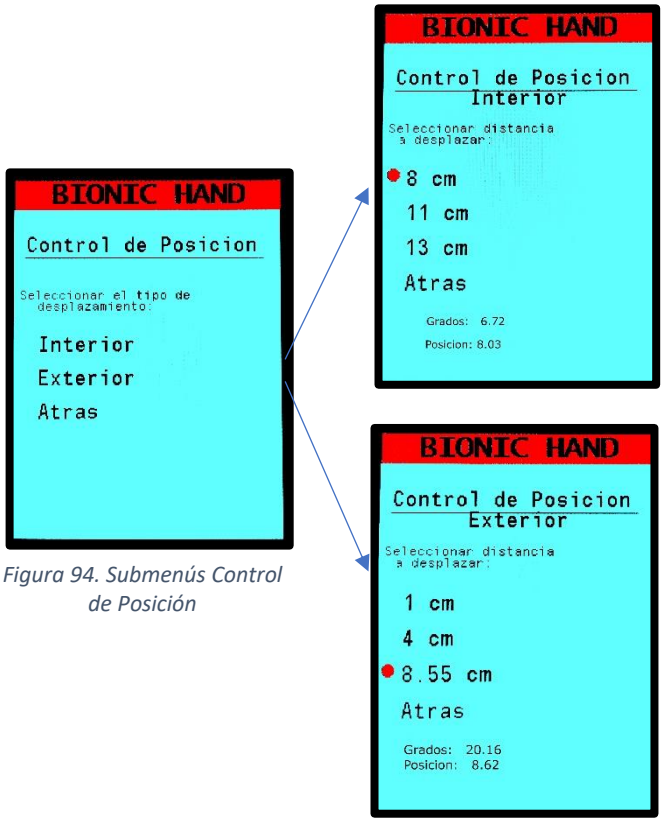


Figura 94. Submenús Control de Posición

Si se pulsa la opción *"Control Posición"* se abre el submenú en el cual se ha de seleccionar el tipo de desplazamiento que se quiere ejecutar pudiendo elegir entre *"Interior"* y *"Exterior"*.

Interior: en este menú se muestran las 3 posiciones programadas a las cuales la pinza se puede desplazar. Si se pulsa sobre cualquiera de ellas, el movimiento se inicia hasta la posición deseada, y cuando este proceso se completa, se muestra un mensaje en pantalla mostrando la posición real alcanzada en grados y en centímetros.

Exterior: en este menú se muestran las opciones disponibles que se pueden ejecutar siendo el funcionamiento el mismo que en el caso del submenú *"Interior"*

Control de Velocidad:

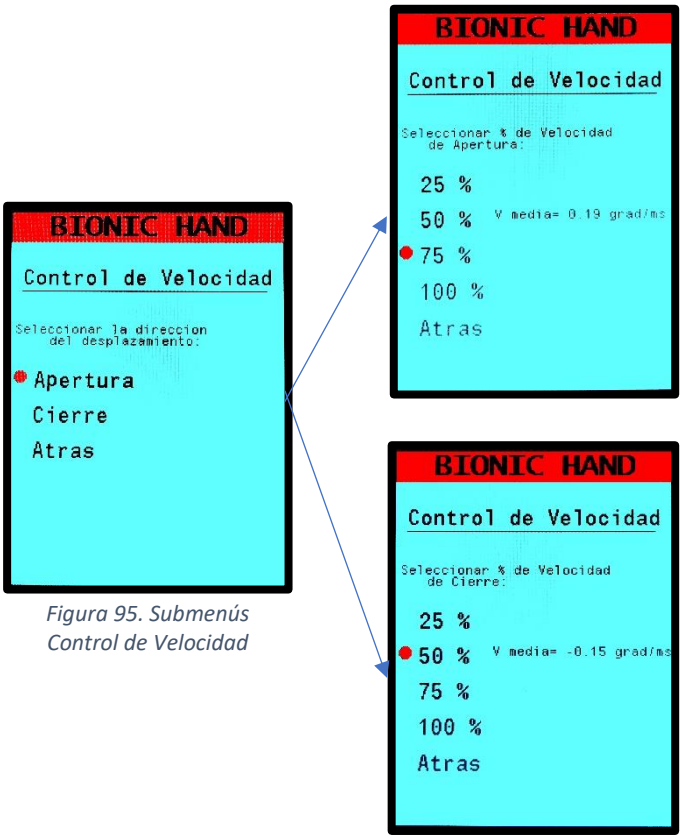


Figura 95. Submenús Control de Velocidad

Si se pulsa la opción *"Control Velocidad"* se abre un submenú en el cual se ha de elegir la dirección del desplazamiento, es decir, *"Apertura"* o *"Cierre"*.

Apertura: en este menú se muestran las cuatro posibles velocidades a las que la pinza se puede mover. Si se selecciona cualquiera de ellas, el proceso se ejecuta y la pinza se mueve a la velocidad seleccionada. Al acabar este proceso, se muestra por pantalla la velocidad a la que se ha desplazado la pinza.

Cierre: en este menú se muestran las mismas opciones que en el caso anterior, pero en este caso la apertura se realiza en el sentido contrario, por lo que la velocidad devuelta por pantalla tendrá signo negativo.

Control de Fuerza:

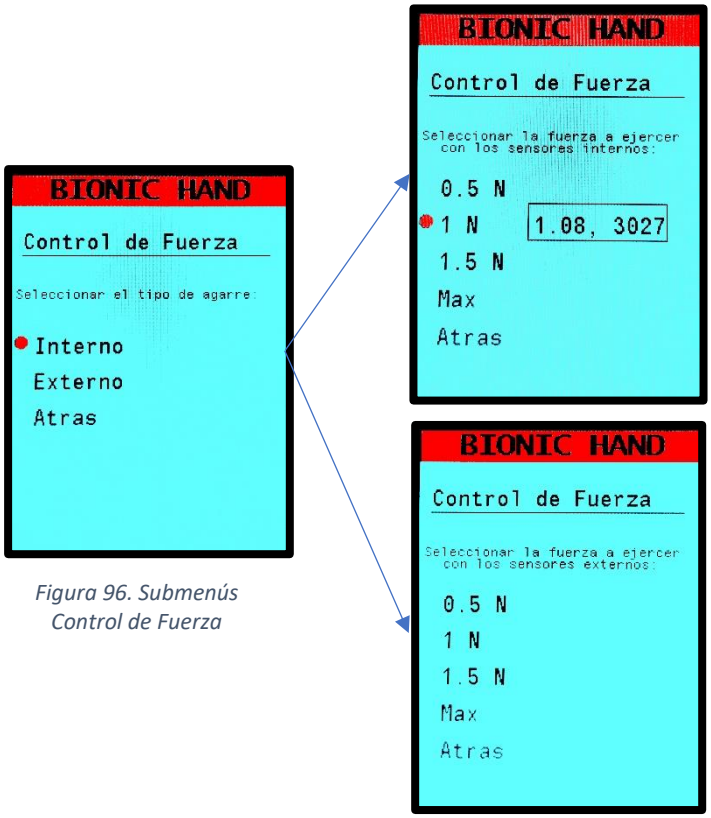


Figura 96. Submenús Control de Fuerza

Si se pulsa la opción *"Control Fuerza"* se abre un submenú en el cual se ha de elegir el tipo de agarre pudiendo elegir entre *"Interno"* o *"Externo".*

Interno: en este menú se muestran las diferentes fuerzas programadas que puede ejercer la pinza sobre un objeto. Cuando se selecciona alguna de ellas, se ejecuta el proceso y se muestra por pantalla la fuerza real que se está ejerciendo junto con la acción de control aplicada.

Externo: en este menú al igual que en el anterior, se muestran las fuerzas seleccionables, y si se selecciona cualquiera de ellas, se muestra por pantalla la fuerza real que se ejerce acompañada de la acción de control aplicada.

5. Pruebas de funcionamiento

5.1. Control de Posición

Las pruebas de funcionamiento realizadas sobre el control de posición se basan en la prueba de diferentes de diferentes valores de k_p en el caso del control P, o de k_p y k_v en el caso del control PD con la finalidad de encontrar un controlador que genere un seguimiento de trayectoria lo más ajustado a la referencia posible.

Pruebas con el control P:

En primer lugar, se realizan pruebas con el control proporcional diseñado en el apartado 4.5.1.3 con un valor de k_p de 1500 para la apertura de 13 cm de la pinza con respecto a los dedos de la parte interior de esta. La separación de los dedos cuando la pinza está completamente cerrada es de 6 cm, por lo que el recorrido se realiza de 6 a 13 cm, tal y como se muestra en la Figura 97.

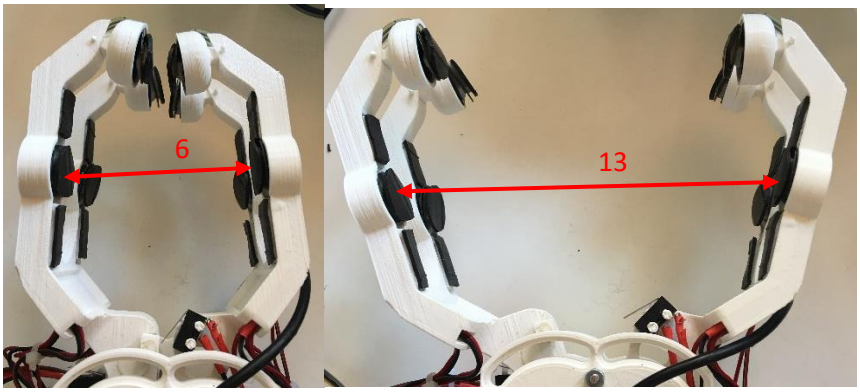


Figura 97. Desplazamiento de la pinza de 6 a 13 cm

Al acabar este recorrido en la pantalla del microcontrolador, se muestra la posición que se ha alcanzado (Figura 98). En este ejemplo, la posición que se ha alcanzado es 12.61 cm cuando debería haber alcanzado los 13 cm, con lo que se observa que hay un pequeño error de posición (unos 4 mm).

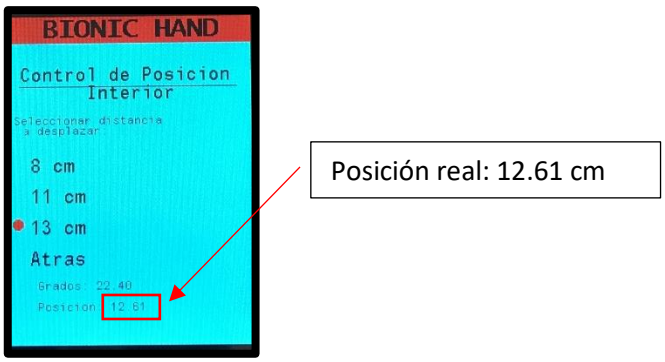


Figura 98. Representación de la posición real sobre la interfaz

Para comprobar este resultado de manera más precisa, se ha representado en una misma gráfica la trayectoria a seguir (creada por el generador de trayectorias) y la trayectoria realmente seguida por la pinza. En la gráfica (Figura 99) se comprueba que la pinza es capaz de seguir la trayectoria, pero con una cierta oscilación, llegando al final de la simulación con el error de posición comentado en el párrafo anterior.

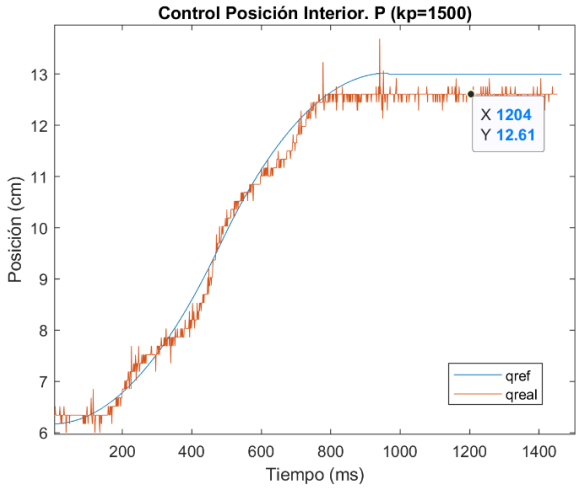


Figura 99. Representación gráfica Control Posición Interior. P (kp=1500)

Tras estos resultados, se ha ajustado el parámetro K_p hasta el valor de 2500 obteniendo así un error de posición prácticamente nulo (12.92 cm respecto a los 13 cm a alcanzar), aunque la oscilación en el seguimiento de la trayectoria aún se mantiene. Estos resultados se muestran a continuación:

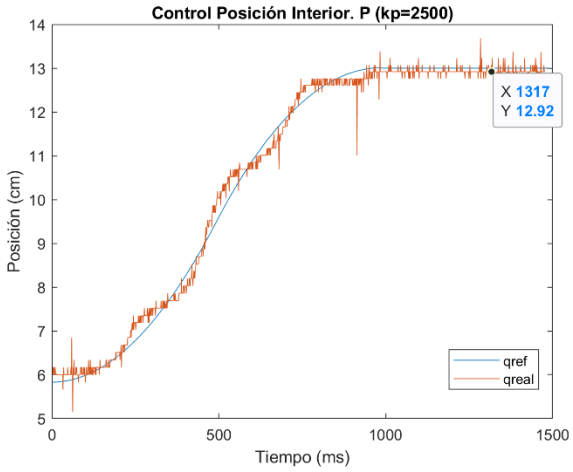


Figura 100. Representación gráfica Control Posición Interior. P (kp=2500)

A continuación, con la finalidad de mitigar esta oscilación durante el movimiento, se realizan pruebas con el controlador PD diseñado:

Pruebas con el control PD:

Después del diseño del control Proporcional-Derivativo tal como se explica en el apartado 4.5.1.4, se realiza una prueba de funcionamiento estableciendo como valor de $k_p = 700$ y $k_v = 5$. En esta prueba la medida de posición se realizará entre los dedos de los extremos, estando estos a 0 cm cuando la pinza está completamente cerrada. El objetivo es que la pinza se desplace de 0 cm a 8.55 cm respecto a los dedos extremos, como se muestra en la Figura 101.

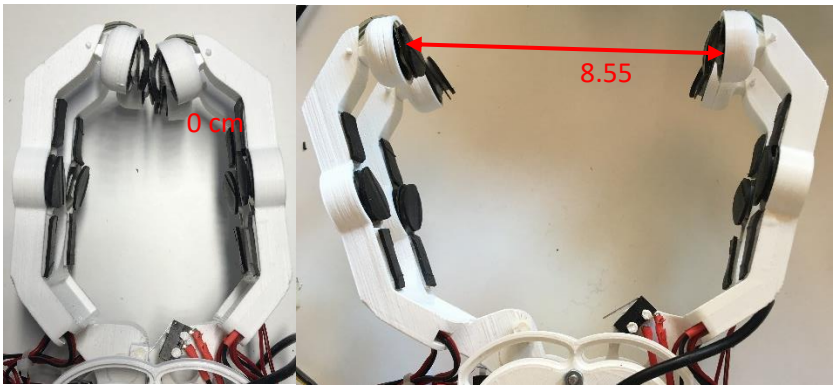


Figura 101. Desplazamiento de la pinza de 0 a 8.55 cm

Tras realizar esta prueba la representación gráfica del recorrido realizado por la pinza es la que se muestra en la gráfica de la Figura 102, donde se observa que no se sigue la referencia y tampoco el control es capaz de alcanzar la posición final por lo que los valores del control se deberán ajustar para conseguir un resultado más preciso.

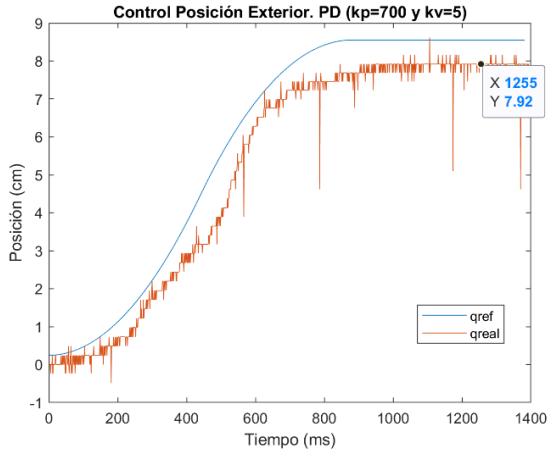


Figura 102. Representación gráfica Control Posición Exterior. PD (kp=700 y kv=5)

A continuación, ajustando los valores de los parámetros del controlador a $k_p = 1200$ y $k_v = 15$ se consiguen unos resultados notablemente mejores que en el caso anterior. En este caso la trayectoria se consigue seguir con mayor precisión junto con la posición final que se alcanza con un error muy reducido ($error = 8.62 - 8.55 = 0.07\ cm$) como se muestra en la Figura 103.

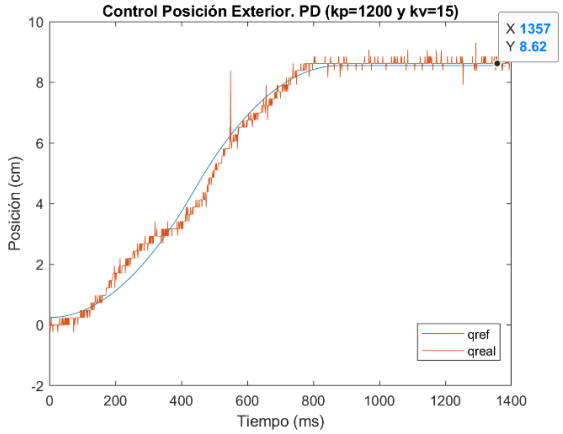


Figura 103. Representación gráfica Control de Posición Exterior. PD (kp=1200 y kv=15)

Finalmente, con la finalidad de verificar que este control funciona correctamente, se prueba el mismo controlador PD que en la prueba anterior, pero en vez de simular la apertura de la pinza hasta un punto concreto se simula el cierre hasta dicho punto, en este caso desde 17 cm hasta 4 cm. En cuanto a los resultados obtenidos (Figura 104), la trayectoria se sigue con un poco más de dificultad que en el caso anterior, pero la posición final se alcanza con un pequeño error ($error = 4 - 3.89 = 0.11\ cm$).

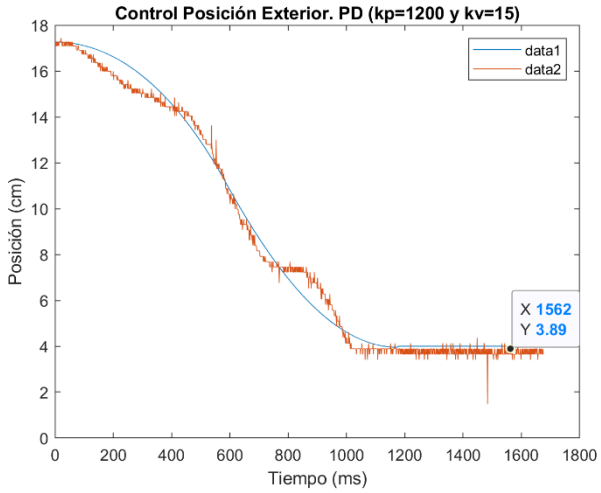


Figura 104. Representación gráfica Control de Posición Exterior. PD (kp=1200 y kv=15)

5.2. Control de Velocidad

En cuanto al control de velocidad, se van a comparar los datos de velocidad que calcula el control tras ser realizado con los datos de la Tabla 11 obtenida en el apartado 4.5.2 en la que ha representado gráficamente el desplazamiento de la pinza en función del tiempo, para más tarde calcular la velocidad media recorrida por esta. Para ello se ejecutan una a una todas las opciones de velocidad que se han programado tanto en sentido de apertura como en sentido de cierre dando lugar a la siguiente tabla:

% Duty Cycle	Velocidad de estudio (grados/ms)	Sim. Apertura (grados/ms)	Sim. Cierre (grados/ms)
25	0.085	0.07	-0.07
50	0.131	0.15	-0.15
75	0.179	0.18	-0.19
100	0.205	0.21	-0.22

Tabla 11. Datos pruebas control de velocidad

Como se observa en la tabla anterior, los datos de velocidad de estudio son similares a los que calcula la pinza entre dos instantes después de cada simulación. Tanto en apertura como en cierre, la pinza es capaz de mantener una velocidad constante similar con lo que el comportamiento que presenta el motor es bueno.

5.3. Control de Fuerza

Para realizar el control de fuerza, se realizan una serie de pruebas en las que la pinza debe sujetar objetos con diferentes fuerzas, tanto en el agarre interno como en el externo.

Pruebas con el agarre interno:

El agarre interno de la pinza está pensado para sujetar objetos de un diámetro máximo de 10 cm de diámetro y unos 500 gramos de peso.

La primera prueba de agarre se realiza con una caja rectangular con un peso de 100 gramos, probando a sujetarla con un valor de fuerza de 0.5 N. Como se observa en la Figura 105, el agarre se realiza de manera correcta.

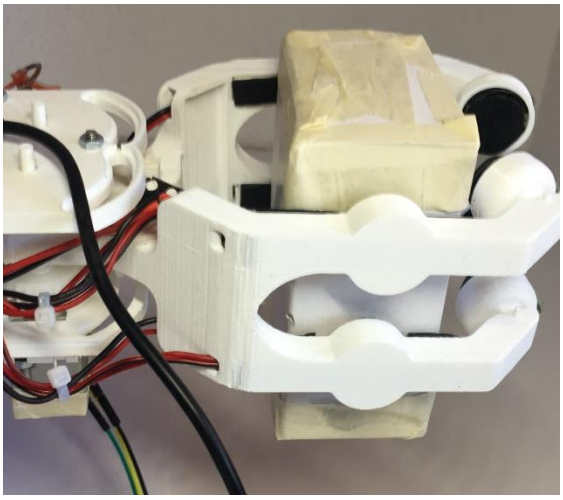


Figura 105. Agarre Interno. Caja 100 g

La segunda prueba de agarre se realiza con un vaso de vidrio de unos 200 gramos probando a sujetarlo con un valor de fuerza de 0.5 N como en el caso anterior. Como se observa en la Figura 106, se consigue aguantar el vaso sin que deslice en al menos 10 segundos puesto que se comprueba que al ser el vaso de cristal se necesitaría ejercer una fuerza un poco mayor o aumentar la fricción.

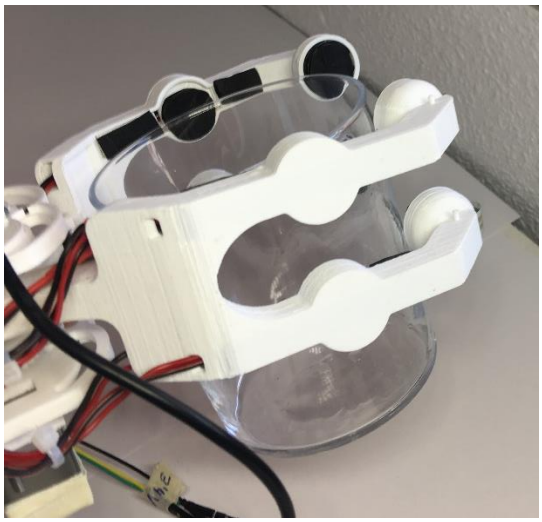


Figura 106. Agarre Interno. Vaso 200 g

La tercera prueba, es una evolución de la segunda prueba puesto que se vuelve a realizar el agarre del vaso anterior pero esta vez este contiene agua siendo el peso total del conjunto, de 350 gramos. Al pesar más, se ha de aplicar mayor fuerza, por lo que se prueba 1.5 N y el vaso se aguanta, pero se observa como poco a poco este va deslizándose, por lo que si en vez de aplicar 1.5 N se aplica la opción *Max*, la pinza agarra el objeto con toda la potencia disponible y, por tanto el vaso se consigue sujetar con firmeza como se muestra en la Figura 107.

Figura 107. Agarre Interno. Vaso 350 g

En la cuarta prueba se intenta agarrar una taza de 400 gramos y un diámetro de 9 cm, pero se comprueba que la pinza no es capaz de agarrarla por lo que se descarta que la pinza pueda sujetar objetos de diámetros mayores a 9 cm y más de 400 g.

Finalmente, la última prueba se realiza a una botella de cristal de 400 gramos y un diámetro de 7.5 cm. Para aguantar esta botella se tiene que seleccionar la máxima fuerza para que la botella no corra el peligro de resbalar y caer. En la siguiente figura se muestra como la pinza es capaz de sujetar esta botella con firmeza.

Figura 108. Agarre Interno. Botella 400 g

Pruebas con el agarre externo:

El agarre externo se ha diseñado con la finalidad de sujetar objetos de menor tamaño y peso que los que es capaz de sujetar la parte interior de la pinza. Para comprobar si es capaz de sujetar determinados objetos, se realizan las siguientes pruebas:

La primera prueba consiste en el agarre de un huevo de unos 70 gramos. Puesto que se trata de un objeto frágil se intenta sujetarlo de manera exitosa con una fuerza de 0.5 N tal y como se muestra en la Figura 109.

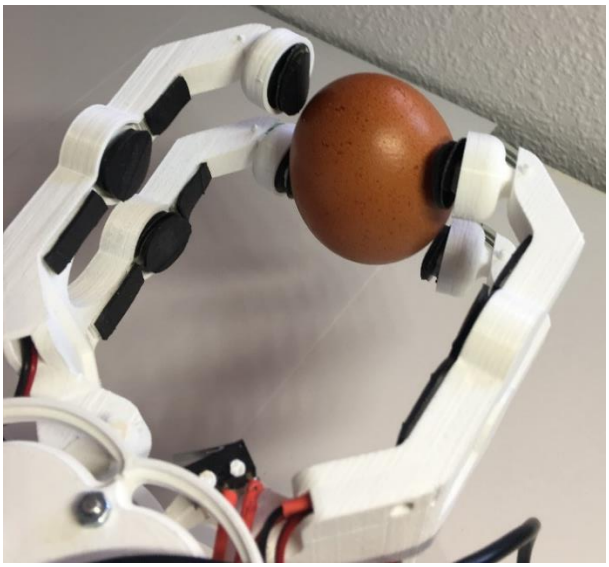


Figura 109. Agarre Externo. Huevo 70 g

La segunda prueba consiste en la sujeción de una lata de refresco vacía (30 gramos) con un agarre de 0.5 N el cual también se realiza de manera exitosa con una gran firmeza, como se muestra en la Figura 110.

Figura 110. Agarre Externo. Lata 30 g

En la tercera prueba se añade más carga, por lo que la pinza se somete al agarre de una lata de maíz de 200 gramos con una fuerza de 1 N. Esta prueba se realiza de manera exitosa consiguiendo al igual que el caso anterior un agarre firme.

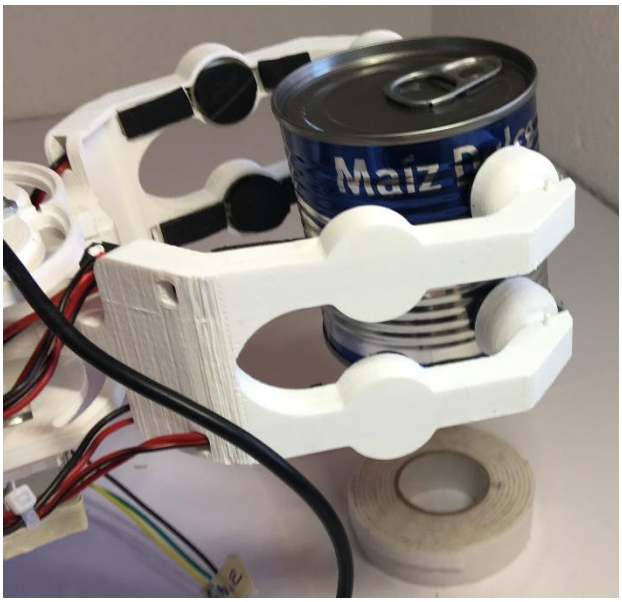


Figura 111. Agarre Externo. Bote 200 g

Finalmente, en la cuarta prueba se realiza el agarre de un tenedor que por su reducido peso debería poder agarrarse con 0.5 N de fuerza, pero como tiene un espesor muy fino, el tenedor es agarrado con poca firmeza, en cambio si se aplica una fuerza máxima el tenedor es sujetado sin ningún problema y no corre el riesgo de caer.

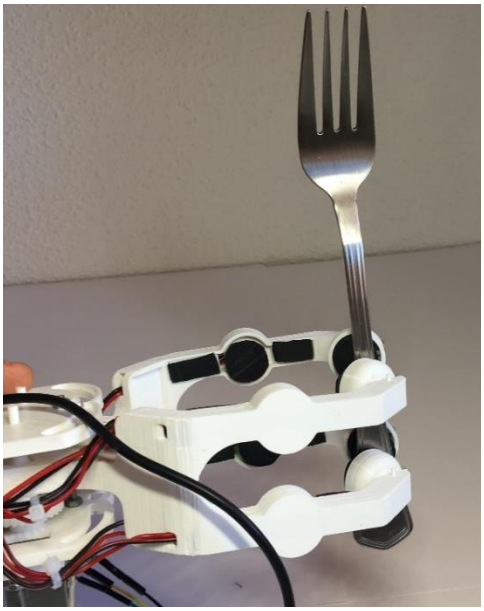


Figura 112. Agarre Externo. Tenedor

Tras todo este conjunto de pruebas, se comprueba que los objetos pueden ser agarrados, aunque se concluye que los sensores empleados no son los ideales para este tipo de aplicación, puesto que la lectura de los datos de fuerza presenta mucho ruido y por tanto cada vez que se realiza una serie de simulaciones con características similares los resultados presentan una gran variabilidad.

6. Conclusiones

6.1. Conclusión

Tras realizar todo el proyecto, se puede afirmar que se ha conseguido un prototipo funcional, robusto y acabado con el que se alcanzan prácticamente todos los objetivos planteados inicialmente con un presupuesto inferior a los 200 euros.

Respecto al diseño de la pinza, esta ha sido diseñada con la finalidad de poder agarrar todo tipo de objetos, aunque finalmente se comprueba que hay ciertas limitaciones estructurales que no permiten que la pinza pueda agarrar objetos de pequeño tamaño u objetos pesados. Esto es debido al deslizamiento entre los objetos (hechos con materiales resbaladizos, como el cristal) y el material antideslizante de caucho (utilizado para aumentar la fricción entre la pinza y los objetos a sujetar).

Continuando con el desarrollo de la electrónica, esta se ha diseñado/montado de la manera más sencilla posible, dando lugar a un circuito simple con pocos componentes. Cabría destacar la eliminación de la electrónica interna del servomotor para el diseño posterior de los controladores del motor. Uno de los problemas surgidos en relación con el motor se basa en el calentamiento del puente en H empleado para controlar su velocidad y sentido de giro. Esto se ha solucionado fácilmente con el reemplazo del puente en H por otro de la misma categoría, pero con unos rangos de funcionamiento más amplios (*L293D -> L293B*).

En cuanto a los sensores de fuerza empleados, estos comprenden el mayor de los problemas surgidos en este proyecto, puesto que el mal comportamiento en cuanto a ruidos y precisión ha provocado que algunos de los controles diseñados no presenten los resultados esperados en un primer momento. Estos sensores no han podido ser sustituidos por otro tipo de sensores mejores debido a diferentes factores entre los cuales se pueden destacar las limitaciones de tiempo, la lentitud de los envíos internacionales y en mayor medida, el precio, puesto que unos sensores de alta calidad provocarían un aumento de precio no contemplado por ser uno de los objetivos principales el implementar una pinza robótica de bajo coste.

Siguiendo con la programación del microcontrolador, esta comporta la mayor parte del tiempo de realización del proyecto, ya que a pesar de que el autor posee conocimientos del lenguaje de programación y plataforma empleada, la programación en este tipo de microcontroladores conlleva una gran fase de aprendizaje con una curva de dificultad elevada.

En lo que se refiere a los controles implementados, el control de posición se ha diseñado obteniendo unos resultados finales buenos, puesto que la pinza es capaz de alcanzar las posiciones que se le determinen implementando únicamente un control PD. En cuanto al control de velocidad, no se han podido determinar las velocidades en cada instante debido a que el potenciómetro que obtiene la posición del motor presenta un elevado ruido por lo que se ha tenido que comprobar que, para un valor de acción de control constante aplicado sobre el motor, este se mueve prácticamente a la misma velocidad en todo su recorrido (sin tener en cuenta la fase de arranque). Finalmente, el control de fuerza diseñado no se ha podido implementar de manera efectiva sobre la pinza debido a que los sensores de fuerza que realimentan la fuerza ejercida presentan grandes ruidos y errores tal y como se ha comentado.

Con el control de fuerza que se ha podido implementar, se comprueba que el motor de la pinza permite sujetar objetos con pesos comprendidos entre 0 y unos 400 gramos, algo por debajo de los 500 gramos planteados en los objetivos de este proyecto.

Finalmente, considero que el prototipo desarrollado, cumple con las especificaciones y objetivos planteados inicialmente, aunque este presenta gran cantidad de posibles mejoras o adiciones las cuales se comentan en el siguiente apartado.

6.2. Mejoras y trabajos futuros

Existen una gran cantidad de posibles mejoras que se pueden realizar sobre el prototipo de la pinza para conseguir un producto más cercano a su comercialización, entre los cuales, en los siguientes puntos, se destacan los más importantes:

- En cuanto a los materiales empleados, se podrían emplear en vez de PLA otros plásticos más resistentes, o directamente fabricar esta estructura con algún tipo de metal ligero como podría ser el titanio.
- Puesto que la pinza está pensada para ser llevada por una persona, sería conveniente que la caja de componentes electrónicos de la pinza ocupara el mínimo espacio posible por lo que en trabajos futuros se debería hacer un circuito impreso (en vez de soldar los componentes de manera individual sobre una PCB) de toda la electrónica diseñada en este proyecto.
- Una mejora muy importante para realizar es el cambio del motor por uno con mayores prestaciones que permita ampliar el rango de pesos que la pinza es capaz de sujetar.
- La sustitución de los sensores de fuerza es obligatoria en un trabajo futuro, porque como se ha demostrado en este proyecto, este tipo de sensores no es compatible con la aplicación a desarrollar.
- Para la que la pinza pueda ser utilizada en el día a día habría que recubrir toda la estructura aislándola de agua y polvo, también para que esta fuera portátil habría que añadir una batería.
- En el caso de que se busque mejorar la experiencia de usuario, sería ideal que el usuario pudiera controlar la pinza con señales electromiográficas o encefalográficas, o en su defecto, desde una aplicación en su teléfono inteligente.
- Por último, sería interesante añadir otro motor para dotar a la pinza de un movimiento de muñeca.

6.3. Relación del trabajo desarrollado con los estudios cursados

Durante todo este proyecto, se observa que se han aplicado prácticamente todas las materias importantes aprendidas en el grado de “Electrónica Industrial y Automática”, entre las cuales se pueden destacar el diseño electrónico, el diseño de controladores, la programación de microcontroladores, o incluso el diseño CAD. De todas estas materias se han aprendido las bases en el grado, pero para la realización de este proyecto se ha tenido que ir más allá de lo aprendido en las clases, destacando entre otros, la programación avanzada del microcontrolador.

Cabe destacar que también se han tenido que aprender algunas técnicas que no se desarrollan a lo largo del grado, como puede ser el diseño, modelado y ensamblado de piezas 3D o la impresión 3D.

En el siguiente diagrama se muestra de manera gráfica aquello que se acaba de explicar en las anteriores líneas:

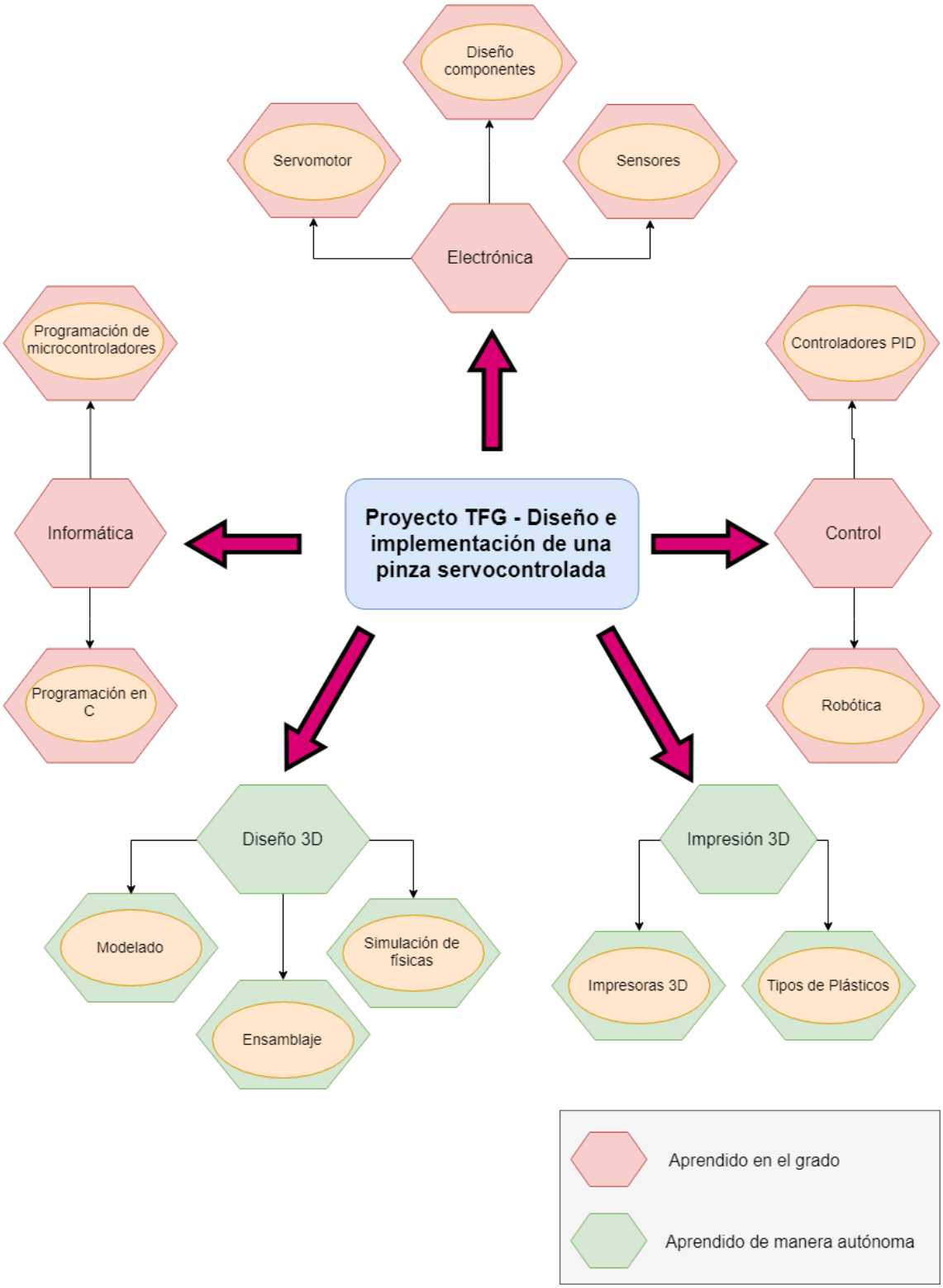


Figura 113. Diagrama relación del proyecto con los estudios cursados

BIBLIOGRAFÍA

[1] J. F. Camacho Muñoz, «Diseño, implementación y control de una pinza servo-controlada para un exoesqueleto,» UPV, Valencia, 2016.

[2] P. Ropero Giralda, «Desarrollo y Fabricación de Mano Robótica Controlada Inalámbricamente por Mano Humana,» Universidad de León, León, 2015.

[3] K. Norton, «Amputee Coalition,» Noviembre 2007. [En línea]. Available: www.amputee-coalition.org. [Último acceso: Abril 2019].

[4] «SynTouch,» SynToucn Inc, 2008-2019. [En línea]. Available: https://www.syntouchinc.com. [Último acceso: Abril 2019].

[5] J. E. Uellenddahl, «Amputee Coalition,» Noviembre 1998. [En línea]. Available: https://www.amputee-coalition.org/resources/spanish-materials-prosthetics-part-2/. [Último acceso: Abril 2019].

[6] K. Fu, Robotics: Control, Sensing and Intelligence, McGraw-Hill, 1987.

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

[8] À. Perles, «ARM Cortex-M práctico,» Valencia, UPV, 2018, p. 138.

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

DISEÑO E IMPLEMENTACIÓN DE UNA PINZA SERVOCONTROLADA

PRESUPUESTO

Autor: **Diego Antolí Soler**

Tutor: **Ranko Zotovic Stanisic**

Curso Académico: 2018-19

PRESUPUESTO

7. Presupuesto y coste económico del proyecto

7.1. Coste de los recursos humanos

El tiempo estimado para la realización de este proyecto han sido alrededor de 450 horas repartidas en 4 meses. Para calcular el precio de dichas horas, se ha tomado precio pagado por hora a un Ingeniero Junior (20 euros/hora).

Trabajador	euros/hora	Horas invertidas	**TOTAL**
Ingeniero Junior	20	450	**9000**

Tabla 12. Costes Recursos Humanos

7.2. Impresión 3D: piezas impresas y coste

En la siguiente tabla se recoge el resumen de todas las piezas que se han impreso en 3D junto con su coste unitario, comprobando que la impresión 3D con plástico PLA es barata.

Pieza	Cantidad	Precio Unitario	Tiempo Unitario	Total	Tiempo Total
Enganche engranaje libre	1	0,05	0,6	0,05	0,6
enganche engranaje motor	1	0,04	0,5	0,04	0,5
dedo superior	4	0,05	0,3	0,2	1,2
Inferior engranaje	1	0,35	1,5	0,35	1,5
dedos pinza	2	0,65	6	1,3	12
superior engranaje	1	0,28	2	0,28	2
soporte servomotor	1	0,66	4	0,66	4
laterales superiores	2	0,15	2	0,3	4
tapa superior	1	0,08	0,5	0,08	0,5
tapa botones	1	0,18	1	0,18	1
caja electrónica	1	1,48	8,5	1,48	8,5
enganche final de carrera	1	0,02	0,25	0,02	0,25
				4,94	**36,05**
				TOTAL COSTE	**TOTAL, HORAS**

Tabla 13. Costes Impresión 3D

7.3. Coste de materiales

En la siguiente tabla se recogen todos los materiales empleados en este proyecto junto con su precio y cantidad:

Producto	Cantidad	Precio	Total
STM32F429I	1	36	36
Servomotor Tower Pro MG959	1	54	54
Sensores de presión FSR 402 Interlink Electronics	8	7	56
Amplificadores Operacionales LM358p	4	0,402	1,608
PCB Baquelita	1	2,58	2,58
PCB Baquelita Pequeña	1	0,98	0,98
Cableado Dupont	1	4,06	4,06
Cableado 2x0,35mm2	4	0,48	1,92
Varillas roscadas de acero M3	4	0,4	1,6
Tuercas M3	14	0,09	1,26
Zócalo 16p y 8p	1	1,96	1,96
Finales de Carrera	2	1,1	2,2
Pulsadores	2	0,68	1,36
Regleta PCB	3	0,6	1,8
Puente en H - L293B dip16	1	5,05	5,05
Resistencias varias	1	1	1
		TOTAL	173,378

Tabla 14. Costes de los materiales

7.4. Coste total

La suma de todos los gastos se muestra en la siguiente tabla:

Coste Total Proyecto	Totales (euros)
Coste recursos humanos	9000
Costes impresión 3D	4,94
Costes materiales	173,38
TOTAL	**9178,32**

Tabla 15. Coste total del proyecto

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

DISEÑO E IMPLEMENTACIÓN DE UNA PINZA SERVOCONTROLADA

PLANOS

Autor: **Diego Antolí Soler**

Tutor: **Ranko Zotovic Stanisic**

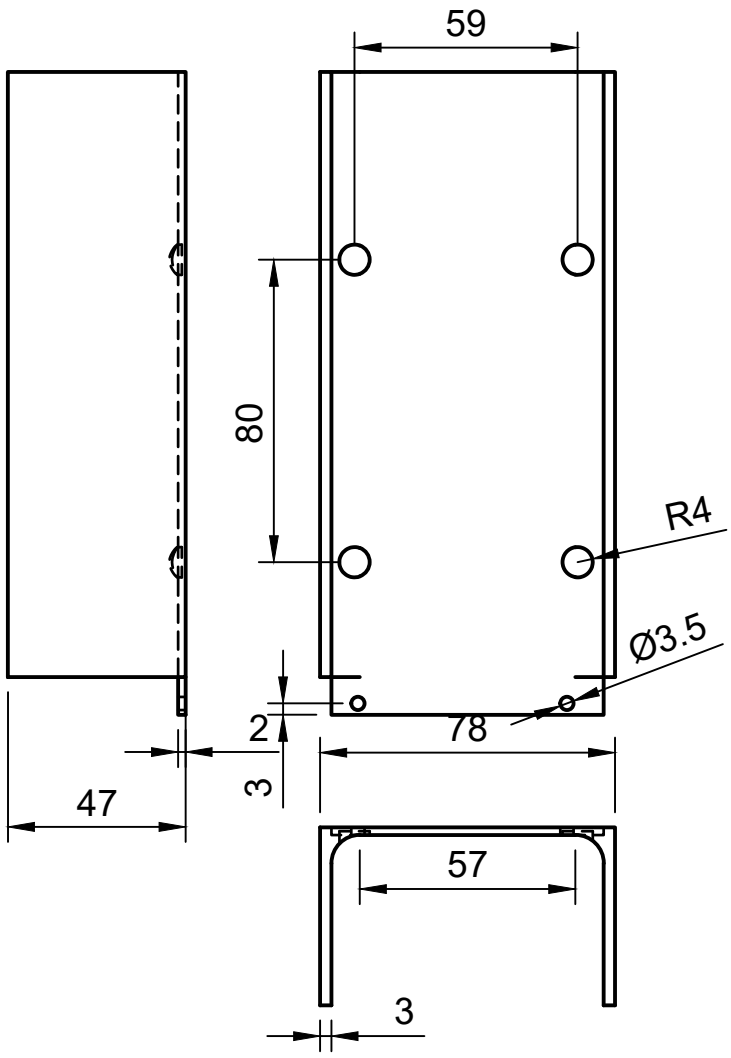
Curso Académico: 2018-19

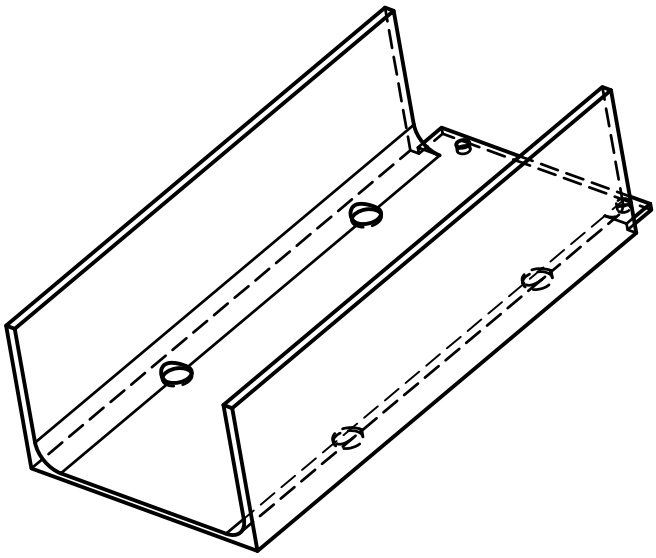
PLANOS

8. Planos acotados de las piezas diseñadas

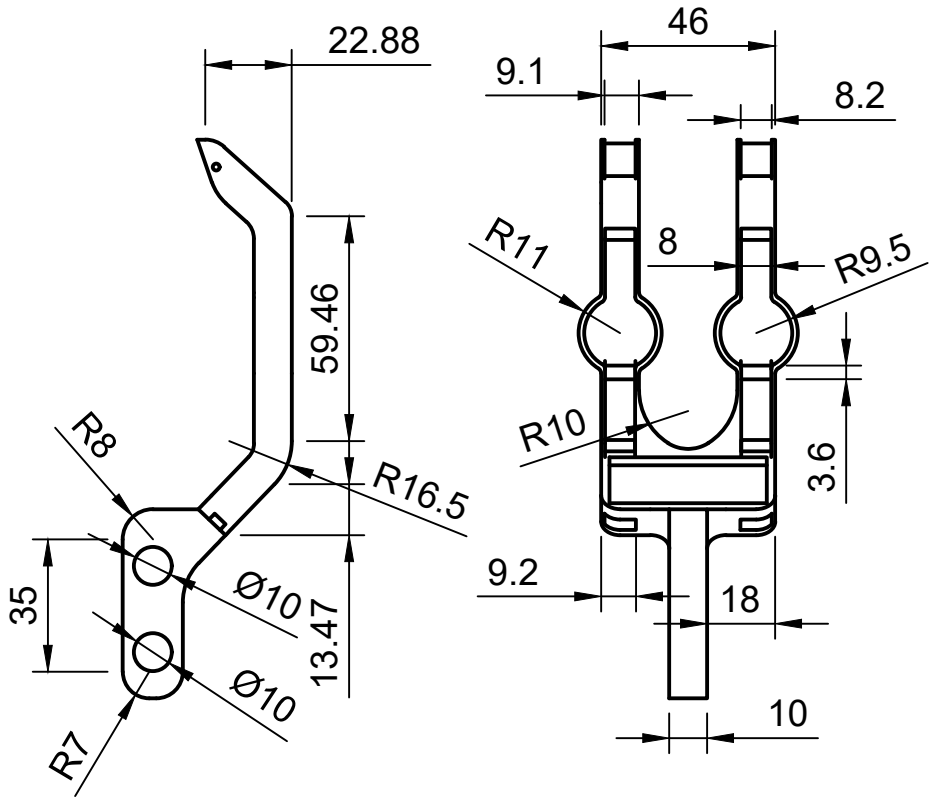
En las siguientes páginas se muestran los planos acotados de todas las piezas que se han diseñado para poder construir la pinza robótica. Estos se muestran en el siguiente orden:

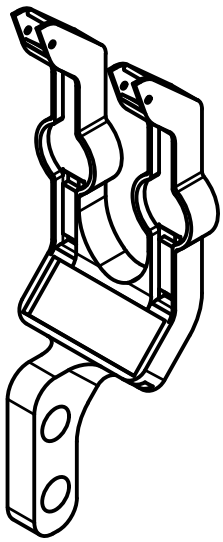
Plano 1:	Caja Electrónica
Plano 2:	Estructura Dedos
Plano 3:	Soporte Sensor Superior
Plano 4:	Eje Engranaje Libre
Plano 5:	Eje Engranaje Motor
Plano 6:	Eje Final de Carrera
Plano 7:	Cara Superior de Unión
Plano 8:	Cara Inferior de Unión
Plano 9:	Laterales Superiores
Plano 10:	Tapa Botones
Plano 11:	Tapa Superior



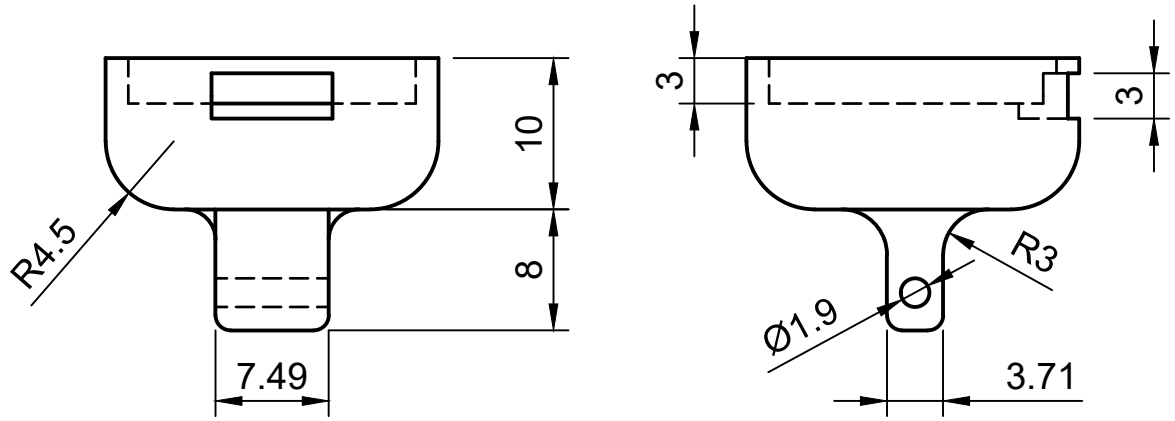


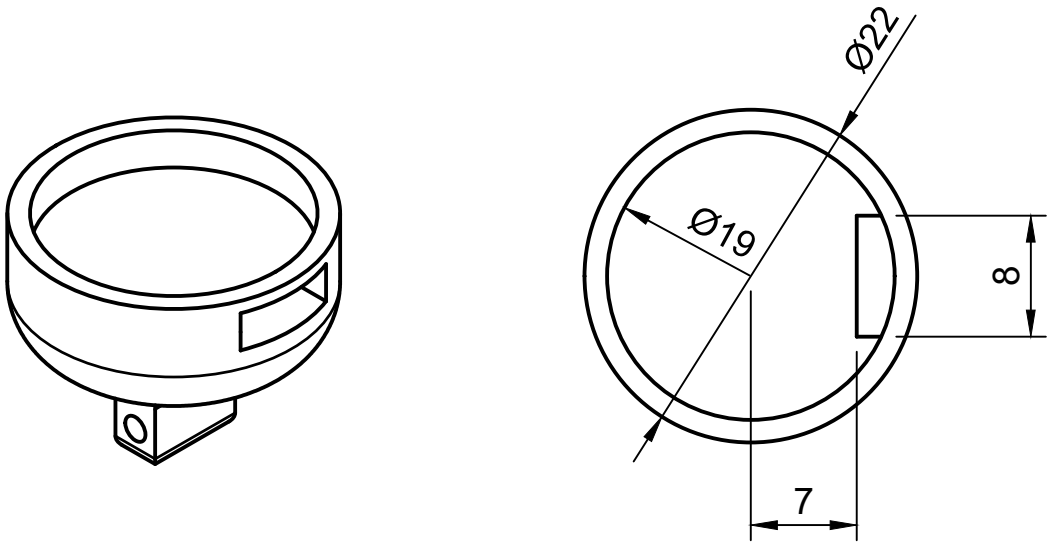
Scale 1:5		Created by Diego Antoli 30/05/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title: Caja Electrónica	DWG No. C.01
			Rev. 8 / Date of issue / Sheet 87



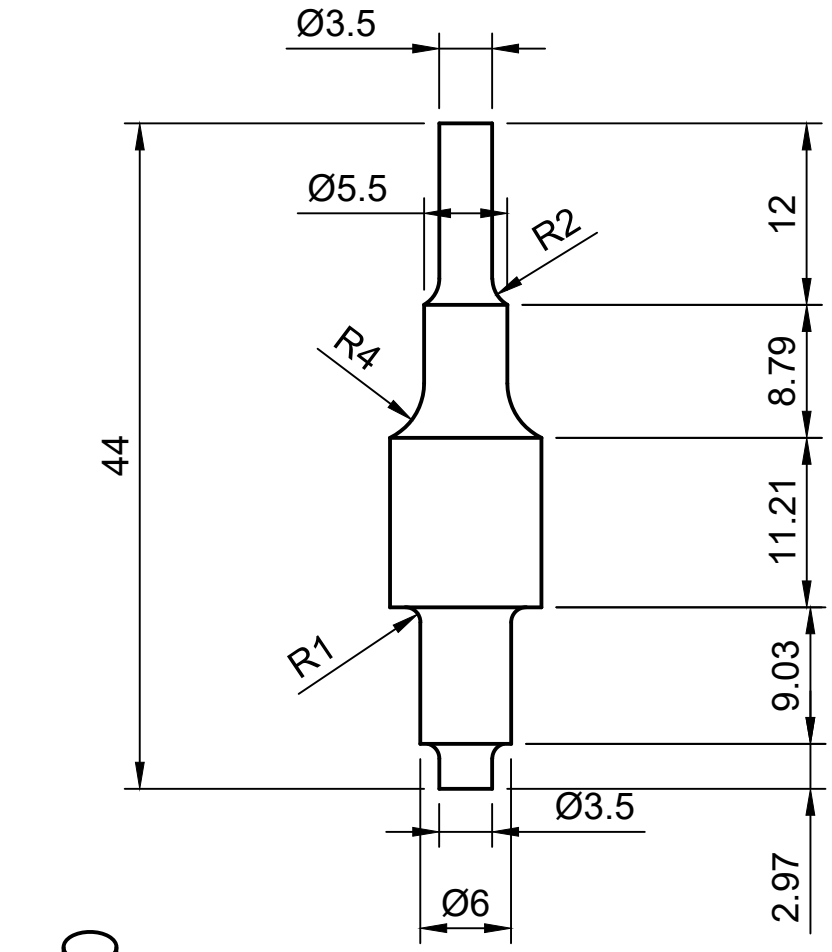


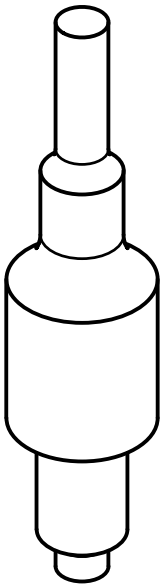
Scale 1:2		Created by Diego Antoli 30/05/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title Estructura Dedos	DWG No. E.01
			Rev. 25 / Date of issue / Sheet 88

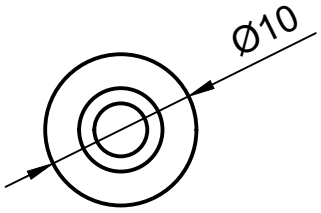




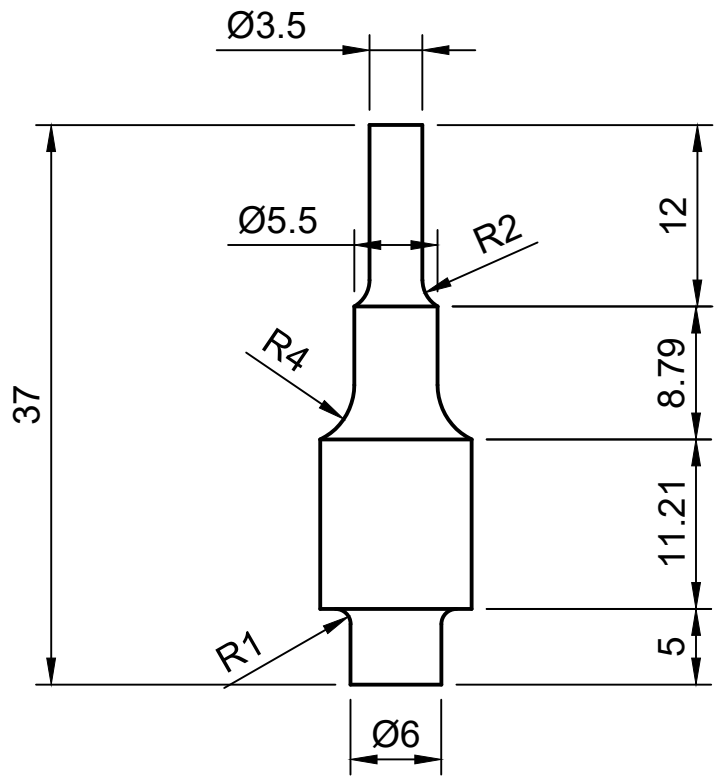
Scale. 2:1		Created by Diego Antoli 31/05/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title Soporte Sensor Superior	DWG No. E.02
			Rev. 10 / Date of issue / Sheet 89

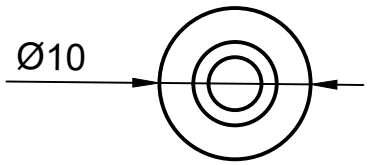


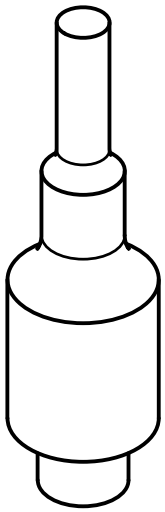




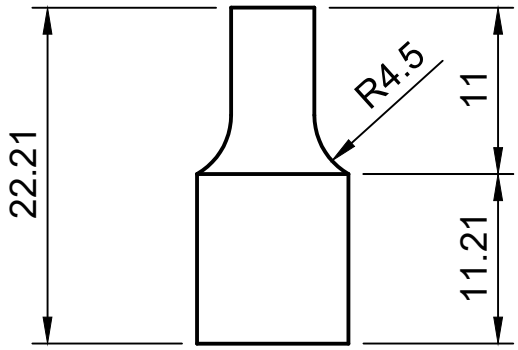
Scale 2:1		Created by Diego Antoli 31/05/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title Eje engranaje libre	DWG No. E.03
			Rev. 5 / Date of issue / Sheet 90

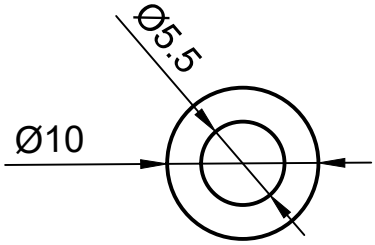


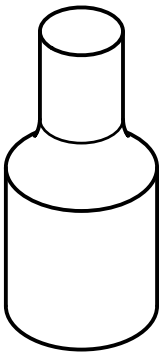




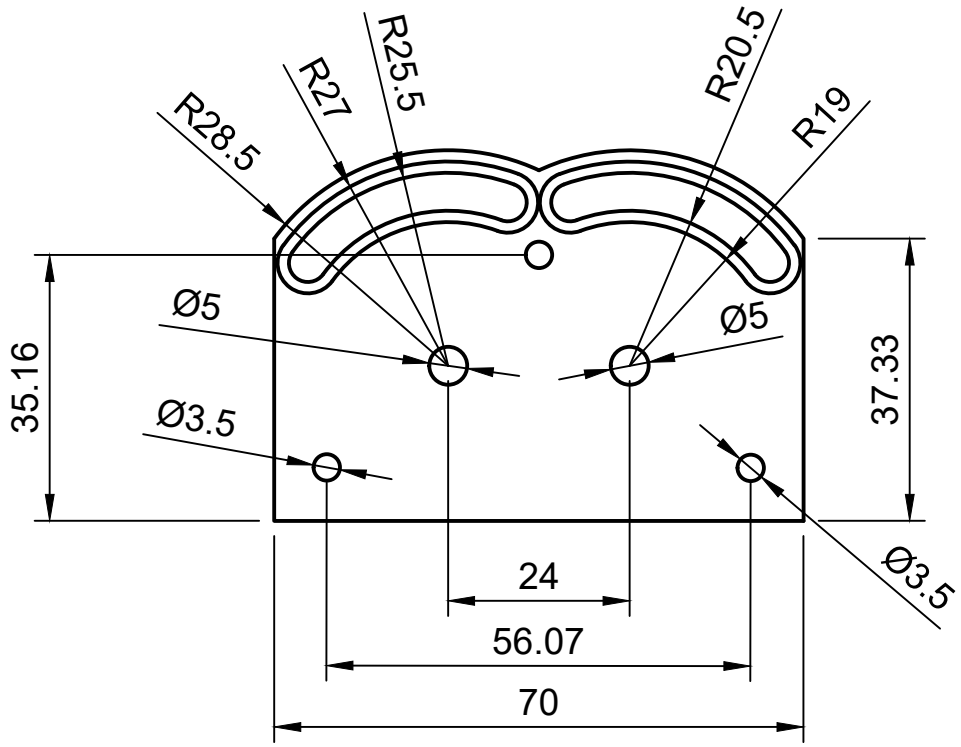
Scale 2:1		Created by Diego Antoli 31/05/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title: Eje engranaje Motor	DWG No.: E.04
			Rev. 4 / Date of issue / Sheet 91

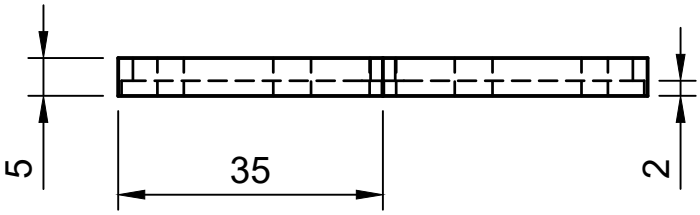


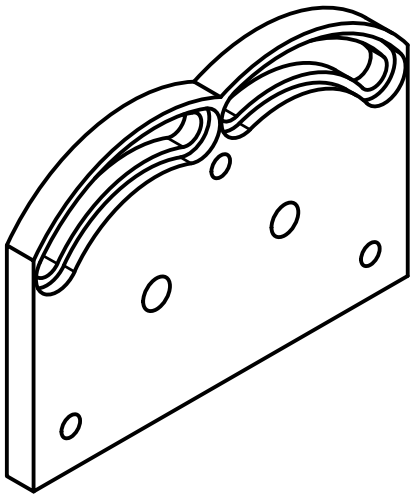




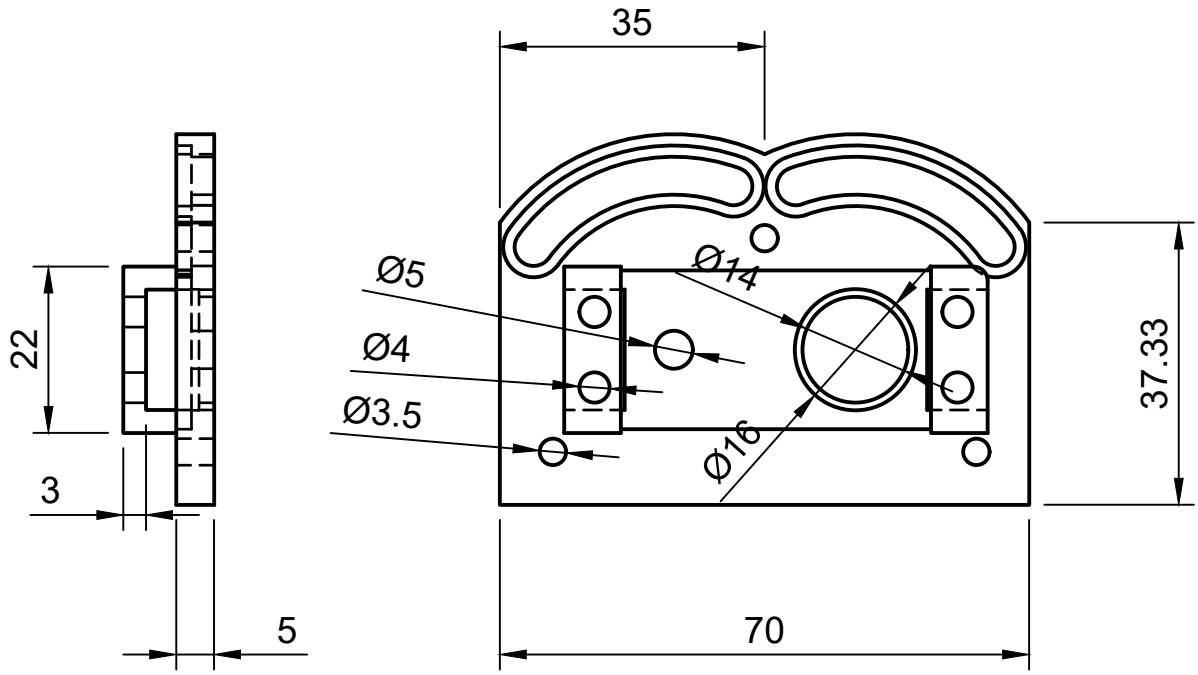
Scale 2:1		Created by Diego Antoli 31/05/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title Eje final de carrera	DWG No. E.05
			Rev. 4 / Date of issue / Sheet 92

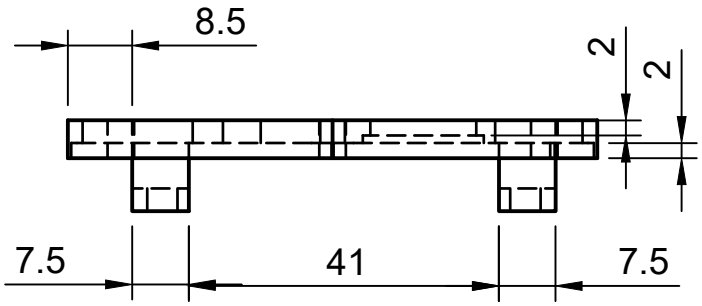


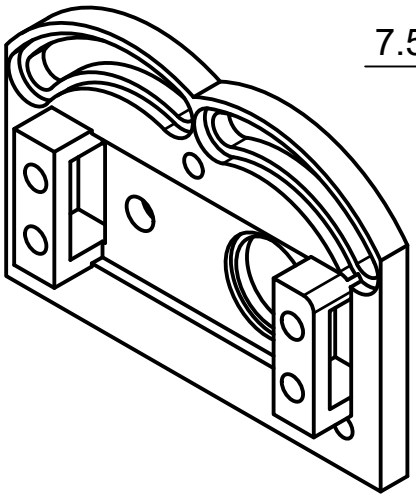




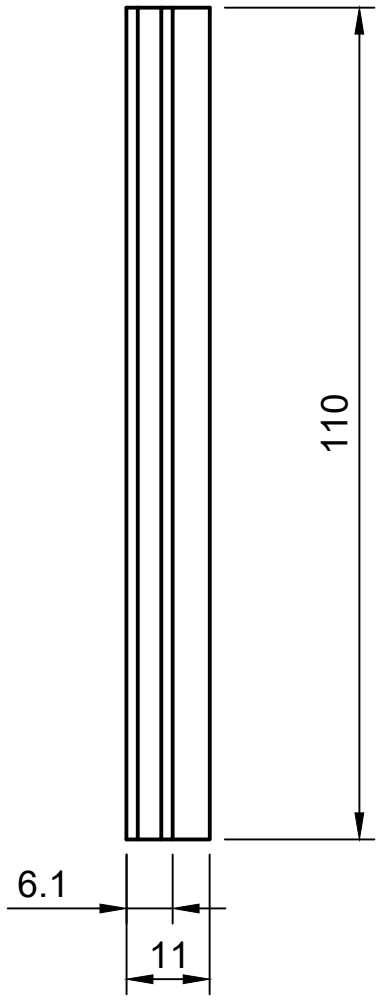
Scale 1:1		Created by Diego Antoli 31/05/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title: Cara superior de unión	DWG No.: E.06
			Rev. 9 / Date of issue / Sheet 93

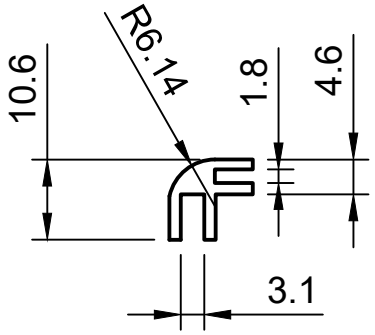




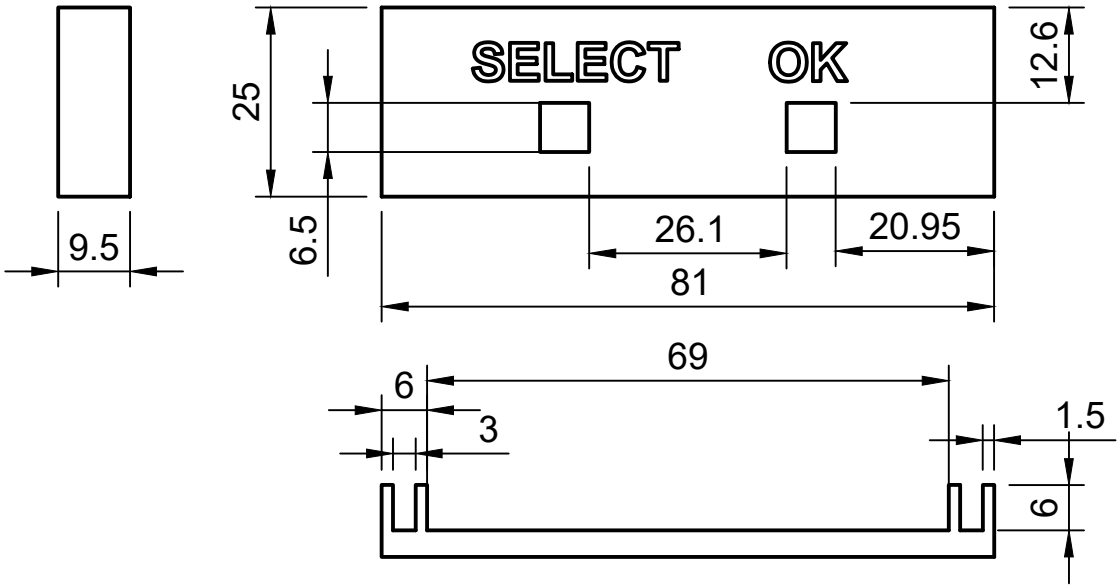


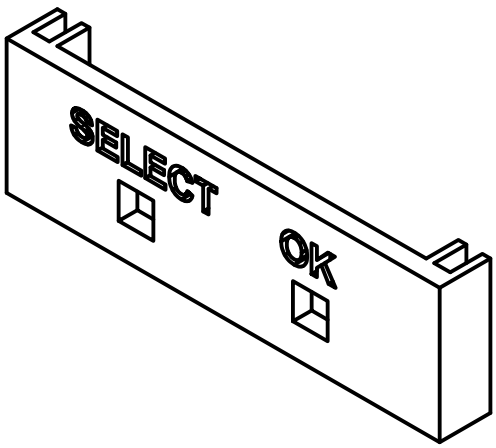
Scale 1:1		Created by Diego Antoli 31/05/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title Cara inferior de unión	DWG No. E.07
			Rev. 11 / Date of issue / Sheet 94



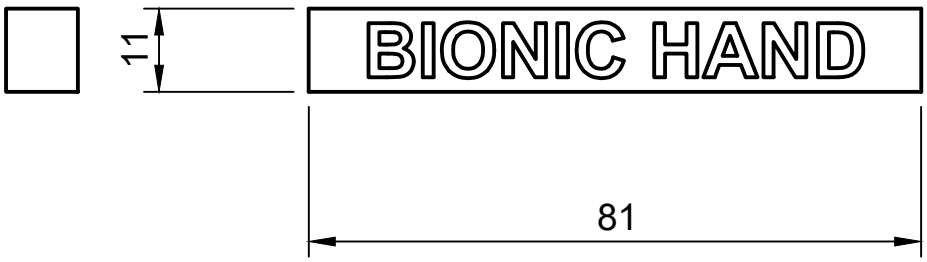


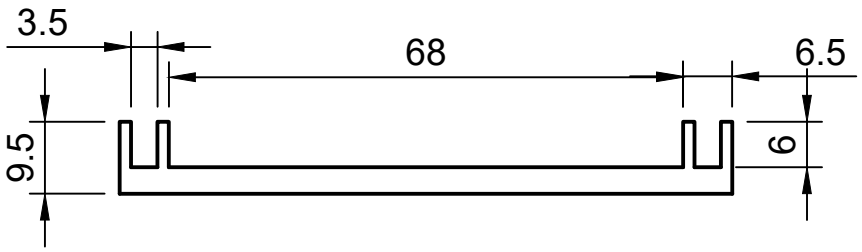
Scale 1:1		Created by Diego Antoli 31/05/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title Laterales superiores	DWG No. C.02
		Rev. 4	Date of issue
			Sheet 95

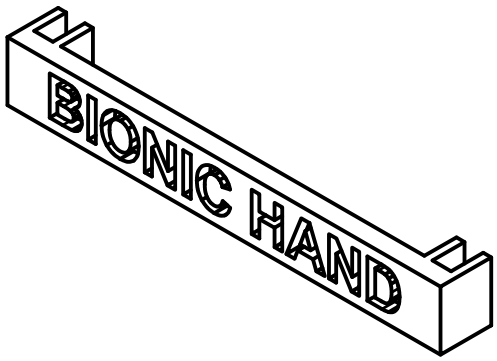




Scale 1:1		Created by Diego Antoli 31/05/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title: Tapa botones	DWG No.: C.03
			Rev. 5 / Date of issue / Sheet 96







Scale 1:1		Created by Diego Antoli 31/05/2019	Approved by
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA		Document type	Document status
		Title Tapa superior	DWG No. C.04
			Rev. 6 / Date of issue / Sheet 97

ANEXOS

Anexo I: Código programación microcontrolador

En este anexo se muestran todas las funciones que se han creado para el proyecto:

Main.c

```
/*
    El main básicamente es la interfaz donde en función de las opciones que se elijan sobre la pantalla, se irá a una función u otra
 */

/* Include core modules */
#include "stm32f4xx.h"
/* Include my libraries here */
#include "defines.h"
#include "tm_stm32f4_ili9341.h"
#include "tm_stm32f4_fonts.h"
#include "tm_stm32f4_delay.h"
#include <stdio.h>
#include <math.h>

/* Includes Módulos */
#include "InicializacionEntradasAnalogicas.h"
#include "InicializacionMotor.h"
#include "InicializacionPWM.h"
#include "InicializacionInterrupcion.h"
#include "ConfiguracionesPantalla.h"
#include "ControlPosicion.h"
#include "ControlVelocidad.h"
#include "ControlFuerza.h"


/* Variables */

    //Variables entradas analógicas pertenecientes a ADC1
    int32_t PC3 = 0;
    int32_t PC0 = 0;
    int32_t PC1 = 0;

    //Variables entradas analógicas pertenecientes a ADC3
    int32_t PF3 = 0;
    int32_t PF4 = 0;
    int32_t PF5 = 0;
    int32_t PF6 = 0;

    //Variables entradas analógicas pertenecientes a ADC2
    int32_t PB0 = 0;
    int32_t PB1 = 0;

    //Conversor de Bits a Voltios de los ADC's
    float voltiosPB1 = 0;
    float voltiosPC0 = 0;
    float voltiosPC1 = 0;
    float voltiosPF3 = 0;
    float voltiosPC3 = 0;
    float voltiosPF4 = 0;
    float voltiosPF5 = 0;
    float voltiosPF6 = 0;

    //Conversor de Bits a Grados del ADC del Potenciómetro
    uint8_t Pos_Grados = 0;

    // Variables impresión en pantalla LCD
    double uwVoltage =125.3;
    uint8_t aTextBuffer[50];


    volatile uint8_t ir_Pos_inicial=0;


    // Paro de los FC cuando molestan
    volatile uint8_t off_FC=0;

    //variables en pruebas
    float N=0;
    float masa=0;

    void led_inicializar(void){     //Esta función inicializa el Led Rojo y el Led Verde

    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOG, ENABLE);


    GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_13|GPIO_Pin_14;
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
```

```
83.
84.     GPIO_Init(GPIOG, &GPIO_InitStructure);
85. }
86.
87. void Configuracion_Pulsadores(void){
88.     GPIO_InitTypeDef puls;
89.     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
90.
91.     puls.GPIO_Mode = GPIO_Mode_IN;
92.     puls.GPIO_PuPd = GPIO_PuPd_NOPULL;
93.     puls.GPIO_Pin = GPIO_Pin_10|GPIO_Pin_12;
94.     GPIO_Init(GPIOB, &puls);
95. }
96.
97. int main(void) {
98.
99.     /* Inicialización Entradas Analógicas ADC1, ADC2 Y ADC3 */
100.    analog_inicializar();
101.
102.    /* Inicialización Salidas Digitales Motor */
103.    InicializacionMotor();
104.
105.    /* Inicialización señal PWM   */
106.    InicializacionGPIOA3_PWM();
107.    TM_TIMER_Init();
108.
109.    /* Inicialización Interrupción pin PD0 */
110.    Configure_PA0();
111.    Configure_PD1();
112.
113.    /* Inicialización Led Verde y Led Rojo */
114.    led_inicializar();
115.    Configuracion_Pulsadores();
116.
117.    /* Inicializar pantalla LCD */
118.    ConfiguracionPantalla();
119.
120. uint8_t boton_siguiente=0;
121. uint8_t boton_ok=0;
122. uint8_t selector=0;
123. uint8_t selector_vel=0;
124. uint8_t selector_fuerza=0;
125. uint16_t baja_linea=115;    //esto ubica el puntero rojo en las diferentes líneas
126. uint8_t pulsador_ant=0;     //esto evita los rebotes de los botones
127. uint8_t salir_interfaz1_pos; //se encarga de salir del bucle de la interfaz de posición 1
128. uint8_t salir_interfaz2_pos;    //Se encarga de salir del bucle de la interfaz de posición -> exterior
129. uint8_t salir_interfaz1_vel;    //se encarga de salir del buble de la interfaz de velocidad 1
130. uint8_t salir_interfaz2_vel;    //se encarga de salir del bucle de la interfaz de velocidad -> cierre
131. uint8_t salir_interfaz1_fuerza; //se encarga de salir del buble de la interfaz de fuerza
132. uint8_t salir_interfaz2_fuerza; //se encarga de salir del buble de la interfaz de fuerza
133.
134.    while (1) {
135.
136.        boton_siguiente = GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_10);
137.        boton_ok = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12);
138.
139.        if(boton_siguiente==1&&pulsador_ant==0){
140.            if(baja_linea>=240) {
141.                TM_ILI9341_DrawFilledCircle(10,235, 5, ILI9341_COLOR_CYAN);
142.                baja_linea=115;
143.                selector=0;
144.            }
145.
146.            TM_ILI9341_DrawFilledCircle(10,baja_linea-30, 5, ILI9341_COLOR_CYAN);
147.            TM_ILI9341_DrawFilledCircle(10,baja_linea, 5, ILI9341_COLOR_RED);
148.            baja_linea= baja_linea+30;
149.            selector++;
150.
151.        }
152.        pulsador_ant=boton_siguiente;
153.
154.        if(boton_ok==1){
155.            switch(selector){
156.                case 1: //Abrir Pinza
157.                                ControlVelocidad(0, 50, 0);
158.                                TM_ILI9341_Puts(170, 110,"OK", &TM_Font_11x18, ILI9341_COLOR_GREEN, ILI9341_COLOR_CYAN);
159.                                Delayms(800);
160.                                TM_ILI9341_Puts(170, 110,"  ", &TM_Font_11x18, ILI9341_COLOR_GREEN, ILI9341_COLOR_CYAN);
161.                                break;
162.
163.                case 2: //Cerrar Pinza
164.                                ControlVelocidad(1, 50, 0);
165.                                TM_ILI9341_Puts(170, 140,"OK", &TM_Font_11x18, ILI9341_COLOR_GREEN, ILI9341_COLOR_CYAN);
166.                                Delayms(800);
167.                                TM_ILI9341_Puts(170, 140,"  ", &TM_Font_11x18, ILI9341_COLOR_GREEN, ILI9341_COLOR_CYAN);
168.                                break;
169.
170.                case 3: //Control Posición
171.                                // 1ª Interfaz
172.                                TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
173.                                TM_ILI9341_Puts(0, 0, "  BIONIC HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK, ILI9341_COLOR_RED);
174.                                TM_ILI9341_Puts(0, 50," Control de Posición
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
175.                                TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
176.                                TM_ILI9341_Puts(5, 100,"Seleccionar el tipo de              desplazamiento:
    ", &TM_Font_7x10,ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
177.                                TM_ILI9341_Puts(0, 140,"  Interior ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
178.                                TM_ILI9341_Puts(0, 170,"  Exterior ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
179.                                TM_ILI9341_Puts(0, 200,"  Atras ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
```

```
180.
181.                                 //Selector
182.                                 baja_linea = 145;
183.                                 selector = 0;
184.                                 salir_interfaz1_pos=0;
185.                                 pulsador_ant = 0;
186.
187.
188.                                 while(salir_interfaz1_pos==0){
189.                                     boton_siguiente = GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_10);
190.                                     boton_ok = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12);
191.
192.                                     if(boton_siguiente==1&&pulsador_ant==0){
193.                                         if(baja_linea>=210) {
194.                                             TM_ILI9341_Puts(0, 200,"  Atras
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
195.                                             baja_linea=145;
196.                                             selector=0;
197.                                         }
198.
199.                                         TM_ILI9341_DrawFilledCircle(10,baja_linea-30, 5, ILI9341_COLOR_CYAN);
200.                                         TM_ILI9341_DrawFilledCircle(10,baja_linea, 5, ILI9341_COLOR_RED);
201.                                         baja_linea= baja_linea+30;
202.                                         selector++;
203.                                     }
204.                                     pulsador_ant=boton_siguiente;
205.
206.                                     if(boton_ok==1){
207.                                         switch(selector){
208.                                             case 1: //Interior
209.                                                         // 2ª Interfaz
210.                                                 TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
211.                                                 TM_ILI9341_Puts(0, 0, "  BIONIC
    HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK,ILI9341_COLOR_RED); // Titulo
212.                                                 TM_ILI9341_Puts(0, 50," Control de Posicion        Interior
    ", &TM_Font_11x18,ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
213.                                                 TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
214.                                                 TM_ILI9341_Puts(5, 100,"Seleccionar distancia              a desplazar:
    ",&TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
215.                                                 TM_ILI9341_Puts(0, 140,"  8 cm
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
216.                                                 TM_ILI9341_Puts(0, 170,"  11 cm
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
217.                                                 TM_ILI9341_Puts(0, 200,"  13 cm
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
218.                                                 TM_ILI9341_Puts(0, 230,"  Atras
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
219.
220.                                                 //Selector
221.                                                 baja_linea = 145;
222.                                                 selector = 0;
223.                                                 salir_interfaz2_pos=0;
224.                                                 pulsador_ant = 0;
225.
226.                                                 while(salir_interfaz2_pos==0){
227.                                                     boton_siguiente = GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_10);
228.                                                     boton_ok = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12);
229.
230.                                                     if(boton_siguiente==1&&pulsador_ant==0){
231.                                                         if(baja_linea>=240) {
232.                                                             TM_ILI9341_DrawFilledCircle(10,235, 5, ILI9341_COLOR_CYAN);
233.                                                             baja_linea=145;
234.                                                             selector=0;
235.                                                         }
236.
237.                                                         TM_ILI9341_DrawFilledCircle(10,baja_linea-
    30, 5, ILI9341_COLOR_CYAN);
238.                                                         TM_ILI9341_DrawFilledCircle(10,baja_linea, 5, ILI9341_COLOR_RED);
239.                                                         baja_linea= baja_linea+30;
240.                                                         selector++;
241.                                                     }
242.                                                     pulsador_ant=boton_siguiente;
243.
244.                                                     if(boton_ok==1){
245.                                                         switch(selector){
246.                                                             case 1:  // 8 cm
247.                                                                                 GeneradorTrayectoria(8,0);
248.                                                                                 break;
249.                                                             case 2:  // 11 cm
250.                                                                                 GeneradorTrayectoria(11,0);
251.                                                                                 break;
252.                                                             case 3:  // 13 cm
253.                                                                                 GeneradorTrayectoria(13, 0);
254.                                                                                 break;
255.                                                             case 4: //salir del menú
256.                                                                                 salir_interfaz2_pos=1;
257.                                                                                 break;
258.                                                         }
259.                                                     }
260.                                                 }
261.
262.                                                 //Aquí se vuelve a imprimir la interfaz del menú anterior
263.                                                     TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
264.                                                     TM_ILI9341_Puts(0, 0, "  BIONIC
    HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK,ILI9341_COLOR_RED); // Titulo
265.                                                     TM_ILI9341_Puts(0, 50," Control de Posicion
    ", &TM_Font_11x18,ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
266.                                                     TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
```

```
267.                                                    TM_ILI9341_Puts(5, 100,"Seleccionar el tipo
    de              desplazamiento: ",&TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
268.                                                    TM_ILI9341_Puts(0, 140,"  Interior
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
269.                                                    TM_ILI9341_Puts(0, 170,"  Exterior
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
270.                                                    TM_ILI9341_Puts(0, 200,"  Atras
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
271.                                                    break;
272.
273.                                                case 2: //Exterior
274.
275.                                                // 2ª Interfaz
276.                                                    TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
277.                                                    TM_ILI9341_Puts(0, 0, "  BIONIC
    HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK,ILI9341_COLOR_RED); // Titulo
278.                                                    TM_ILI9341_Puts(0, 50," Control de
    Posicion        Exterior", &TM_Font_11x18,ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
279.                                                    TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
280.                                                    TM_ILI9341_Puts(5, 100,"Seleccionar distancia              a desplazar:
    ",&TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
281.                                                    TM_ILI9341_Puts(0, 140,"  1 cm
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
282.                                                    TM_ILI9341_Puts(0, 170,"  4 cm
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
283.                                                    TM_ILI9341_Puts(0, 200,"  8.55 cm
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
284.                                                    TM_ILI9341_Puts(0, 230,"  Atras
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
285.
286.                                                    //Selector
287.                                                    baja_linea = 145;
288.                                                    selector = 0;
289.                                                    salir_interfaz2_pos=0;
290.                                                    pulsador_ant = 0;
291.
292.                                                    while(salir_interfaz2_pos==0){
293.                                                        boton_siguiente = GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_10);
294.                                                        boton_ok = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12);
295.
296.                                                        if(boton_siguiente==1&&pulsador_ant==0){
297.                                                            if(baja_linea>=240) {
298.                                                                TM_ILI9341_DrawFilledCircle(10,235, 5, ILI9341_COLOR_CYAN);
299.                                                                baja_linea=145;
300.                                                                selector=0;
301.                                                            }
302.
303.                                                            TM_ILI9341_DrawFilledCircle(10,baja_linea-
    30, 5, ILI9341_COLOR_CYAN);
304.                                                            TM_ILI9341_DrawFilledCircle(10,baja_linea, 5, ILI9341_COLOR_RED);
305.                                                            baja_linea= baja_linea+30;
306.                                                            selector++;
307.                                                        }
308.                                                        pulsador_ant=boton_siguiente;
309.
310.                                                        if(boton_ok==1){
311.                                                            switch(selector){
312.                                                                case 1:  // 1 cm
313.                                                                                GeneradorTrayectoria(1,1);  //equivalente a
    2.29 grados
314.                                                                                break;
315.
316.                                                                case 2:  // 4cm
317.                                                                                GeneradorTrayectoria(4,1); //equivalente a
    9.20 grados
318.                                                                                break;
319.                                                                case 3:  // 8.55 cm
320.                                                                                GeneradorTrayectoria(8.55, 1); //equivalent
    e a 20 grados
321.                                                                                break;
322.                                                                case 4: //salir del menú
323.                                                                                salir_interfaz2_pos=1; break;
324.                                                            }
325.                                                        }
326.                                                    }
327.
328.                                                    //Aquí se vuelve a imprimir la interfaz del menú anterior
329.                                                        TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
330.                                                        TM_ILI9341_Puts(0, 0, "  BIONIC
    HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK,ILI9341_COLOR_RED); // Titulo
331.                                                        TM_ILI9341_Puts(0, 50," Control de Posicion
    ", &TM_Font_11x18,ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
332.                                                        TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
333.                                                        TM_ILI9341_Puts(5, 100,"Seleccionar el tipo
    de              desplazamiento: ",&TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
334.                                                        TM_ILI9341_Puts(0, 140,"  Interior
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
335.                                                        TM_ILI9341_Puts(0, 170,"  Exterior
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
336.                                                        TM_ILI9341_Puts(0, 200,"  Atras
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
337.                                                    break;
338.                                                case 3:  salir_interfaz1_pos=1; break;
339.                                            }
340.                                        }
341.                                        Delayms(1);
342.                                    }
343.                                    ConfiguracionPantalla();
344.                                    selector=0;
                                        baja_linea=115;
```

```
                                break;

                    case 4: //Control Velocidad
                                TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
                                TM_ILI9341_Puts(0, 0, "  BIONIC
HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK, ILI9341_COLOR_RED); // Titulo
                                TM_ILI9341_Puts(0, 50," Control de Velocidad
", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
                                TM_ILI9341_Puts(5, 100,"Seleccionar la direccion              del desplazamiento:
", &TM_Font_7x10,ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                TM_ILI9341_Puts(0, 140,"  Apertura ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                TM_ILI9341_Puts(0, 170,"  Cierre ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                TM_ILI9341_Puts(0, 200,"  Atras ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);


                                //Selector velocidad
                                baja_linea = 145;
                                selector_vel = 0;
                                salir_interfaz1_vel=0;
                                pulsador_ant = 0;

                                while(salir_interfaz1_vel==0){
                                    boton_siguiente = GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_10);
                                    boton_ok = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12);

                                    if(boton_siguiente==1&&pulsador_ant==0){
                                        if(baja_linea>=210) {
                                            TM_ILI9341_Puts(0, 200,"  Atras
", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                            baja_linea=145;
                                            selector_vel=0;
                                        }

                                        TM_ILI9341_DrawFilledCircle(10,baja_linea-30, 5, ILI9341_COLOR_CYAN);
                                        TM_ILI9341_DrawFilledCircle(10,baja_linea, 5, ILI9341_COLOR_RED);
                                        baja_linea= baja_linea+30;
                                        selector_vel++;
                                    }
                                    pulsador_ant=boton_siguiente;

                                    if(boton_ok==1){
                                        switch(selector_vel){
                                            case 1: //Apertura
                                                        // 2ª Interfaz
                                                        TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 0, "  BIONIC
HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK,ILI9341_COLOR_RED); // Titulo
                                                        TM_ILI9341_Puts(0, 50," Control de Velocidad
", &TM_Font_11x18,ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
                                                        TM_ILI9341_Puts(5, 100,"Seleccionar % de Velocidad          de
Apertura: ",&TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 140,"  25 %
", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 170,"  50 %
", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 200,"  75 %
", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 230,"  100 %
", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 260,"  Atras
", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);

                                                        //Selector
                                                        baja_linea = 145;
                                                        selector_vel = 0;
                                                        salir_interfaz2_vel=0;
                                                        pulsador_ant = 0;

                                                        while(salir_interfaz2_vel==0){
                                                            boton_siguiente = GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_10);
                                                            boton_ok = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12);

                                                            if(boton_siguiente==1&&pulsador_ant==0){
                                                                if(baja_linea>=270) {
                                                                    TM_ILI9341_DrawFilledCircle(10,265, 5, ILI9341_COLOR_CYAN);
                                                                    baja_linea=145;
                                                                    selector_vel=0;
                                                                }

                                                                TM_ILI9341_DrawFilledCircle(10,baja_linea-
30, 5, ILI9341_COLOR_CYAN);
                                                                TM_ILI9341_DrawFilledCircle(10,baja_linea, 5, ILI9341_COLOR_RED
);
                                                                baja_linea= baja_linea+30;
                                                                selector_vel++;
                                                            }
                                                            pulsador_ant=boton_siguiente;

                                                            if(boton_ok==1){
                                                                switch(selector_vel){
                                                                    case 1:  // 25 %
                                                                                ControlVelocidad(0,25, 1);
                                                                                break;

                                                                    case 2:  // 50 %
                                                                                ControlVelocidad(0,50, 1);
                                                                                break;
```

```
                                                                case 3:  // 75 %
                                                                        ControlVelocidad(0,75, 1);
                                                                        break;

                                                                case 4:     //100 %
                                                                        ControlVelocidad(0,100, 1);
                                                                        break;

                                                                case 5: //salir del menú
                                                                        salir_interfaz2_vel=1; break;
                                                        }
                                                }
                                        }
                                        TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
                                        TM_ILI9341_Puts(0, 0, "  BIONIC HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK, ILI9341_COLOR_RED);// Título
                                        TM_ILI9341_Puts(0, 50," Control de Velocidad ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                        TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
                                        TM_ILI9341_Puts(5, 100,"Seleccionar la direccion              del desplazamiento:",&TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                        TM_ILI9341_Puts(0, 140,"  Apertura ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                        TM_ILI9341_Puts(0, 170,"  Cierre ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                        TM_ILI9341_Puts(0, 200,"  Atras ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                        break;

                                        case 2: //Cierre
                                                // 2ª Interfaz
                                                TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
                                                TM_ILI9341_Puts(0, 0, "  BIONIC HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK,ILI9341_COLOR_RED); // Título
                                                TM_ILI9341_Puts(0, 50," Control de Velocidad ", &TM_Font_11x18,ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                                TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
                                                TM_ILI9341_Puts(5, 100,"Seleccionar % de Velocidad           de Cierre:",&TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                                TM_ILI9341_Puts(0, 140,"  25 % ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                TM_ILI9341_Puts(0, 170,"  50 % ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                TM_ILI9341_Puts(0, 200,"  75 % ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                TM_ILI9341_Puts(0, 230,"  100 % ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                TM_ILI9341_Puts(0, 260,"  Atras ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);

                                                //Selector
                                                baja_linea = 145;
                                                selector_vel = 0;
                                                salir_interfaz2_vel=0;
                                                pulsador_ant = 0;

                                                while(salir_interfaz2_vel==0){
                                                    boton_siguiente = GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_10);
                                                    boton_ok = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12);

                                                    if(boton_siguiente==1&&pulsador_ant==0){
                                                        if(baja_linea>=270) {
                                                            TM_ILI9341_DrawFilledCircle(10,265, 5, ILI9341_COLOR_CYAN);
                                                            baja_linea=145;
                                                            selector_vel=0;
                                                        }

                                                        TM_ILI9341_DrawFilledCircle(10,baja_linea-30, 5, ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_DrawFilledCircle(10,baja_linea, 5, ILI9341_COLOR_RED);

                                                        baja_linea= baja_linea+30;
                                                        selector_vel++;
                                                    }
                                                    pulsador_ant=boton_siguiente;

                                                    if(boton_ok==1){
                                                        switch(selector_vel){
                                                            case 1:  // 25 %
                                                                    ControlVelocidad(1,25, 1);
                                                                    break;

                                                            case 2:  // 50 %
                                                                    ControlVelocidad(1,50, 1);
                                                                    break;

                                                            case 3:  // 75 %
                                                                    ControlVelocidad(1,75, 1);
                                                                    break;

                                                            case 4:     //100 %
                                                                    ControlVelocidad(1,100, 1);
                                                                    break;

                                                            case 5: //salir del menú
                                                                    salir_interfaz2_vel=1; break;
                                                        }
                                                    }
```

```
513.                                                }
514.                                            TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
515.                                            TM_ILI9341_Puts(0, 0, "  BIONIC
    HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK, ILI9341_COLOR_RED);// Titulo
516.                                            TM_ILI9341_Puts(0, 50," Control de Velocidad
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
517.                                            TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
518.                                            TM_ILI9341_Puts(5, 100,"Seleccionar la direccion             del desplazamiento:
    ",&TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
519.                                            TM_ILI9341_Puts(0, 140,"  Apertura
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
520.                                            TM_ILI9341_Puts(0, 170,"  Cierre
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
521.                                            TM_ILI9341_Puts(0, 200,"  Atras
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
522.                                            break;
523.
524.                                            case 3: //retroceder al menu anterior
525.                                                    salir_interfaz1_vel=1;
526.                                                    break;
527.                                        }
528.                                    }
529.                                    Delayms(1);
530.                                }
531.                            ConfiguracionPantalla();
532.                            selector=0;
533.                            baja_linea=115;
534.                             break;
535.
536.                case 5: // Control Fuerza
537.                                    TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
538.                                    TM_ILI9341_Puts(0, 0, "  BIONIC
    HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK, ILI9341_COLOR_RED); // Titulo
539.                                    TM_ILI9341_Puts(0, 50," Control de Fuerza
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
540.                                    TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
541.                                    TM_ILI9341_Puts(5, 100,"Seleccionar el tipo de agarre:
    ", &TM_Font_7x10, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
542.                                    TM_ILI9341_Puts(0, 140, "  Interno ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
543.                                    TM_ILI9341_Puts(0, 170, "  Externo ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
544.                                    TM_ILI9341_Puts(0, 200, "  Atras ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
545.
546.                                //Selector FUERZA
547.                                baja_linea = 145;
548.                                selector_fuerza = 0;
549.                                salir_interfaz1_fuerza=0;
550.                                pulsador_ant = 0;
551.
552.                                while(salir_interfaz1_fuerza==0){
553.                                    boton_siguiente = GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_10);
554.                                    boton_ok = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12);
555.
556.                                    if(boton_siguiente==1&&pulsador_ant==0){
557.                                        if(baja_linea>=210) {
558.                                            TM_ILI9341_Puts(0, 200,"  Atras
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
559.                                            baja_linea=145;
560.                                            selector_fuerza=0;
561.                                        }
562.
563.                                        TM_ILI9341_DrawFilledCircle(10,baja_linea-30, 5, ILI9341_COLOR_CYAN);
564.                                        TM_ILI9341_DrawFilledCircle(10,baja_linea, 5, ILI9341_COLOR_RED);
565.                                        baja_linea= baja_linea+30;
566.                                        selector_fuerza++;
567.                                    }
568.                                    pulsador_ant=boton_siguiente;
569.
570.                                    if(boton_ok==1){
571.                                        switch(selector_fuerza){
572.                                            case 1: //Fuerza con sensores internos
573.                                                    // 2ª Interfaz
574.                                                    TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
575.                                                    TM_ILI9341_Puts(0, 0, "  BIONIC
    HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK,ILI9341_COLOR_RED); // Titulo
576.                                                    TM_ILI9341_Puts(0, 50," Control de Fuerza
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
577.                                                    TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
578.                                                    TM_ILI9341_Puts(5, 100,"Seleccionar la fuerza a ejercer     con los
    sensores internos: ", &TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
579.                                                    TM_ILI9341_Puts(0, 140,"  0.5
    N", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
580.                                                    TM_ILI9341_Puts(0, 170,"  1
    N", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
581.                                                    TM_ILI9341_Puts(0, 200,"  1.5
    N", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
582.                                                    TM_ILI9341_Puts(0, 230,"  Max", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI
    9341_COLOR_CYAN);
583.                                                    TM_ILI9341_Puts(0, 260,"  Atras
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
584.
585.                                                    //Selector
586.                                                    baja_linea = 145;
587.                                                    selector_fuerza = 0;
588.                                                    salir_interfaz2_fuerza=0;
589.                                                    pulsador_ant = 0;
590.
591.                                                    while(salir_interfaz2_fuerza==0){
592.                                                        boton_siguiente = GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_10);
593.                                                        boton_ok = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12);
```

```
                                                    if(boton_siguiente==1&&pulsador_ant==0){
                                                        if(baja_linea>=270) {
                                                            TM_ILI9341_DrawFilledCircle(10,265, 5, ILI9341_COLOR_CYAN);
                                                            baja_linea=145;
                                                            selector_fuerza=0;
                                                        }

                                                        TM_ILI9341_DrawFilledCircle(10,baja_linea-30, 5, ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_DrawFilledCircle(10,baja_linea, 5, ILI9341_COLOR_RED);
                                                        baja_linea= baja_linea+30;
                                                        selector_fuerza++;
                                                    }
                                                    pulsador_ant=boton_siguiente;

                                                    if(boton_ok==1){
                                                        switch(selector_fuerza){
                                                            case 1:  // F= 0.5 N
                                                                        ControlFuerzaInterior(0.5, 2000);
                                                                        break;

                                                            case 2:  // F= 1 N
                                                                        ControlFuerzaInterior(1, 3000);
                                                                        break;

                                                            case 3:  // F= 1.5N
                                                                        ControlFuerzaInterior(1.5, 4000);
                                                                        break;

                                                            case 4:     // F= MAX
                                                                        ControlFuerzaInterior(0.5, 9000);
                                                                        break;

                                                            case 5: //salir del menú
                                                                    salir_interfaz2_fuerza=1;
                                                                    break;
                                                        }
                                                    }
                                                }
                                            TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
                                            TM_ILI9341_Puts(0, 0, "  BIONIC HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK, ILI9341_COLOR_RED);// Título
                                            TM_ILI9341_Puts(0, 50," Control de Fuerza ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                            TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
                                            TM_ILI9341_Puts(5, 100,"Seleccionar el tipo de agarre: ", &TM_Font_7x10, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                            TM_ILI9341_Puts(0, 140, "  Interno ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                            TM_ILI9341_Puts(0, 170, "  Externo ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                            TM_ILI9341_Puts(0, 200, "  Atras ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                            break;

                                            case 2: //Fuerza con sensores externos
                                                        // 2ª Interfaz
                                                        TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 0, "  BIONIC HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK,ILI9341_COLOR_RED); // Título
                                                        TM_ILI9341_Puts(0, 50," Control de Fuerza ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
                                                        TM_ILI9341_Puts(5, 100,"Seleccionar la fuerza a ejercer     con los sensores externos: ", &TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 140,"  0.5 N", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 170,"  1 N", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 200,"  1.5 N", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 230,"  Max", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                                        TM_ILI9341_Puts(0, 260,"  Atras ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);

                                                        //Selector
                                                        baja_linea = 145;
                                                        selector_fuerza = 0;
                                                        salir_interfaz2_fuerza=0;
                                                        pulsador_ant = 0;

                                                        while(salir_interfaz2_fuerza==0){
                                                            boton_siguiente = GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_10);
                                                            boton_ok = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_12);

                                                            if(boton_siguiente==1&&pulsador_ant==0){
                                                                if(baja_linea>=270) {
                                                                    TM_ILI9341_DrawFilledCircle(10,265, 5, ILI9341_COLOR_CYAN);
                                                                    baja_linea=145;
                                                                    selector_fuerza=0;
                                                                }

                                                                TM_ILI9341_DrawFilledCircle(10,baja_linea-30, 5, ILI9341_COLOR_CYAN);
                                                                TM_ILI9341_DrawFilledCircle(10,baja_linea, 5, ILI9341_COLOR_RED);
```

```
                                                        baja_linea= baja_linea+30;
                                                        selector_fuerza++;
                                                    }
                                                    pulsador_ant=boton_siguiente;

                                                    if(boton_ok==1){
                                                        switch(selector_fuerza){
                                                            case 1:  // F= 0.5 N
                                                                        ControlFuerzaExterior(0.5, 2000);
                                                                        break;

                                                            case 2:  // F= 1 N
                                                                        ControlFuerzaExterior(1, 3000);
                                                                        break;

                                                            case 3:  // F= 1.5N
                                                                        ControlFuerzaExterior(1.5, 4000);
                                                                        break;

                                                            case 4:     // F= MAX
                                                                        ControlFuerzaExterior(0.5, 9000);
                                                                        break;

                                                            case 5:     //salir del menú
                                                                        salir_interfaz2_fuerza=1;
                                                                        break;
                                                        }
                                                    }
                                                }
                                            TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
                                            TM_ILI9341_Puts(0, 0, "  BIONIC
    HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK, ILI9341_COLOR_RED);// Título
                                            TM_ILI9341_Puts(0, 50," Control de Fuerza
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                            TM_ILI9341_DrawLine(10,70,225,70, ILI9341_COLOR_BLACK);
                                            TM_ILI9341_Puts(5, 100,"Seleccionar el tipo de agarre:
    ", &TM_Font_7x10, ILI9341_COLOR_BLACK,ILI9341_COLOR_CYAN);
                                            TM_ILI9341_Puts(0, 140, "  Interno
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                            TM_ILI9341_Puts(0, 170, "  Externo
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                            TM_ILI9341_Puts(0, 200, "  Atras
    ", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                                            break;

                                            case 3: //retroceder al menu anterior
                                                        salir_interfaz1_fuerza=1;
                                                        break;
                                        }
                                    }
                                    Delayms(1);
                                }
                            ConfiguracionPantalla();
                            selector=0;
                            baja_linea=115;
                             break;

                }
            }
        Delayms(1);
        }
}
```

InicializaciónEntradasAnalógicas.c

```
1.  /*INICIALIZACIÓN Y LECTURA DE LAS ENTRADAS ANALÓGICAS
2.
3.  Mediante la función analog_inicializar() se inicializan todos los puertos necesarios para los ADC's
4.  y mediante las funciones int32_t leer_valor_PXX se hace la lectura de los datos leidos por el ADC
5.
6.  La función leer_valor_PXX_grados transforma los datos leidos del potenciómetro del servomotor a grados
7.
8.  Las funciones leer_valor_PXX_voltios transforman los datos leidos de los sensores de fuerza a voltios
9.
10. */
11.
12. #include <stdio.h>
13. #include <stm32f4xx.h>
14. #include "stm32f4xx_rcc.h"
15. #include "stm32f4xx_gpio.h"
16. #include "stm32f4xx_adc.h"
17.
18.
19. void analog_inicializar(void) {
20.
21.     GPIO_InitTypeDef        GPIO_InitStructure;
22.     ADC_InitTypeDef         ADC_InitStructure;
23.     ADC_CommonInitTypeDef   ADC_CommonInitStructure;
24.
25.     /* Puerto C, Puerto B y Puerto F -*/
26.     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
27.         RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
28.         RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);
29.
30.     /* Entradas analógicas para ADC1 con PC0/PC1/PC3 */
31.     GPIO_StructInit(&GPIO_InitStructure);
32.     GPIO_InitStructure.GPIO_Pin     = GPIO_Pin_0|GPIO_Pin_3|GPIO_Pin_1;
33.     GPIO_InitStructure.GPIO_Mode    = GPIO_Mode_AN;
34.     GPIO_InitStructure.GPIO_PuPd    = GPIO_PuPd_NOPULL ;
35.     GPIO_Init(GPIOC, &GPIO_InitStructure);
36.
37.         /* Entradas analógicas para ADC2 con PB0/PB1 */
38.     GPIO_StructInit(&GPIO_InitStructure);
39.     GPIO_InitStructure.GPIO_Pin     = GPIO_Pin_0|GPIO_Pin_1;
40.     GPIO_InitStructure.GPIO_Mode    = GPIO_Mode_AN;
41.     GPIO_InitStructure.GPIO_PuPd    = GPIO_PuPd_NOPULL ;
42.     GPIO_Init(GPIOB, &GPIO_InitStructure);
43.
44.         /* Entradas analógicas para ADC3 con PF3/PF4/PF5/PF6 */
45.     GPIO_StructInit(&GPIO_InitStructure);
46.     GPIO_InitStructure.GPIO_Pin     = GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6;
47.     GPIO_InitStructure.GPIO_Mode    = GPIO_Mode_AN;
48.     GPIO_InitStructure.GPIO_PuPd    = GPIO_PuPd_NOPULL ;
49.     GPIO_Init(GPIOF, &GPIO_InitStructure);
50.
51.     /* Activar ADC1, ADC2 y ADC3 */
52.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
53.         RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC2, ENABLE);
54.         RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC3, ENABLE);
55.
56.
57.     /* ADC Common Init */
58.     ADC_CommonStructInit(&ADC_CommonInitStructure);
59.     ADC_CommonInitStructure.ADC_Mode                = ADC_Mode_Independent;
60.     ADC_CommonInitStructure.ADC_Prescaler           = ADC_Prescaler_Div4; // max 30 MHz segun datasheet
61.     ADC_CommonInitStructure.ADC_DMAAccessMode       = ADC_DMAAccessMode_Disabled;
62.     ADC_CommonInitStructure.ADC_TwoSamplingDelay    = ADC_TwoSamplingDelay_5Cycles;
63.     ADC_CommonInit(&ADC_CommonInitStructure);
64.
65.     /* ADC Init */
66.     ADC_StructInit (&ADC_InitStructure);
67.     ADC_InitStructure.ADC_Resolution            = ADC_Resolution_12b;
68.     ADC_InitStructure.ADC_ScanConvMode          = ENABLE;
69.     ADC_InitStructure.ADC_ContinuousConvMode    = DISABLE;
70.     ADC_InitStructure.ADC_ExternalTrigConvEdge  = ADC_ExternalTrigConvEdge_None;
71.     ADC_InitStructure.ADC_DataAlign             = ADC_DataAlign_Right;
72.     ADC_InitStructure.ADC_NbrOfConversion       = 4;       //numero de conversiones  ¿?
73.
74.             /* ADC1 Init*/
75.             ADC_Init(ADC1, &ADC_InitStructure);
76.             /* ADC2 Init*/
77.             ADC_Init(ADC2, &ADC_InitStructure);
78.             /* ADC3 Init*/
79.             ADC_Init(ADC3, &ADC_InitStructure);
80.
81.     /* Establecer la configuración de conversión del ADC1 */
82.     ADC_InjectedSequencerLengthConfig(ADC1, 3);     //hasta 4
83.     ADC_SetInjectedOffset(ADC1, ADC_InjectedChannel_1, 0);
84.         ADC_SetInjectedOffset(ADC1, ADC_InjectedChannel_2, 0);
85.         ADC_SetInjectedOffset(ADC1, ADC_InjectedChannel_3, 0);
86.         //ADC_SetInjectedOffset(ADC1, ADC_InjectedChannel_4, 0);
87.     ADC_InjectedChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_480Cycles);
88.         ADC_InjectedChannelConfig(ADC1, ADC_Channel_11, 2, ADC_SampleTime_480Cycles);
89.     ADC_InjectedChannelConfig(ADC1, ADC_Channel_13, 3, ADC_SampleTime_480Cycles);
90.         //ADC_InjectedChannelConfig(ADC1, ADC_Channel_14, 4, ADC_SampleTime_480Cycles);
91.
92.     /* Establecer la configuración de conversión del ADC2 */
```

```
93.      ADC_InjectedSequencerLengthConfig(ADC2, 2);     //hasta 4
94.      ADC_SetInjectedOffset(ADC2, ADC_InjectedChannel_1, 0);
95.          ADC_SetInjectedOffset(ADC2, ADC_InjectedChannel_2, 0);
96.      ADC_InjectedChannelConfig(ADC2, ADC_Channel_8, 1, ADC_SampleTime_480Cycles);
97.          ADC_InjectedChannelConfig(ADC2, ADC_Channel_9, 2, ADC_SampleTime_480Cycles);
98.
99.
100.     /* Establecer la configuración de conversión del ADC3 */
101.     ADC_InjectedSequencerLengthConfig(ADC3, 4);     //hasta 4
102.     ADC_SetInjectedOffset(ADC3, ADC_InjectedChannel_1, 0);
103.         ADC_SetInjectedOffset(ADC3, ADC_InjectedChannel_2, 0);
104.         ADC_SetInjectedOffset(ADC3, ADC_InjectedChannel_3, 0);
105.         ADC_SetInjectedOffset(ADC3, ADC_InjectedChannel_4, 0);
106.         ADC_InjectedChannelConfig(ADC3, ADC_Channel_4, 1, ADC_SampleTime_480Cycles);
107.     ADC_InjectedChannelConfig(ADC3, ADC_Channel_9, 2, ADC_SampleTime_480Cycles);
108.         ADC_InjectedChannelConfig(ADC3, ADC_Channel_14, 3, ADC_SampleTime_480Cycles);
109.     ADC_InjectedChannelConfig(ADC3, ADC_Channel_15, 4, ADC_SampleTime_480Cycles);
110.
111.     /* Poner en marcha ADC 1 */
112.     ADC_Cmd(ADC1, ENABLE);
113.     /* Poner en marcha ADC 2 */
114.         ADC_Cmd(ADC2, ENABLE);
115.         /* Poner en marcha ADC 3 */
116.         ADC_Cmd(ADC3, ENABLE);
117.
118. }
119. //ENTRADAS ANALÓGICAS CORRESPONDIENTES A ADC1
120. int32_t leer_valor_PC0(void){           //Sensor Fuerza
121.
122.     uint16_t valor_adc_PC0;
123.
124.     // borrar flag de fin conversion
125.     ADC_ClearFlag(ADC1,ADC_FLAG_JEOC);
126.     ADC_SoftwareStartInjectedConv(ADC1);    // inicial conversion
127.
128.     while (ADC_GetFlagStatus(ADC1,ADC_FLAG_JEOC) == RESET); // esperar conversion
129.
130.     valor_adc_PC0 = ADC_GetInjectedConversionValue(ADC1, ADC_InjectedChannel_1);
131.
132.     //return;
133.     return valor_adc_PC0;
134. }
135.
136. int32_t leer_valor_PC1(void){           //Sensor Fuerza
137.         uint16_t valor_adc_PC1;
138.
139.     ADC_ClearFlag(ADC1,ADC_FLAG_JEOC);
140.     ADC_SoftwareStartInjectedConv(ADC1);
141.
142.     while (ADC_GetFlagStatus(ADC1,ADC_FLAG_JEOC) == RESET); // esperar conversion
143.
144.       valor_adc_PC1 = ADC_GetInjectedConversionValue(ADC1, ADC_InjectedChannel_2);
145.
146.     //return;
147.     return valor_adc_PC1;
148. }
149.
150. int32_t leer_valor_PC3(void){           //Sensor Fuerza
151.
152.       uint16_t valor_adc_PC3;
153.
154.     // borrar flag de fin conversion
155.     ADC_ClearFlag(ADC1,ADC_FLAG_JEOC);
156.     ADC_SoftwareStartInjectedConv(ADC1);    // inicial conversion
157.
158.     while (ADC_GetFlagStatus(ADC1,ADC_FLAG_JEOC) == RESET); // esperar conversion
159.
160.       valor_adc_PC3 = ADC_GetInjectedConversionValue(ADC1, ADC_InjectedChannel_3);
161.
162.     //return;
163.     return valor_adc_PC3;
164. }
165.
166.
167. //ENTRADAS ANALÓGICAS CORRESPONDIENTES A ADC3
168. int32_t leer_valor_PF3(void){           //Sensor Fuerza
169.
170.     uint16_t valor_adc_PF3;
171.
172.     // borrar flag de fin conversion
173.     ADC_ClearFlag(ADC3,ADC_FLAG_JEOC);
174.     ADC_SoftwareStartInjectedConv(ADC3);    // inicial conversion
175.
176.     while (ADC_GetFlagStatus(ADC3,ADC_FLAG_JEOC) == RESET); // esperar conversion
177.
178.     valor_adc_PF3 = ADC_GetInjectedConversionValue(ADC3, ADC_InjectedChannel_2);
179.
180.     //return;
181.     return valor_adc_PF3;
182. }
183.
184. int32_t leer_valor_PF4(void){           //Sensor Fuerza
185.
186.         uint16_t valor_adc_PF4;
187.
188.     // borrar flag de fin conversion
189.     ADC_ClearFlag(ADC3,ADC_FLAG_JEOC);
190.     ADC_SoftwareStartInjectedConv(ADC3);    // inicial conversion
191.
```

```
192.     while (ADC_GetFlagStatus(ADC3,ADC_FLAG_JEOC) == RESET); // esperar conversion
193.
194.     valor_adc_PF4 = ADC_GetInjectedConversionValue(ADC3, ADC_InjectedChannel_3);
195.
196.     //return;
197.     return valor_adc_PF4;
198. }
199.
200. int32_t leer_valor_PF5(void){           //Sensor Fuerza
201.
202.     uint16_t valor_adc_PF5;
203.
204.     // borrar flag de fin conversion
205.     ADC_ClearFlag(ADC3,ADC_FLAG_JEOC);
206.     ADC_SoftwareStartInjectedConv(ADC3);    // inicial conversion
207.
208.     while (ADC_GetFlagStatus(ADC3,ADC_FLAG_JEOC) == RESET); // esperar conversion
209.
210.     valor_adc_PF5 = ADC_GetInjectedConversionValue(ADC3, ADC_InjectedChannel_4);
211.
212.     //return;
213.     return valor_adc_PF5;
214. }
215.
216. int32_t leer_valor_PF6(void){           //Sensor Fuerza
217.
218.     uint16_t valor_adc_PF6;
219.
220.     // borrar flag de fin conversion
221.     ADC_ClearFlag(ADC3,ADC_FLAG_JEOC);
222.     ADC_SoftwareStartInjectedConv(ADC3);    // inicial conversion
223.
224.     while (ADC_GetFlagStatus(ADC3,ADC_FLAG_JEOC) == RESET); // esperar conversion
225.
226.     valor_adc_PF6 = ADC_GetInjectedConversionValue(ADC3, ADC_InjectedChannel_1);
227.
228.     //return;
229.     return valor_adc_PF6;
230. }
231.
232.
233. //ENTRADA ANALÓGICA CORRESPONDIENTE A ADC2
234. int32_t leer_valor_PB0(void){           //POTENCIÓMETRO MOTOR
235.
236.     uint16_t valor_adc_PB0;
237.
238.     // borrar flag de fin conversion
239.     ADC_ClearFlag(ADC2,ADC_FLAG_JEOC);
240.     ADC_SoftwareStartInjectedConv(ADC2);    // inicial conversion
241.
242.     while (ADC_GetFlagStatus(ADC2,ADC_FLAG_JEOC) == RESET); // esperar conversion
243.
244.     valor_adc_PB0 = ADC_GetInjectedConversionValue(ADC2, ADC_InjectedChannel_1);
245.     valor_adc_PB0 = valor_adc_PB0/10;
246.
247.     //return;
248.     return valor_adc_PB0;
249. }
250. int32_t leer_valor_PB1(void){           //Sensor Fuerza
251.
252.         uint16_t valor_adc_PB1;
253.
254.     ADC_ClearFlag(ADC2,ADC_FLAG_JEOC);
255.     ADC_SoftwareStartInjectedConv(ADC2);
256.
257.     while (ADC_GetFlagStatus(ADC2,ADC_FLAG_JEOC) == RESET); // esperar conversion
258.
259.       valor_adc_PB1 = ADC_GetInjectedConversionValue(ADC2, ADC_InjectedChannel_2);
260.
261.     //return;
262.     return valor_adc_PB1;
263. }
264.
265. //Conversión a Grados del valor de PB0
266. float leer_valor_PB0_grados(void){
267.     float valor_PB0_grados;
268.     valor_PB0_grados=(leer_valor_PB0()-330)/(-1.7857);
269.     return valor_PB0_grados;
270. }
271. //Conversiones a voltios de los sensores de presión
272. float leer_valor_PC0_voltios(void){
273.     float valor_PC0_voltios;
274.     valor_PC0_voltios = (leer_valor_PC0()*2.9)/4096.00;
275.     return valor_PC0_voltios;
276. }
277. float leer_valor_PC1_voltios(void){
278.     float valor_PC1_voltios;
279.     valor_PC1_voltios = (leer_valor_PC1()*2.9)/4096.00;
280.     return valor_PC1_voltios;
281. }
282. float leer_valor_PC3_voltios(void){
283.     float valor_PC3_voltios;
284.     valor_PC3_voltios = (leer_valor_PC3()*2.9)/4096.00;
285.     return valor_PC3_voltios;
286. }
287. float leer_valor_PF3_voltios(void){
288.     float valor_PF3_voltios;
289.     valor_PF3_voltios = (leer_valor_PF3()*2.9)/4096.00;
290.     return valor_PF3_voltios;
```

```
291. }
292. float leer_valor_PF4_voltios(void){
293.     float valor_PF4_voltios;
294.     valor_PF4_voltios = (leer_valor_PF4()*2.9)/4096.00;
295.     return valor_PF4_voltios;
296. }
297. float leer_valor_PF5_voltios(void){
298.     float valor_PF5_voltios;
299.     valor_PF5_voltios = (leer_valor_PF5()*2.9)/4096.00;
300.     return valor_PF5_voltios;
301. }
302. float leer_valor_PF6_voltios(void){
303.     float valor_PF6_voltios;
304.     valor_PF6_voltios = (leer_valor_PF6()*2.9)/4096.00;
305.     return valor_PF6_voltios;
306. }
307. float leer_valor_PB1_voltios(void){
308.     float valor_PB1_voltios;
309.     valor_PB1_voltios = (leer_valor_PB1()*2.9)/4096.00;
310.     return valor_PB1_voltios;
311. }
```

InicializacionMotor.c

```
1.  /* INICIALIZACIÓN Y CONTROL DEL MOTOR
2.
3.  En este fichero se controla el sentido de giro del motor
4.
5.  En la función Inicialización motor, se inicializan los pines 6 y 7 como salidas digitales
6.
7.  El resto de funciones controlan la apertura, cierre y paro del motor.
8.
9.  */
10.
11. #include <stdio.h>
12. #include <stm32f4xx.h>
13. #include "stm32f4xx_rcc.h"
14. #include "stm32f4xx_gpio.h"
15.
16. /* Inicialización salidas digitales motor. GPIOA 7 y 6 */
17. void InicializacionMotor(void){
18.
19.     GPIO_InitTypeDef motor;
20.
21.     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
22.
23.     motor.GPIO_Mode = GPIO_Mode_OUT;
24.     motor.GPIO_PuPd = GPIO_PuPd_NOPULL;
25.     motor.GPIO_OType = GPIO_OType_PP;
26.     motor.GPIO_Speed = GPIO_Speed_100MHz;
27.     motor.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7;
28.     GPIO_Init(GPIOA, &motor);
29. }
30.
31. /*  Funciones para que el motor funcione en ambos sentidos o se pare    */
32. void MotorCierrePinza(void){
33.
34.     GPIO_SetBits(GPIOA, GPIO_Pin_6);
35.     GPIO_ResetBits(GPIOA, GPIO_Pin_7);
36. }
37.
38. void MotorAperturaPinza(void){
39.
40.     GPIO_SetBits(GPIOA, GPIO_Pin_7);
41.     GPIO_ResetBits(GPIOA, GPIO_Pin_6);
42. }
43.
44. void MotorParado(void){
45.
46.     GPIO_ResetBits(GPIOA, GPIO_Pin_6);
47.     GPIO_ResetBits(GPIOA, GPIO_Pin_7);
48. }
```

InicializacionPWM.c

```
/* INICIALIZACIÓN PWM

La función InicializacionGPIOA3_PWM() inicia el puerto GPIOA3 como una salida digital modulada mediante el PWM que se inicia mediante TM_TIMER_Init()
donde se hacen todas las configuraciones.
Si llamamos a la función TM_PWM_Init(VALOR) se puede introducir el valor del PWM

*/

#include <stdio.h>
#include "defines.h"
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_tim.h"

void InicializacionGPIOA3_PWM(void) {

    GPIO_InitTypeDef GPIO_InitStruct;

    /* Clock for GPIOA */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    /* Alternating functions for pins */
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_TIM2);

    /* Set pins */
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_3;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(GPIOA, &GPIO_InitStruct);
}

void TM_TIMER_Init(void) {
    TIM_TimeBaseInitTypeDef TIM_BaseStruct;

    /* Enable clock for TIM2 */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
/*

    Set timer prescaller
    Timer count frequency is set with

    timer_tick_frequency = Timer_default_frequency / (prescaller_set + 1)

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

    timer_tick_frequency = 84000000 / (0 + 1) = 84000000
        timer_tick_frequency = 84000000 / (0 + 1) = 90000000
*/
    TIM_BaseStruct.TIM_Prescaler = 0;
    /* Count up */
    TIM_BaseStruct.TIM_CounterMode = TIM_CounterMode_Up;
/*
    Set timer period when it have reset
    First you have to know max value for timer
    In our case it is 16bit = 65535
    To get your frequency for PWM, equation is simple

    PWM_frequency = timer_tick_frequency / (TIM_Period + 1)

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

    TIM_Period = timer_tick_frequency / PWM_frequency - 1

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

    TIM_Period = 84000000 / 10000 - 1 = 8399

    If you get TIM_Period larger than max timer value (in our case 65535),
    you have to choose larger prescaler and slow down timer tick frequency
*/
    TIM_BaseStruct.TIM_Period = 9000; /* 10kHz PWM */ //8399 //1356
    TIM_BaseStruct.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_BaseStruct.TIM_RepetitionCounter = 0;
    /* Initialize TIM4 */
    TIM_TimeBaseInit(TIM2, &TIM_BaseStruct);
    /* Start count on TIM4 */
    TIM_Cmd(TIM2, ENABLE);
}


void TM_PWM_Init(uint32_t variablePWM) {
    TIM_OCInitTypeDef TIM_OCStruct;

    /* PWM mode 2 = Clear on compare match */
    /* PWM mode 1 = Set on compare match */
    TIM_OCStruct.TIM_OCMode = TIM_OCMode_PWM2;
```

```
92.       TIM_OCStruct.TIM_OutputState = TIM_OutputState_Enable;
93.       TIM_OCStruct.TIM_OCPolarity = TIM_OCPolarity_Low;
94.
95.       TIM_OCStruct.TIM_Pulse = variablePWM;
96.       TIM_OC4Init(TIM2, &TIM_OCStruct);
97.       TIM_OC4PreloadConfig(TIM2, TIM_OCPreload_Enable);
98.
99.   }
```

InicializacionInterrupcion.c

```
1.  /* INICIALIZACIÓN INTERRUPCIONES
2.
3.   En este fichero, la función Configure_PD1() se encarga de todas las configuraciones para que el puerto PD1 actue como una entrada
    con interrupción, es decir, cuando se pulse cualquiera de los dos finales de carrera ubicados en los extremos de la pinza se
    activará la entrada PD1 y se parará el motor.
4.
5.   Las funciones encargadas de detectar la interrupción se encuentran en el fichero stm32f4xx_it.c
6.
7.  */
8.
9.
10. #include <stdio.h>
11. #include "defines.h"
12. #include "stm32f4xx.h"
13. #include "stm32f4xx_rcc.h"
14. #include "stm32f4xx_gpio.h"
15. #include "stm32f4xx_exti.h"
16. #include "stm32f4xx_syscfg.h"
17. #include "misc.h"
18.
19. void Configure_PA0(void) {
20.     /* Set variables used */
21.     GPIO_InitTypeDef GPIO_InitStruct;
22.     EXTI_InitTypeDef EXTI_InitStruct;
23.     NVIC_InitTypeDef NVIC_InitStruct;
24.
25.     /* Enable clock for GPIOD */
26.     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
27.     /* Enable clock for SYSCFG */
28.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
29.
30.     /* Set pin as input */
31.         GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
32.         GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
33.         GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;
34.         GPIO_Init(GPIOA, &GPIO_InitStruct);
35.
36.     /* Tell system that you will use PD0 for EXTI_Line0 */
37.     SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
38.
39.     /* PD0 is connected to EXTI_Line0 */
40.     EXTI_InitStruct.EXTI_Line = EXTI_Line0;
41.     /* Enable interrupt */
42.     EXTI_InitStruct.EXTI_LineCmd = ENABLE;
43.     /* Interrupt mode */
44.     EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
45.     /* Triggers on rising and falling edge */
46.     EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
47.     /* Add to EXTI */
48.     EXTI_Init(&EXTI_InitStruct);
49.
50.     /* Add IRQ vector to NVIC */
51.     /* PD0 is connected to EXTI_Line0, which has EXTI0_IRQn vector */
52.     NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
53.     /* Set priority */
54.     NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x01;
55.     /* Set sub priority */
56.     NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x01;
57.     /* Enable interrupt */
58.     NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
59.     /* Add to NVIC */
60.     NVIC_Init(&NVIC_InitStruct);
61. }
62.
63.
64. void Configure_PD1(void) {
65.     /* Set variables used */
66.     GPIO_InitTypeDef GPIO_InitStruct;
67.     EXTI_InitTypeDef EXTI_InitStruct;
68.     NVIC_InitTypeDef NVIC_InitStruct;
69.
70.     /* Enable clock for GPIOD */
71.     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
72.     /* Enable clock for SYSCFG */
73.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
74.
75.     /* Set pin as input */
76.         GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
77.         GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
78.         GPIO_InitStruct.GPIO_Pin = GPIO_Pin_1;
79.         GPIO_Init(GPIOD, &GPIO_InitStruct);
80.
81.     /* Tell system that you will use PD0 for EXTI_Line0 */
82.     SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOD, EXTI_PinSource1);
83.
84.     /* PD0 is connected to EXTI_Line0 */
85.     EXTI_InitStruct.EXTI_Line = EXTI_Line1;
86.     /* Enable interrupt */
87.     EXTI_InitStruct.EXTI_LineCmd = ENABLE;
88.     /* Interrupt mode */
89.     EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
90.     /* Triggers on rising and falling edge */
```

```
91.       EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
92.       /* Add to EXTI */
93.       EXTI_Init(&EXTI_InitStruct);
94.
95.       /* Add IRQ vector to NVIC */
96.       /* PD0 is connected to EXTI_Line0, which has EXTI0_IRQn vector */
97.       NVIC_InitStruct.NVIC_IRQChannel = EXTI1_IRQn;
98.       /* Set priority */
99.       NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x00;
100.      /* Set sub priority */
101.      NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;
102.      /* Enable interrupt */
103.      NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
104.      /* Add to NVIC */
105.      NVIC_Init(&NVIC_InitStruct);
106.  }
```

ConfiguracionesPantalla.c

```
1.  /* CONFIGURACIONES ESENCIALES PARA EL FUNCIONAMIENTO DE LA PANTALLA LCD
2.
3.  Se realizan las configuraciones básicas de inicialización de la pantalla
4.
5.  */
6.
7.  #include "stm32f4xx.h"
8.  #include "defines.h"
9.  #include "tm_stm32f4_ili9341.h"
10. #include "tm_stm32f4_fonts.h"
11. #include "tm_stm32f4_delay.h"
12. #include <stdio.h>
13.
14. void ConfiguracionPantalla(void){
15.     /* Initialize system */
16.     SystemInit();
17.
18.     /* Initialize delay */
19.     TM_DELAY_Init();
20.
21.     /* Initialize ILI9341 */
22.     TM_ILI9341_Init();
23.
24.     /* Rotate LCD for 90 degrees */
25.     TM_ILI9341_Rotate(TM_ILI9341_Orientation_Portrait_2);
26.
27.     /* Fill lcd with color */
28.     TM_ILI9341_Fill(ILI9341_COLOR_CYAN);
29.
30.     // Título
31.     TM_ILI9341_Puts(0, 0, "  BIONIC HAND  ", &TM_Font_16x26, ILI9341_COLOR_BLACK, ILI9341_COLOR_RED);
32.
33.     // Menú Principal
34.     TM_ILI9341_Puts(0, 50," Menu Principal:", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
35.     TM_ILI9341_DrawLine(10,70,160,70, ILI9341_COLOR_BLACK);
36.
37.     //Submenús
38.     TM_ILI9341_Puts(0, 110,"  Abrir Pinza", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
39.     TM_ILI9341_Puts(0, 140,"  Cerrar Pinza", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
40.     TM_ILI9341_Puts(0, 170,"  Control Posicion", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
41.     TM_ILI9341_Puts(0, 200,"  Control Velocidad", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
42.     TM_ILI9341_Puts(0, 230,"  Control Fuerza", &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
43. }
```

ControlPosicion.c

```
/* CONTROL DE POSICIÓN

Se ejecuta todo el proceso que comprende el control de posición

*/


#define PI 3.141592

#include "stm32f4xx.h"
#include "defines.h"
#include <stdio.h>
#include <math.h>
#include "tm_stm32f4_delay.h"
#include "InicializacionMotor.h"
#include "InicializacionPWM.h"
#include "InicializacionEntradasAnalogicas.h"
#include "tm_stm32f4_ili9341.h"
#include "tm_stm32f4_fonts.h"

void GeneradorTrayectoria(float Pos_cm, int Tipo_Apertura){     //Tipo_Apertura=1 ->externa     Tipo_Apertura=0 ->interna

    double qr1, qr2, qpr, qsec, tm, textra, qprim;
    double s, t1, t2, tfin, q, TiempoSimulacion, acel;
    double tiempo;
    int cierre=0;
    float kp_P,kp_PD, kv_PD;
    float v2, v1, a0, a1, a2, v1_ant1=0, v1_ant2=0;
    float e;
    float u=0;          //acción de control inicial = 0
    float qreal, qreal_ant=0;       //valor de la posición real
    double Pos_Grados;
    double Pos_alcanzada_cm;


    extern volatile uint8_t off_FC;
    off_FC=1;   // Desactiva los FC para que no generaen falsas activaciones

    TM_PWM_Init(u); //Puesta a 0 del PWM

    //Parámetros del controlador
        // Control Proporcional
        kp_P= 2500;

        //Control Proporcional-Derivativo
        kp_PD = 1200;
        kv_PD = 15;

        a0 = 0.7;
        a1 = 0.2;
        a2 = 0.1;

    //Constantes
    qr1 = leer_valor_PB0_grados();  //Pos inicial (grados)
    if (Tipo_Apertura == 0) qr2 = (asin((Pos_cm-6)/(8.67*2)))*360/(2*PI);               //Apertura Interna
    else qr2 = (asin(Pos_cm/(12.5*2)))*360/(2*PI);      //Apertura Externa -- Posición final - transformación de la posición en cm a grados
    qpr = 80;   //Velocidad (grados/s)
    qsec = 100;     //Aceleración (grados/s^2)
    tm = 0.001;  //tiempo de muestreo
    textra = 0.5;   //tiempo extra a representar

    //Inicio de la función
    s=qr2-qr1;

    if(s>0) MotorAperturaPinza();
    else if (s<0){
         MotorCierrePinza();
        s=-s;   //poner el valor de s en valor absoluto
        cierre = 1; //esta variable se utilizará para condicionar determinados puntos en los que el código difiera según apertura o cierre
    }

    if (s>((qpr*qpr)/qsec)){
        t1 = qpr/qsec;
        t2 = s/qpr;
        tfin = (s/qpr)+(qpr/qsec);

    } else{
        t1 = sqrt(s/qsec);
        tfin = 2*t1;
        t2 = 0;
    }

    // Inicialización de la posición y la velocidad
    q = qr1;
    qprim = 0;  //inicialmente velocidad=0

    TiempoSimulacion = tfin+textra;
    tiempo = 0.000;

    //Bucle generador de trayectoria
```

```
91.       while(tiempo<=TiempoSimulacion){
92.
93.           // Posición y Velocidad durante el periodo de aceleración
94.           if(tiempo<t1){
95.               if(cierre==1){      //Cierre Pinza
96.                   acel = qsec;
97.                   qprim = qprim + acel*tm;
98.                   q = q - qprim*tm - 0.5*acel*tm*tm;
99.               }else{                  //Apertura Pinza
100.                  acel = qsec;
101.                  qprim = qprim + acel*tm;
102.                  q = q + qprim*tm + 0.5*acel*tm*tm;
103.              }
104.          }
105.          //Posición y Velocidad durante el periodo de velocidad constante
106.          else if (tiempo >= t1 && tiempo < t2){
107.              if(cierre==1){      //Cierre Pinza
108.                  acel = 0;
109.                  qprim = qpr;
110.                  q = q - qprim*tm;
111.              }else{                      //Apertura Pinza
112.                  acel = 0;
113.                  qprim = qpr;
114.                  q = q + qprim*tm;
115.              }
116.          }
117.          //Posición y Velocidad durante el periodo de deceleración
118.          else if (tiempo >= t2 && tiempo < tfin){
119.              if(cierre==1){          //Cierre Pinza
120.                  acel = -qsec;
121.                  qprim = qprim + acel*tm;
122.                  q = q - qprim*tm - 0.5*acel*tm*tm;
123.              }else{                          //Apertura Pinza
124.                  acel = -qsec;
125.                  qprim = qprim + acel*tm;
126.                  q = q + qprim*tm + 0.5*acel*tm*tm;
127.              }
128.          }
129.          else{
130.              acel = 0;
131.              qprim = 0;
132.              q = qr2;
133.          }
134.
135.          qreal = leer_valor_PB0_grados();
136.
137.          e = q - qreal;   //cálculo del error de posición
138.
139.          if(cierre==1) e = -e;
140.
141.          // CONTROL P
142.          /*
143.              u = kp_P * e;   //cálculo de la acción de control PROPORCIONAL
144.          */
145.          // CONTROL PD
146.
147.              v1 = (qreal - qreal_ant)/tm;    //estimación de la velocidad
148.              v2 = a0*v1 + a1*v1_ant1 +a2*v1_ant2;    //filtro
149.
150.              u = kp_PD*e - kv_PD*v2; //acción de control
151.
152.              //actualización de los valores anteriores
153.              qreal_ant = qreal;
154.              v1_ant2 = v1_ant1;
155.              v1_ant1 = v1;
156.
157.
158.          if(u<0){u=0;} //Para evitar la existencia de u negativa (ya que si se introduce un valor negativo en PWM, este funciona
     mal)
159.              TM_PWM_Init(u);
160.
161.      tiempo=tiempo + 0.001;
162.      Delayms(1);
163.
164.      }
165.
166.          MotorParado();      // Para el motor cuando se acaba de generar la trayectoria
167.          off_FC=0;           //Vuelve a activar la interrupción de los FC
168.
169.          //  Mostrar en pantalla el valor de la lectura del potenciómetro
170.
171.          Pos_Grados = leer_valor_PB0_grados();
172.
173.          if (Tipo_Apertura == 0)Pos_alcanzada_cm = (8.67*sin(Pos_Grados*2*PI/360)*2)+6;      //interno
174.          else    Pos_alcanzada_cm = 12.5*sin(Pos_Grados*2*PI/360)*2;             //externo
175.
176.          uint8_t aTextBuffer[50];
177.
178.          sprintf((char*)aTextBuffer, "Grados: %.2f", Pos_Grados);
179.          TM_ILI9341_Puts(30, 260, (uint8_t*)aTextBuffer, &TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
180.          sprintf((char*)aTextBuffer, "Posicion: %.2f", Pos_alcanzada_cm);
181.          TM_ILI9341_Puts(30, 280, (uint8_t*)aTextBuffer, &TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
182.  }
```

ControlVelocidad.c

```
/* CONTROL DE VELOCIDAD

Esta función contiene un switch que establece una velocidad y una dirección de la pinza, en función de los parámetros
que seleccione el usuario.
A la salida, devuelve el cálculo de la velocidad entre dos puntos para comprobar si se sigue la velocidad que se le ha introducido

*/

#include "stm32f4xx.h"
#include "defines.h"
#include <stdio.h>
#include <math.h>
#include "tm_stm32f4_delay.h"
#include "InicializacionMotor.h"
#include "InicializacionPWM.h"
#include "InicializacionEntradasAnalogicas.h"
#include "tm_stm32f4_ili9341.h"
#include "tm_stm32f4_fonts.h"

void ControlVelocidad(int sentido, int a_control, int impr_pantalla){
    float u=0;
    float pos_inicial=0.00, pos_final=0.00, velocidad=0;
    TM_PWM_Init(u); //Puesta a 0 del PWM

    if(sentido==0){                     //Apertura Pinza
        MotorAperturaPinza();
        switch(a_control){
            case 25: u = 2250; break;
            case 50: u = 4500; break;
            case 75: u = 6750; break;
            case 100: u = 9000; break;
            default: u = 0;
        }
        TM_PWM_Init(u);

    }else if(sentido==1){           //Cierre Pinza
        MotorCierrePinza();
        switch(a_control){
            case 25: u = 2250; break;
            case 50: u = 4500; break;
            case 75: u = 6750; break;
            case 100: u = 9000; break;
            default: u = 0;
        }
        TM_PWM_Init(u);
    }else MotorParado();

    //Medición de la velocidad en un instante de timepo de 150 ms
    Delayms(50);
    pos_inicial=leer_valor_PB0_grados();
    Delayms(200);
    pos_final=leer_valor_PB0_grados();
    velocidad= (pos_final-pos_inicial)/150;

    // Impresión en pantalla de resultados
            uint8_t aTextBuffer[50];
            if(impr_pantalla==1){
                sprintf((char*)aTextBuffer, "      ");
                TM_ILI9341_Puts(85, 170, (uint8_t*)aTextBuffer, &TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
                sprintf((char*)aTextBuffer, "V media= %.2f grad/ms", velocidad);
                TM_ILI9341_Puts(85, 170, (uint8_t*)aTextBuffer, &TM_Font_7x10, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
            }
}
```

ControlFuerza.c

```
/* CONTROL DE FUERZA

En estas dos funciones se realiza el control de fuerza según se haya seleccionado agarre interior o exterior

*/

#include <stdio.h>
#include <stm32f4xx.h>
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_adc.h"
#include "defines.h"
#include "tm_stm32f4_ili9341.h"
#include "tm_stm32f4_fonts.h"
#include <math.h>
#include "tm_stm32f4_delay.h"
#include "InicializacionMotor.h"
#include "InicializacionPWM.h"
#include "InicializacionEntradasAnalogicas.h"

void ControlFuerzaInterior (float f_ref, float u_ref){

    float PC1=0, PF5=0, PC3=0;  //sensores interior
    float umbral=0.2;
    int u, fase1=0, fase2=0;
    float a0,a1,a2;
    float masa=0, f_medida=0, f_medida_ant1, f_medida_ant2;
    float f_est=0, f_est_der=0, f_est_der_ant=0;
    float kp,kv;
    float voltiosPC3;
    uint8_t aTextBuffer[50];

    //Variables
        a0= 0.7;
        a1=0.2;
        a2=0.1;

        kp=200;
        kv=10;


    while((PC1<=umbral)&&(fase1==0)){  //1ª Fase - Velocidad CTE hasta que un sensor detecte umbral

                PC1=leer_valor_PC1_voltios();

                u=4000;     //movimiento del motor lento
                MotorCierrePinza();
                TM_PWM_Init(u);
                GPIO_ResetBits(GPIOG, GPIO_Pin_13);
    }

    fase1=1;        //se bloquea el bucle de arriba para que una vez uno de los sensores supere umbral, no se vuelva a entrar

    while((f_medida<f_ref)&&(fase2==0)){
                GPIO_SetBits(GPIOG, GPIO_Pin_13);       //Enciende el Led Verde

                PC1=leer_valor_PC1_voltios();
                PF5=leer_valor_PF5_voltios();
                PC3=leer_valor_PC3_voltios();

                if((PC1>PC3)&&(PC1>PF5))    masa = (leer_valor_PC1_voltios()-0.2763)/(0.0048);
                else if((PC3>PC1)&&(PC3>PF5))       masa = (leer_valor_PC3_voltios()-0.2763)/(0.0048);
                else if ((PF5>PC1)&&(PF5>PC3))  masa = (leer_valor_PF5_voltios()-0.2763)/(0.0048);

                f_medida = (masa/1000)*9.81;

                f_est = a0*f_medida + a1*f_medida_ant1 +a2*f_medida_ant2 ;
                f_est_der = f_est - f_est_der_ant;

                //CONTROL CON PREALIMENTACIÓN
                u = kp*(f_ref - f_est) + kv*(0 - f_est_der) + u_ref;

                if(u<0) u=0;
                TM_PWM_Init(u);

        f_medida_ant2= f_medida_ant1;
        f_medida_ant1= f_medida;
        f_est_der_ant = f_est_der;

        sprintf((char*)aTextBuffer, "          ");
        TM_ILI9341_Puts(100, 170, (uint8_t*)aTextBuffer, &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
        sprintf((char*)aTextBuffer, "%.2F, %d", f_medida, u);
        TM_ILI9341_Puts(100, 170, (uint8_t*)aTextBuffer, &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
        TM_ILI9341_DrawRectangle(95,160,210,190, ILI9341_COLOR_BLACK);

                Delayms(1);

    }
    GPIO_ResetBits(GPIOG, GPIO_Pin_13);
    fase2=1;

    Delayms(1);
```

```
93.  }
94.
95.  void ControlFuerzaExterior (float f_ref, float u_ref){
96.
97.      float PC0=0, PF3=0, PB1=0, PF4=0;   //sensores exterior
98.      float umbral=0.2;
99.      int u, fase1=0, fase2=0;
100.
101.     float a0,a1,a2;
102.     float masa=0, f_medida=0, f_medida_ant1, f_medida_ant2;
103.     float f_est=0, f_est_der=0, f_est_der_ant=0;
104.     float kp,kv;
105.
106.
107.     uint8_t aTextBuffer[50];
108.
109.     //Variables
110.         a0= 0.7;
111.         a1=0.2;
112.         a2=0.1;
113.
114.         kp=200;
115.         kv=10;
116.
117.
118.     while((PB1<=umbral)&&(fase1==0)){  //1ª Fase - Velocidad CTE hasta que un sensor detecte umbral
119.
120.                 PB1=leer_valor_PB1_voltios();
121.
122.                 u=4000;     //movimiento del motor lento
123.                 MotorCierrePinza();
124.                 TM_PWM_Init(u);
125.                 GPIO_ResetBits(GPIOG, GPIO_Pin_13);
126.     }
127.
128.     fase1=1;        //se bloquea el bucle de arriba para que una vez uno de los sensores supere umbral, no se vuelva a entrar
129.
130.     while((f_medida<f_ref)&&(fase2==0)){
131.                 GPIO_SetBits(GPIOG, GPIO_Pin_13);       //Enciende el Led Verde
132.
133.                 PB1=leer_valor_PB1_voltios();
134.                 PF4=leer_valor_PF4_voltios();
135.                 PC0=leer_valor_PC0_voltios();
136.                 PF3=leer_valor_PF3_voltios();
137.
138.                 if((PB1>PF4)&&(PB1>PC0)&&(PB1>PF3))     masa = (leer_valor_PB1_voltios()-0.2763)/(0.0048);
139.                 else if((PF4>PB1)&&(PF4>PC0)&&(PF4>PF3))        masa = (leer_valor_PF4_voltios()-0.2763)/(0.0048);
140.                 else if ((PC0>PB1)&&(PC0>PF4)&&(PC0>PF3))       masa = (leer_valor_PC0_voltios()-0.2763)/(0.0048);
141.                 else if ((PF3>PB1)&&(PF3>PF4)&&(PF3>PC0))       masa = (leer_valor_PF3_voltios()-0.2763)/(0.0048);
142.
143.                 f_medida = (masa/1000)*9.81;
144.
145.                 f_est = a0*f_medida + a1*f_medida_ant1 +a2*f_medida_ant2 ;
146.                 f_est_der = f_est - f_est_der_ant;
147.
148.                 //CONTROL CON PREALIMENTACIÓN
149.                 u = kp*(f_ref - f_est) + kv*(0 - f_est_der) + u_ref;
150.
151.                 if(u<0) u=0;
152.                 TM_PWM_Init(u);
153.
154.         f_medida_ant2= f_medida_ant1;
155.         f_medida_ant1= f_medida;
156.         f_est_der_ant = f_est_der;
157.
158.
159.
160.         sprintf((char*)aTextBuffer, "          ");
161.         TM_ILI9341_Puts(100, 170, (uint8_t*)aTextBuffer, &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
162.         sprintf((char*)aTextBuffer, "%.2F, %d", f_medida, u);
163.         TM_ILI9341_Puts(100, 170, (uint8_t*)aTextBuffer, &TM_Font_11x18, ILI9341_COLOR_BLACK, ILI9341_COLOR_CYAN);
164.         TM_ILI9341_DrawRectangle(95,160,210,190, ILI9341_COLOR_BLACK);
165.
166.                 Delayms(1);
167.
168.     }
169.     GPIO_ResetBits(GPIOG, GPIO_Pin_13);
170.     fase2=1;
171.
172.     Delayms(1);
173. }
```