UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Gutiérrez Jiménez, José

Tutor/a: Zotovic Stanisic, Ranko

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico

Trabajo Fin de Grado
Grado en Ingeniería Electrónica Industrial y Automática

Documentos:

1. Memoria
2. Planos
3. Pliego de condiciones
4. Presupuesto

Autor: José Gutiérrez Jiménez
Tutor: Ranko Zotovic Stanisic

Curso académico 2023-2024

RESUMEN

Los servomotores son actuadores rotativos o lineales que permiten el funcionamiento de diferentes dispositivos gracias a su control de la posición, velocidad o aceleración de mecanismos. Es de vital importancia para el alumnado saber utilizar correctamente estos dispositivos ya que tiene aplicaciones en una gran cantidad de ámbitos y situaciones, desde aplicaciones industriales y robóticas hasta la aviónica. Para ello, el fin de este proyecto es servir como apoyo en las sesiones de prácticas donde se enseñe las diferentes formas de controlar un servomotor para que el alumnado pueda comprobar de forma real las acciones de los diferentes tipos de controladores.

Para lograr este objetivo, la realización de este trabajo consiste en la configuración de un servomotor de bajo coste en sus diferentes modos (posición, velocidad, tensión y corriente) para luego implementar diferentes formas de control: PID, compensación de rozamiento, de gravedad, etc.

Para la implementación se ha usado servomotor Dynamixel XL330-M288-T, un dispositivo de bajo coste que consigue hacer una fuerza inesperada para sus dimensiones al mismo tiempo que resulta sencillo en su programación. Para llevar esta a cabo se ha seleccionado la placa OpenRB-150, que está preparada especialmente para los servomotores de Dynamixel además de ser compatible con ARDUINO IDE. Para completar el circuito se ha usado la etapa de alimentación U2D2 Power Hub Board. Además, para una cómoda implementación durante las sesiones académicas, se han diseñado las diferentes partes del prototipo para ver las capacidades del robot de forma sencilla y se ha creado una interfaz mediante Matlab App Designer para hacer la comunicación entre servomotor-humano.

Palabras clave: Control; PID; compensación de gravedad; servomotor; Arduino; Matlab

ABSTRACT

Servomotors are rotary or linear actuators that enable the operation of various devices thanks to their control over the position, speed, or acceleration of mechanisms. It is vital for students to learn how to use these devices correctly as they have applications in a wide range of fields and situations, from industrial and robotic applications to avionics. Therefore, the goal of this project is to support practical sessions where different ways to control a servomotor are taught so that students can understand the actions of different types of controllers in real scenarios.

To achieve this objective, this work consists of the configuration of a low-cost servomotor in its different modes (position, speed, voltage, and current) and then implementing different control methods: PID, friction compensation, gravity compensation, etc.

For the implementation, it was used the Dynamixel XL330-M288-T servomotor, a low-cost device that achieves unexpected force for its size while being easy to program. Regarding programming, the OpenRB-150 board, which is specially prepared for Dynamixel servomotors and compatible with the ARDUINO IDE, was selected. To complete the circuit, the U2D2 Power Hub Board power stage has been used. Additionally, for comfortable implementation during academic sessions, the different parts of the prototype have been designed to easily demonstrate the robot's capabilities. Finally, an interface has been created using Matlab App Designer to facilitate communication between the servomotor and the user.

Keywords: Control; PID; gravity compensation; servomotor; Arduino; Matlab

RESUM

Els servomotors són actuadors rotatius o lineals que permeten el funcionament de diferents dispositius gràcies al seu control de la posició, velocitat o acceleració de mecanismes. És de vital importància per a l'alumnat saber utilitzar correctament aquests dispositius ja que tenen aplicacions en una gran quantitat d'àmbits i situacions, des d'aplicacions industrials i robòtiques fins a l'aviònica. Per això, la finalitat d'aquest projecte és servir com a suport en les sessions de pràctiques on s'ensenye les diferents formes de controlar un servomotor perquè l'alumnat puga comprobar de manera real les accions dels diferents tipus de controladors.

Per a aconseguir aquest objectiu, la realització d'aquest treball consisteix en la configuració d'un servomotor de baix cost en els seus diferents modes (posició, velocitat, tensió i corrent) per a després implementar diferents formes de control: PID, compensació de fricció, de gravetat, etc.

Per a la implementació s'ha utilitzat el servomotor Dynamixel XL330-M288-T, un dispositiu de baix cost que aconsegueix fer una força inesperada per a les seues dimensions al mateix temps que resulta senzill en la seua programació. Para dur a terme aquesta tasca s'ha seleccionat la placa OpenRB-150, que està preparada especialment per als servomotors de Dynamixel a més de ser compatible amb ARDUINO IDE. Per a completar el circuit s'ha utilitzat l'etapa d'alimentació U2D2 Power Hub Board. A més, per a una còmoda implementació durant les sessions acadèmiques, s'han dissenyat les diferents parts del prototip per a veure les capacitats del robot de manera senzilla i s'ha creat una interfície mitjançant Matlab App Designer per a fer la comunicació entre servomotor-humà.

Paraules Clau: Control; PID; compensació de gravetat; servomotor; Arduino

AGRADECIMIENTOS

Primero de todo, me gustaría agradecer a mi Tutor, Ranko Zotovic, por toda la formación y tiempo a través de tutorías y largos hilos de correos que me ha dedicado para sacar este proyecto adelante. Del mismo modo, me gustaría agradecer al técnico de impresión 3D Pedro José Ayala por toda su ayuda durante el proceso de diseño e impresión de las piezas desarrolladas en este trabajo.

Por último, pero no menos importante, quiero agradecer a mi familia y personas más cercanas por todo el apoyo y comprensión que me han transmitido a lo largo de este TFG.

ÍNDICE GENERAL DEL PROYECTO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.

Trabajo Fin de Grado
Grado en Ingeniería Electrónica Industrial y Automática

Documento 1:

Memoria

Autor: José Gutiérrez Jiménez
Tutor: Ranko Zotovic Stanisic

Curso académico 2023-2024

ÍNDICE

1 OBJETO DEL TRABAJO

1.1 Objeto

El objeto de este Trabajo de Fin de Grado (abreviado TFG) se refiere al diseño e implementación de los diferentes tipos de control más comunes para servomotores (abreviados como servos) con un objetivo académico. Para ello se ha seleccionado un servo de bajo coste que de fábrica ofrecía varios de los modos de control más comunes y se ha expandido en torno a ellos utilizando tanto Matlab como Arduino para tener una interfaz entre servomotor y usuario clara y sencilla.

Con este propósito se han creado diferentes piezas para el montaje de un prototipo capaz de soportar un peso considerable y que rote de manera segura tanto de manera horizontal como vertical.

El diseño definitivo busca ofrecer un producto económico que pueda ser usado para educar sobre los distintos modos de control a través de una implementación real. Por lo tanto, posee un valor práctico, académico y funcional.

Por ende, este proyecto abarca todas las cuatro etapas de la elaboración del dispositivo. Primero, se detalla la selección de los componentes electrónicos. Luego, con base en las especificaciones de estos componentes, se diseñan y eligen las distintas partes del prototipo. En tercer lugar, se desarrolla la programación de los diferentes modos de control y, por último, se realizan pruebas para comprobar el correcto funcionamiento del servomotor.

1.2 Justificación

En la actualidad, los servomotores juegan un papel crucial en distintos ámbitos de la tecnología, con especial mención en el mundo de la industria, ya que los servos no solo son componentes en diferentes máquinas de ensamblaje o en manipuladores, sino que son los principales motores y motivo de la precisión de los sistemas robotizados.

Este TFG se justifica ante la necesidad de que futuros ingenieros comprendan de manera real y efectiva como afectan los diferentes tipos de algoritmos de control a un servomotor.

El producto final desarrollado soluciona esta necesidad a través de programas familiares para cualquier estudiante de tecnología, como son Arduino y Matlab, y un prototipo seguro y fácil de controlar.

1.3 Motivación

El motivo para el desarrollo de este trabajo surge de la necesidad de comprender de manera efectiva y clara el propósito de los diferentes tipos de control en la electrónica, así como su correcta implementación en dispositivos reales que enfrentan diversas perturbaciones y no se encuentran en un entorno teórico ideal.

Además, este proyecto aborda lo que se podría considerar como las áreas más importantes para la ingeniería electrónica. En primer lugar, implica un claro entendimiento de las especificaciones de los diferentes tipos de componentes electrónicos, el diseño adecuado de sus diversas partes y la correcta disposición para la creación de un prototipo que cumpla con

los objetivos establecidos. También se fundamenta en una base teórica sólida de los distintos tipos de control y automatización, y finalmente, en cómo implementar estos principios a través de la programación de dispositivos electrónicos.

2 ANTECEDENTES

2.1 Antecedentes servomotor

Un servomotor, muchas veces abreviado como servo, es un motor rotacional o lineal que tiene la capacidad de controlar de manera precisa la posición, la velocidad y la aceleración además de aplicar un par o fuerza a un sistema mecánico, como puede ser un actuador. Una manera fácil de imaginar un servomotor es un dispositivo que combina la fuerza de un motor con la precisión de un reloj suizo. El término "servo" proviene de la palabra latina "servus", que significa "esclavo" o "sirviente", lo que refleja el papel del motor en seguir de manera precisa las señales de control.

La historia de los servomotores comienza con los servomecanismos, cuyos fundamentos teóricos se pueden rastrear hasta 1868, cuando J.C. Maxwell publicó su trabajo sobre los reguladores centrífugos en máquinas de vapor. Este trabajo sentó las bases para el desarrollo de sistemas de control automático y retroalimentación. Sin embargo, la implementación práctica de servomecanismos eléctricos capaces de controlar la posición de componentes como los timones de los barcos se desarrolló durante los años 1920. En la década de 1930, el desarrollo de amplificadores electrónicos permitió avances notables en la tecnología de servomecanismos, lo que resultó en sistemas de control más precisos y con mejor capacidad de respuesta. En los años 40, durante la Segunda Guerra Mundial, estos sistemas se usaron ampliamente en aplicaciones militares como la colocación de cañones y el control de tiro, demostrando su eficacia en situaciones de alto riesgo que requerían precisión y control.

Alrededor de 1950, la integración de servomecanismos en la maquinaria industrial transformó los procesos de fabricación, llevando al desarrollo de las primeras máquinas de control numérico (CN). Siguiendo, en los años 60 la carrera espacial impulsó el desarrollo de sofisticados sistemas servo para el posicionamiento de cohetes y satélites, probando su fiabilidad en condiciones extremas.

Durante los años 70, los avances en microelectrónica y control digital expandieron aún más las capacidades de los servomecanismos, permitiendo su uso en sistemas más complejos y compactos, pareciéndose a lo que se conoce hoy en día como servo, un motor de tamaño modesto en comparación a la fuerza que pueden aplicar. Desde la década de 1980 hasta la actualidad, la proliferación de ordenadores y la tecnología digital ha integrado los servomecanismos en una amplia gama de aplicaciones, incluyendo robótica, dispositivos médicos, sistemas de energías renovables y componentes claves de la automatización industrial.

Finalizando el contexto histórico, una definición más precisa de servomotor es un servomecanismo de circuito cerrado que utiliza retroalimentación de posición para controlar su velocidad y posición. La señal de control puede ser una señal analógica o una señal digital que represente el comando de posición final del eje. Los codificadores desempeñan un papel fundamental en este sistema, actuando como sensores y proporcionando retroalimentación de velocidad y posición.

En la robótica moderna, los actuadores más comunes son los servomotores eléctricos, que se dividen en dos tipos principales: los motores de corriente continua con imán permanente y los motores sin escobillas de corriente alterna. Aunque la preferencia está inclinada hacia los últimos debido a su eficiencia, el modelo matemático subyacente es idéntico para ambos tipos. Por lo tanto, todo lo explicado en este proyecto será aplicable sin distinción para ambos tipos de motores.

2.2 Antecedentes del control automático

Un sistema de control automático se define como un conjunto integrado de componentes que regulan su propio rendimiento o salida sin necesidad de intervención externa, incluyendo la corrección de posibles errores durante su operación. En cualquier sistema mecánico, industrial o planta, se distingue una parte actuadora que ejecuta la acción física y otra de control que genera las instrucciones para dirigir dicha acción. Para ilustrar el principio de funcionamiento de un sistema de control, podemos utilizar el ejemplo de un arquero. El arquero apunta y dispara a una diana. Si la flecha impacta por debajo del objetivo, ajusta su próxima flecha elevando más el arco; si la flecha se eleva demasiado, reduce la elevación del arco en el próximo intento. Este proceso de ajuste continuo, realizado por el arquero, representa el componente de control que emite las órdenes para modificar la acción del brazo, que actúa como el elemento actuador en este contexto.

La historia del control automático ha experimentado un rápido desarrollo en los últimos años, aunque su evolución se remonta mucho tiempo atrás. En la civilización griega, desde el año 800 a.C., ya se vislumbraban los primeros pensamientos sobre la automatización del movimiento, como lo refleja Homero en sus escritos, como “La Ilíada”. Más adelante, en el siglo IV a.C., Aristóteles anticipaba que la automatización podría liberar el trabajo humano si cada herramienta pudiera ejecutar su tarea obedeciendo órdenes o incluso anticipándose a ellas. Este periodo marcó los primeros indicios del interés humano por controlar y automatizar procesos, sentando las bases para el desarrollo posterior del control automático como disciplina tecnológica y científica.

El inicio de los sistemas de control moderno está en 1788 con James Watt y su regulador realimentado mecánico, y hoy en día, gracias a su drástica evolución, el campo del control automático industrial cubre muchos temas interesantes que tratar. En este apartado nos centraremos en el controlador proporcional, integral y derivativo, o también conocido como controlador PID.

El concepto de control PID, que incluye acciones proporcional, integral y derivativa, de ahí sus siglas “PID”, tiene sus raíces en el siglo XIX, aunque su formulación práctica como un regulador de tres términos específicos no se introdujo hasta finales de la década de 1930 por la Taylor Instrument Company. Nicolás Minorsky, en 1922, fue pionero en el análisis y aplicación de controladores PID para mejorar la estabilidad direccional de los buques, lo cual sentó las bases teóricas para su desarrollo. A través de sus investigaciones, Minorsky demostró cómo los controladores de tipo PID podían corregir errores causados por perturbaciones temporales, aunque también identificó sus limitaciones frente a perturbaciones constantes como vientos estables.

A pesar de los avances y la efectividad probada de los sistemas implementados por Minorsky en el buque New Mexico en 1923, persiste una cierta desconfianza hacia los sistemas automáticos de control, lo cual ha sido un desafío continuo para su adopción generalizada. No obstante, los estudios de Minorsky sobre la teoría de sistemas y la estabilidad han contribuido significativamente a la comprensión y aceptación de los controladores PID en diversas aplicaciones industriales.

En la actualidad, el control discreto es ampliamente predominante en la mayoría de las aplicaciones, permitiendo implementar desde técnicas básicas como el control proporcional hasta métodos avanzados como el control predictivo, adaptativo y basado en redes neuronales, entre otros. A pesar de estas innovaciones, el clásico controlador PID sigue siendo ampliamente utilizado en la industria debido a su efectividad y comprensión bien establecida de su comportamiento.

3 ESTUDIO DE NECESIDADES, FACTORES A CONSIDERAR: LIMITACIONES Y CONDICIONANTES

3.1 Condicionantes teóricos

El objetivo de este trabajo es crear una herramienta para utilizar en prácticas de laboratorio que antes utilizaban solo simulación de los diferentes casos de control. Concretamente, está centrada en ser un apoyo para la práctica número 4 de la asignatura de Sistemas Robotizados del grado de Ingeniería Electrónica Industrial y Automática. En esta, el alumnado utiliza Matlab y Simulink para crear una trayectoria trapezoidal de posición además de crear modelos teóricos de los distintos modos control para comprobar el efecto del control en la trayectoria de referencia. Aunque la herramienta haya sido diseñada para estos casos esta puede ser usada para cualquier asignatura que necesite enseñar en la realidad los funcionamientos de un controlador PID.

Volviendo a las necesidades derivadas directamente de la práctica, encontramos que el servomotor seleccionado ha de ser de corriente continua a imán permanente o de motores sin escobillas de corriente alterna, ya que el marco teórico del control se aplica a los dos tipos sin distinción. Adicionalmente, el servomotor ha de ser capaz de ser configurado por los 4 modos principales de control:

- Control por tensión (PWM/modulación por ancho de pulso).
- Control por corriente (o par).
- Control por velocidad.
- Control por posición.

Por último, la interfaz para mandar instrucciones al servomotor no debe suponer un gran esfuerzo de compresión para el usuario ya que han de centrarse en entender la parte teórica del control.

3.2 Condicionantes físicos

Debido a que el dispositivo ha de ser manipulado durante las prácticas por los estudiantes, este ha de tener un tamaño y peso capaz de sentir los diferentes rozamientos y perturbaciones que se quieren compensar a través del control. Al mismo tiempo, ese peso no puede resultar en un peligro para los estudiantes si se llegara a descontrolar debido a unos valores erróneos en el prototipo.

Por último, al tener un fin de uso académico, se ha de buscar reducir los costes dentro de lo posible para que el factor económico no impida poner en práctica el dispositivo. Esto afecta en la elección de componentes electrónicos, ya que sin un precio límite se pueden escoger componentes de gran calidad pero que en realidad no resulten en una mejoría del aprendizaje

del alumnado.

Por estos motivos, la selección del servomotor tiene que seguir las siguientes condiciones:

- Servo relativamente pequeño, como se podrían encontrar como actuadores en los robots fabricados durante el grado de ingeniería electrónica (robots de dimensiones parecidas a 30 x 30 x 30 cm).
- Servo con suficiente fuerza para sostener una articulación de aproximadamente 10 cm de largo con un peso razonable.
- Su precio ha de ser de bajo coste, como puede ser inferior a 50 €.
- Capaz de ser alimentado por un voltaje y amperaje dentro del rango de 1-15 V y de 1-3 A, para que en caso de funcionamiento erróneo de los componentes no haya peligros de seguridad.

4 PLANTEAMIENTO DE SOLUCIONES ALTERNATIVAS Y JUSTIFICACIÓN DE LA SOLUCIÓN ADOPTADA

4.1 Alternativas de servomotor

El primer servo que se podría considerar es uno con el que los alumnos estén algo familiarizados con él, que son los de muy bajo coste. El servo MG90S es un micro servo ampliamente utilizado en aplicaciones de robótica y modelismo, conocido por su combinación de tamaño compacto, ligereza y durabilidad, y sobretodo su económico precio. Con un precio de 4,50 € tiene engranajes metálicos, ofreciendo resistencia al desgaste, lo que lo hace adecuado para tareas que requieren movimientos precisos y repetidos. Presenta un par de fuerza (“torque” en inglés) máximo de aproximadamente 2.2 kg.cm a 4.8V y una velocidad de 100 revoluciones por minuto (rpm) a 4.8V, lo que le permite realizar movimientos rápidos y precisos. Sus dimensiones son 22.8 x 12.2 x 28.5 mm y pesa alrededor de 13.4 g, lo que lo convierte en una opción ideal para proyectos modestos donde el espacio y el peso son factores críticos. Como defectos, no tiene un codificador (“encoder” en inglés) o sensores en su diseño para medir su posición o velocidad actuales, de ahí su bajo precio. En su defecto, utiliza un potenciómetro interno para proporcionar retroalimentación de posición. Este potenciómetro está conectado al eje de salida y envía una señal de retroalimentación al controlador del servo, lo que permite que el servo ajuste su posición en función de la señal de control recibida.

Imagen 1. Servomotor MG90S

Una opción, todavía de relativo bajo coste, es el servo Dynamixel XL330-M288-T . Este motor inteligente de la serie XL de Robotis está diseñado para aplicaciones robóticas avanzadas que requieren control preciso y retroalimentación detallada. Por lo tanto, ofrece un par de

fuerza de aproximadamente 5.3 kg.cm a 5.0 V y una velocidad de 103 rpm a 5.0 V, lo que lo hace adecuado para tareas que requieren tanto fuerza como precisión. El XL-330-M288-T incorpora comunicación mediante protocolo TTL, lo que permite la integración en sistemas complejos y el control de múltiples servos simultáneamente. Con unas dimensiones de 20 x 34 x 26 mm y un peso de 18 g, este servo es compacto pero robusto, ideal para aplicaciones en robótica educativa y proyectos de investigación. La principal ventaja con el servo anterior se encuentra en el encoder integrado. Este lo distingue de servos más simples, como el MG90S, que utilizan potenciómetros para la retroalimentación de posición. Los encoders proporcionan una mayor precisión y capacidades avanzadas de control, lo que los hace adecuados para aplicaciones más complejas y exigentes. Además, al ser un servo inteligente, de fábrica tiene instalados los modos de control por corriente, tensión, velocidad y posición. Se puede encontrar en la tienda oficial de Robotis por 23.90 €, y el mayor defecto es que por ese precio no se pueden encontrar codificadores de mayor precisión, por lo que puede dar lugar a resultados no tan preciosos.

Imagen 2. Dynamixel XL330-M288-T

Por último, con un coste en su página web de 85.68 €, podemos encontrar la opción del motor DC-max 16 S Ø16 mm de Maxon. Este es un motor de corriente continua de alta precisión y rendimiento, utilizado en una amplia gama de aplicaciones industriales y robóticas. Con un diámetro de 16 mm, este motor compacto ofrece una alta densidad de potencia y eficiencia energética, características esenciales para aplicaciones exigentes. Los motores DCX de Maxon, como el DC-max 16 S, son conocidos por su construcción robusta y larga vida útil, y pueden ser personalizados para adaptarse a necesidades específicas de torque y velocidad. Con un voltaje nominal de 6 V, este servomotor es capaz de llegar a 7890 rpm de velocidad sin carga y capaz de soportar 0.107 kg.cm. Este motor es ideal para aplicaciones donde se requiere un control preciso y una respuesta rápida, como en sistemas de automatización, robótica de alta precisión y dispositivos médicos. En su contra se puede encontrar su alto precio y su bajo par de fuerza. Tiene unas especificaciones muy altas en términos de rpm, pero que son innecesarias para el uso que se le quiere dar en la práctica. Además, este servo no cuenta con un encoder propio de fábrica.

Imagen 3. Servomotor DC-max 16 S

4.2 Alternativas de microcontrolador

La primera placa electrónica considerada fue el Arduino Nano, ya que es muy probable que los alumnos ya hayan trabajado con ella durante del grado al ser una de las placas más populares para hacer los primeros proyectos de electrónica, además de ofrecer mayores capacidades que una placa Arduino Uno convencional. El Arduino Nano cuenta con un microcontrolador ATMEGA328; 14 pines digitales, de los cuales 6 pueden utilizarse para generar señales PWM (esenciales para controlar servomotores), y 8 pines de entrada analógica. Su pequeño tamaño y bajo coste (se puede encontrar por 7,50 €) lo hacen ideal para proyectos compactos y económicos. Un añadido común para proyectos con actuadores es el "Extension Shield", que permite conectar una fuente de alimentación externa a la placa además de contar con más pines para alimentar a los diferentes componentes, como podrían ser los servos. En las ventajas encontramos la facilidad de uso del entorno de programación y su bajo precio. Sin embargo, el Arduino Nano carece de características avanzadas de protección, lo que puede ser un riesgo en aplicaciones más exigentes donde se pueden presentar picos de corriente o cortocircuitos. Además, el máximo que puede proporcionar de corriente uno de sus pines son 40 mA. Comparando con los servos expuestos en el apartado 4.1, esta placa sería adecuada preferentemente para el servo MG90S, ya que no requiere de ningún tipo especial de comunicación. Para el servo Dynamixel XL330-M288-T haría falta configurar la configuración TTL a través del código de la placa, lo que puede llevar a fallos entre la comunicación servo-placa.

La siguiente opción considerada fue la OPENRB-150, fabricada por la propia Robotis, fabricantes del servo Dynamixel. Entre sus especificaciones encontramos un microcontrolador SAMD21 Cortex-M0, lo que le permite manejar múltiples servos y motores simultáneamente con un control preciso y estable, 24 puertos de salida y entrada, 12 de los cuáles son capaces de producir señales PWM y 4 puertos para comunicación TTL diseñados para la comunicación con los servos Dynamixel. Estos puertos pueden llegar a proporcionar hasta 3000 Ma, expandiendo las posibilidades de acción de los actuadores. Una de sus principales ventajas es su capacidad para suministrar una corriente adecuada a varios servomotores a la vez, gracias a su diseño que incluye reguladores de voltaje y protecciones contra picos de corriente. Al mismo tiempo, es compatible con Arduino IDE, la herramienta de software libre comúnmente utilizada para programar placas de arduino, lo que facilita en gran medida la comunicación con el ordenador. La placa OPENRB-150 puede encontrar en su página web por 24.90 €.

4.3 Justificación de la solución adoptada

Expuestas las alternativas razonables que fueron planteadas al inicio del proyecto, se llega a la conclusión de escoger los siguientes dos componentes:

- Dynamixel XL330-M288-T.
- OPENRB-150.

Esta elección se debe a que es la opción más económica que al mismo tiempo ofrece grandes características y no limitan en gran cantidad el proyecto. Es más, esta combinación es recomendada por la propia fabricante ya que hay un paquete de inicio que incluye tanto el servo como la placa por 45.90 €. Como ventajas tenemos que la conexión entre ellos es sencilla y clara, y si en algún caso falla es fácil encontrar si es problemática del cableado. La placa electrónica ofrece una interfaz compatible con lo aprendido durante el grado además de ser capaz de controlar varios servos con precisión en el caso que se quisiese mejorar este proyecto. Por otro lado, el servo escogido ofrece información detallada de la posición, velocidad, corriente y PWM durante las trayectorias lo que facilita y mejora la exactitud en gran medida de los diferentes sistemas de control aplicados. En conclusión, el servo y la placa ofrecen una base clara y fácilmente expandible para hacer proyectos de robótica que requieran tener acceso a opciones más avanzadas, como pueden ser sus tolerancias al voltaje y a la corriente, en las que se requieren mejores especificaciones que las que se

pueden encontrar componentes electrónicos más básicos.

5 DESCRIPCIÓN DETALLADA DE LA SOLUCIÓN ADOPTADA

5.1 Componentes de la solución adoptada

5.1.1 Placa electrónica

Como se ha expuesto en el apartado 4.3, la placa electrónica seleccionada es la OPENRB-150. Esta placa, muy parecida en forma a un Arduino MKR, es compatible tanto con el sistema de programación Arduino IDE como con Dynamixel Wizard 2.0, el programa desarrollado por Robotis para hacer un control básico de sus servos. Tiene 4 puertos diseñados para comunicación TTL capaces de suministrar hasta 3000 mA, que gracias a su SAMD21 Cortex-M0+ 32bit low power ARM® MCU puede controlar varios servomotores de forma precisa, aunque en este proyecto solo se vaya a hacer el control de un solo actuador. Entre los pines se pueden contar 24 pines de entrada y salida digital, 12 de modulación de tensión, 7 de entrada analógica y 1 de salida analógica.

Imagen 4. Placa OPENRB-150

Para la comunicación con el ordenador la placa OPENRB-150 cuenta con un puerto Serial al que es fácil de acceder a través de un cable USB tipo C, tipo de cable comúnmente usado en Europa para cargar todo tipo de dispositivos, especialmente teléfonos móviles. Debido a que no viene ningún cable al comprar el dispositivo, se ha procedido a la compra de uno más especializado en la transmisión de datos, aunque como mencionado anteriormente la placa OPENRB-150 funciona perfectamente con un cable USB Tipo C común. El cable usado para el prototipo es de la marca Nanocable, y de características destacbales podemos encontrar una velocidad de transmisión de datos de10 GBPS y capaz de transmitir 3 amperios. Este cable es más adecuado que uno común ya que facilita la comunicación entre el ordenador y la placa. Para alimentar la placa hay dos formas posibles de hacerlo:

- Alimentación a través del cable USB conectado al ordenador, donde este proporciona la energía a través del mismo cable de comunicación, proporcionando 5 V.
- Alimentación externa, donde se puede suministrar a la placa desde 3.7 V hasta 12 V. De fábrica viene configurada por alimentación de ordenador, para poder alimentarla de manera externa se necesita poner el “jumper” de la placa en VIN(DXL).

Imagen 5. Cable USB Tipo C

Imagen 6. Características cable USB Tipo C

Por último, esta placa es capaz de soportar el control del servo directamente desde sus puertos TTL, pero para hacer un montaje seguro se usará una etapa de alimentación para evitar que picos de corriente puedan dañar la placa, más información en su sección (5.1.3).

5.1.2 Servomotor

Imagen 7. Servomotor Dynamixel XL-330-M288-T

El servomotor escogido es el Dynamixel XL330-M88-T, un servo inteligente compacto y ligero, diseñado para aplicaciones más avanzadas de robótica y automatización que otros servos de bajo coste. Este servo ofrece un control preciso de velocidad, posición y par de fuerza a través del control de la corriente. Algunas de sus características se pueden ver en la siguiente tabla:

Característica	Valor
Tipo de motor	Corriente continua de imán permanente con escobillas
Peso	18 g
Procesador	ARM CORTEX-M0+ (64 [MHz], 32Bit)
Dimensiones	20x34x26 [mm]
Par de parada	0.52 [N.m] (a 5.0 [V], 1.47 [A])
Velocidad sin carga	103 [rpm] (a 5.0 [V])
Sensor de Posición	Encoder absoluto (12 Bit, 360 º)
Resolución	4096 [pulsos/rev]
Tensión de entrada	3.7 - 6 [V]

Tabla 1. Especificaciones del servomotor

Un valor que no se especifica en la información suministrada es la constante de par (K_t), que en la sección de cálculos se demuestra que $K_t = 0.353$ N.m/A.

El sistema de comunicación de este servomotor es del tipo TTL Half Duplex serial asíncrona, lo que permite al servo recibir instrucciones y enviar datos, como pueden ser su posición, velocidad, corriente o demás valores que se necesiten en ese momento, facilitando los sistemas de control. Con la compra ser servo se incluye un cable TTL. Aquí se muestran los dos puertos del motor:

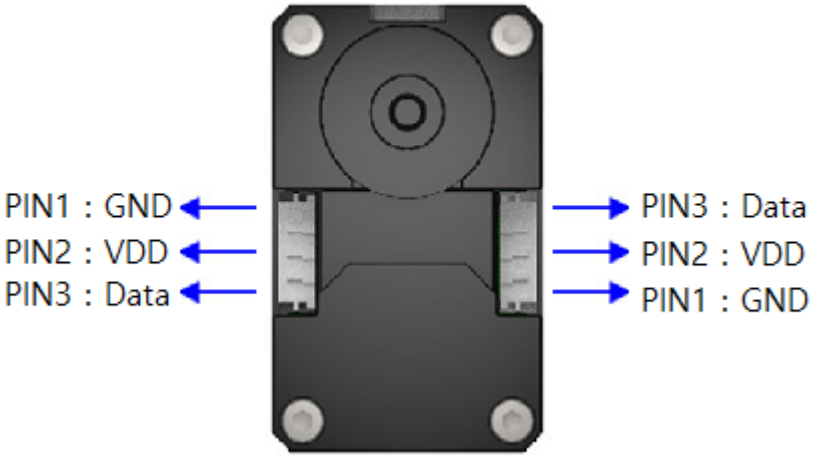


Imagen 8. Puertos TTL Servomotor XL330

Una de las utilidades más destacadas del servo es los distintos modos de control que vienen de fábrica:

- Modo de posición
- Modo de posición extendida (multi-giro)
- Modo de velocidad
- Modo de PWM (control del voltaje)
- Modo de corriente
- Modo de posi6ción por corriente

De estos modos se va a proceder a explicar más en detalle los utilizados en este proyecto.

El modo de control de posición permite al usuario especificar una posición objetivo en 360 grados y el servo controlará el giro hasta llegar al ángulo especificado. Este modo ofrece la opción de añadirle alimentación anticipada (feedforward) si el sistema requiere mejorar la respuesta dinámica (compensación de fuerzas). Además, incluye la posibilidad de ajustar los valores PID para que se ajuste mejor a la trayectoria que se desea, reduciendo el error y mejorando la estabilidad. A continuación, se incluye el esquema de control de posición:

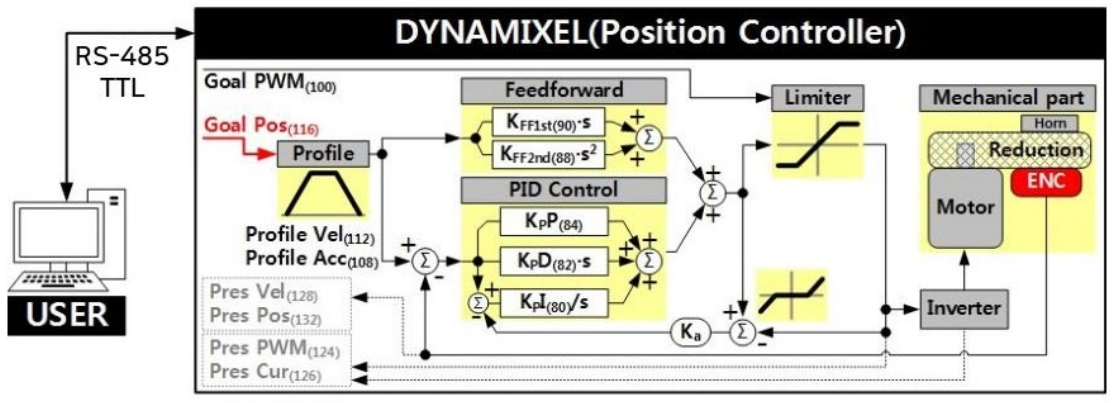


Imagen 9. Esquema del modo de posición de Dynamixel

Al mismo tiempo, el motor incluye otro modo de posición, que cuenta con el mismo esquema de control que el anterior. Este modo, llamado modo de posición extendida, se diferencia en que acepta valores superiores a 360 grados como posición objetivo. Un ejemplo claro se puede ver en el modo de posición normal, en el cuál al indicarle al servo ir a 720 grados el

motor no girará. En cambio, en la posición extendida, marcar como objetivo 720 grados hace girar al servo 2 vueltas.

El siguiente modo utilizado es el modo velocidad, donde se controla la velocidad de rotación de manera continuada. En este caso, se especifica una velocidad objetivo, y el servo ajustará su rotación para primero alcanzar esa velocidad y luego mantenerla. Este modo solo cuenta con un control PI (Proporcional e Integral) de fábrica, que ayuda minimizar el error y estabilizar la velocidad deseada, como se puede observar en su esquema de control:

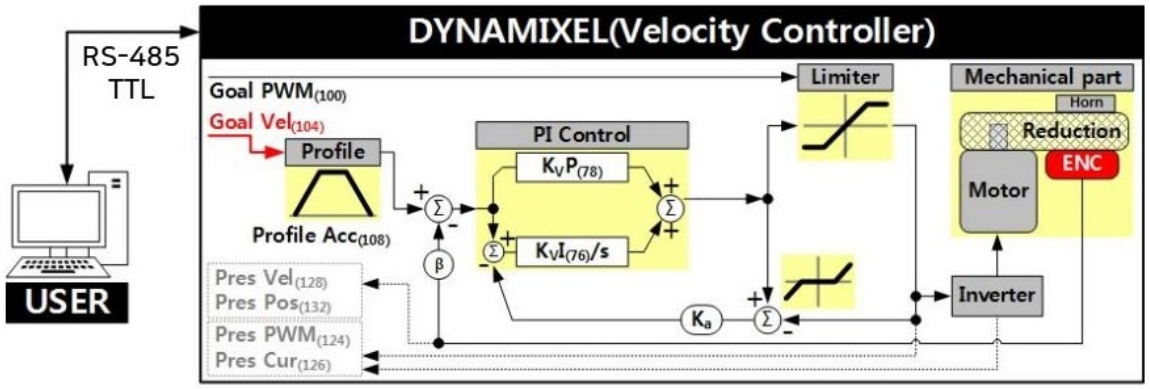


Imagen 10. Esquema del modo Velocidad de Dynamixel

Otro modo utilizado durante el trabajo es el modo PWM o modo de modulación por ancho de pulsos, que a efectos prácticos lo que hace es ajustar el voltaje. Durante este tipo de control se especifica un porcentaje de PWM objetivo (el ciclo de trabajo o “duty cycle”), por lo que el servo regulará la potencia enviada al motor, lo que a su vez afectará tanto a su velocidad como a su par de fuerza. Este modo no cuenta con un esquema de control del propio Dynamixel al no tener reguladores PID.

El último modo usado en este proyecto es el modo corriente, que permite un control del par de fuerza a través de un control de la corriente suministrada. En este modo, se especifica una corriente objetiva que el servo tiene que alcanzar y el servo ajustará su salida para mantener esa corriente, lo que a su vez significa un control preciso de la fuerza que ejerce el motor ya que la corriente está directamente relacionada con el par de fuerza. El control por corriente es el modo más utilizado en aplicaciones robóticas ya que hace un control preciso de la fuerza y es capaz de reaccionar a fuerzas externas como cambios en la carga.

Un concepto clave para entender el funcionamiento del servo Dynamixel XL-330-M88-T es las tablas de control, o más concretamente, su tabla de control. Esta tabla define sus parámetros configurables y accede a los datos de estado del servo. Seguidamente se presenta una versión simplificada de la tabla:

Parámetro		Valor
EEPROM	Drive Mode	Valores que controlan el generador de trayectorias como sentido del servo (horario, antihorario), el tiempo/velocidad de aceleración o el tiempo de la trayectoria
	Operating Mode	Valores que controlan qué modo de control sigue el servo
	Límites	Valores que dictan el límite de corriente, velocidad, pwm…

RAM	Torque Enable	Activa o desactiva el par del motor
	Ganancias PI, PID	Valores de las ganancias PI del control de Velocidad y PID del control de Posición
	Referencias de control	Valores que se intentan alcanzar en los distintos modos, es decir, los valores objetivos
	Sensores	Valores que indican que posición, velocidad, corriente detecta actualmente el servo

Tabla 2. Tabla de control simplificada del servo XL330

Entender correctamente la tabla de control es una parte importante de cómo funciona el servo inteligente. Por ejemplo, los valores que están en la parte de EEPROM son aquellos que necesitan que el servo esté desactivado para poder ser cambiados. Esto, específicamente, quiere decir que el par tiene que estar deshabilitado para modificar límites, trayectorias o el modo del control del servo, lo que provoca que el servo deje de hacer fuerza al cambiar de modos. Este caso ocurre al cambiar cualquier valor que esté guardado en el EEPROM del servo, por lo que no es posible si se quiere modificar algunos de esos valores en medio de una trayectoria, como sí se puede hacer con los valores PID, con los valores objetivos u otros casos de valores en el RAM.

5.1.3 Etapa de alimentación

Para la etapa de alimentación del prototipo se ha seleccionado la placa “U2D2 Power Hub” para solventar que corrientes picos puedan dañar de algún modo la placa OPENRB-150.

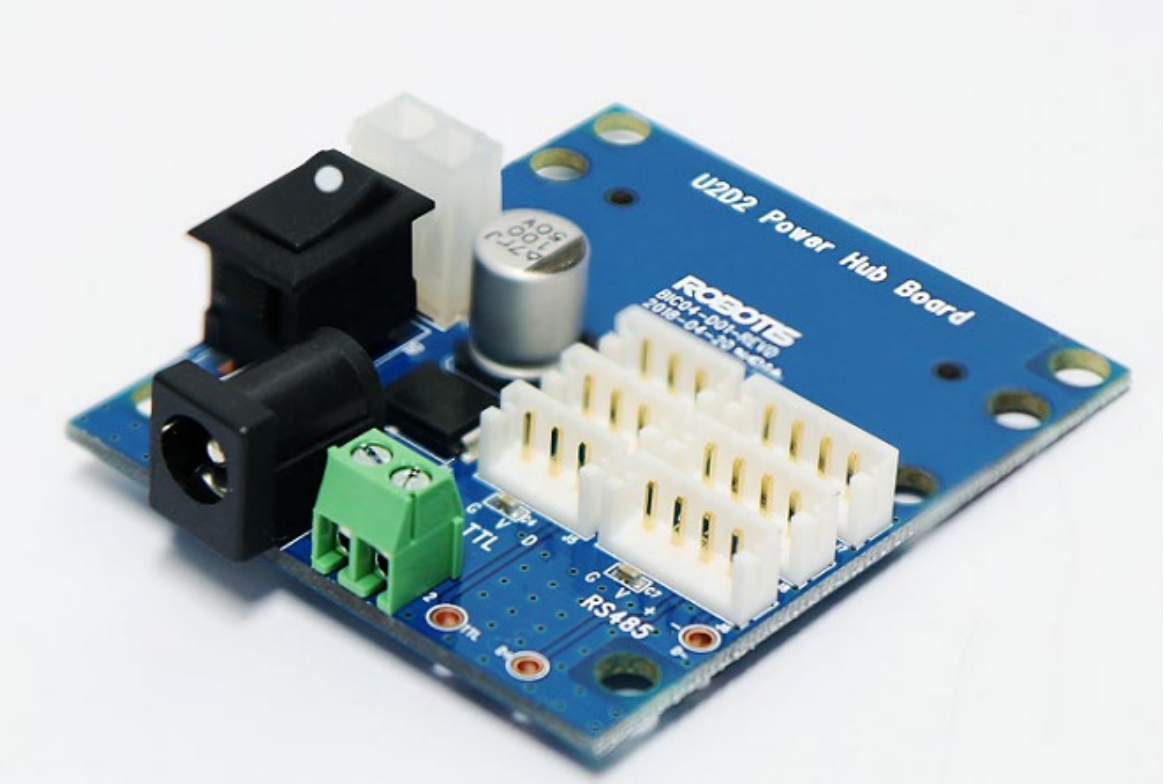


Imagen 11. Placa U2D2 Power Hub

Esta placa actúa como adaptador entre varios servos y asegura que la alimentación sea repartida correctamente entre todos los servos conectados. Como se puede observar en la

imagen 11, cuenta con varios puertos TTL y RS485, aunque en este proyecto solo nos interesan los primeros. En el paquete de la placa se incluyen cables TTL y RS485.

En general, su capacidad para simplificar la alimentación, junto con su robusta capacidad de comunicación lo convierten en una buena herramienta ya que gracias a su interruptor incorporado es muy sencillo cortar la alimentación al servo de manera segura. Para suministrar la placa se contemplan varias opciones, como alimentarla por la terminal de PCB o a través del "SMPS DC Connector" (Switched-Mode Power Supply, conector diseñado para conectar una fuente de alimentación a un dispositivo de corriente continua).

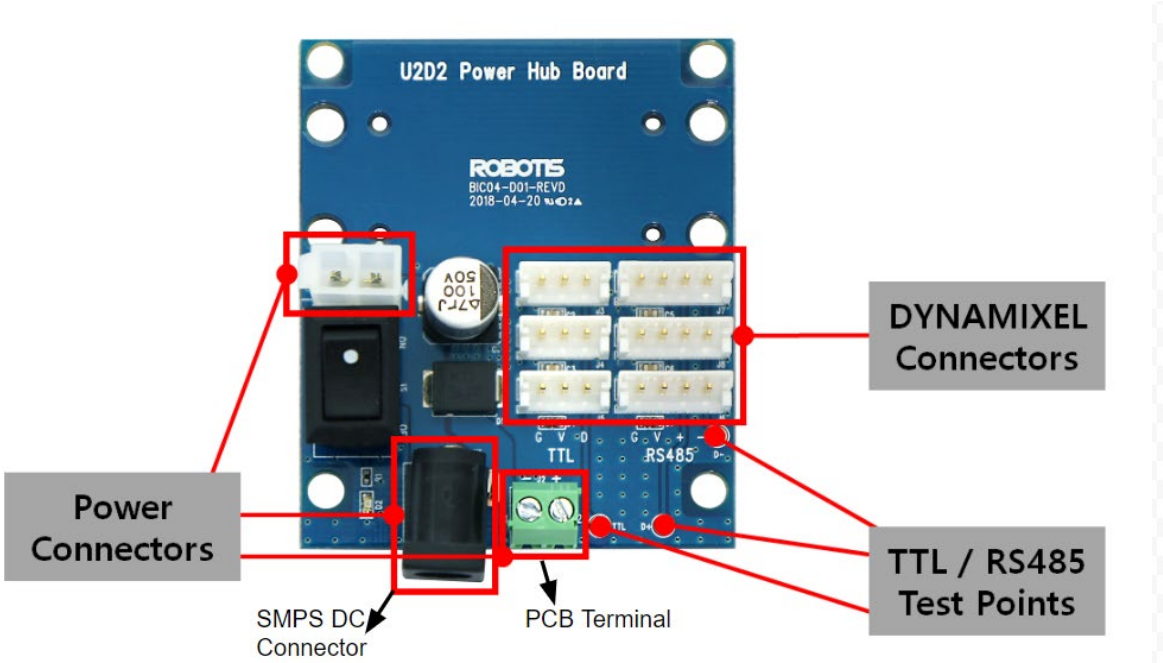


Imagen 12. Puertos alimentación Placa U2D2 Power Hub

Después de varias pruebas con la terminal PCB, que resultaron en error, se escogió usar el SMPS DC Connector. Este conector tiene unas dimensiones de 2.5 mm ID / 5.5 mm OD, que significan que tiene un diámetro interior de 2.5 mm y uno exterior de 5.5 mm, esto genera el problema ya que la mayoría de cables de conexión encontrados son de 2.1 mm ID / 5.5 mm OD. Esto se ha solucionado con un adaptador de 2.1 mm a 2.5 mm, es decir, recibe un cable con terminal 2.1 mm y lo convierte en 2.5 mm, como el mostrado en la imagen:

Imagen 13. Adpatador de terminal a 2.1 mm a 2.5 mm

5.1.4 Fuente de alimentación

Para la fuente de alimentación se ha escogido un convertidor de corriente alterna a corriente continua que tiene de salida 5 V y 2 A para cumplir con las especificaciones de alimentación del servomotor. Este tipo de convertidor es muy común para móviles, por lo que se usó uno que se tenía disponible:

Imagen 14. Fuente de alimentación usada

Imagen 15. Características de la fuente de alimentación usada

Estos tipos de adaptadores tienen salida para un cable USB, por lo que se compró un cable USB a Jack de alimentación 5.5 x 2.1 mm, como pone en su etiqueta:

Imagen 16. Cable de alimentación

Imagen 17. Características y precio del cable de alimentación

5.1.5 Peso de la articulación

Debido a que en este proyecto se hace una compensación de la gravedad a través del control, es necesario una masa capaz de hacer un esfuerzo al servo. Debido a este motivo se ha escogido este tipo de placa metálica como masa extra del eslabón:

Imagen 18. Placa metálica que actúa como peso de la articulación

Cada una de estas placas pesa de 20-26 gramos y mide 97 mm de largo, 19 mm de ancho y 2 mm de grosor. Con 5 de estas placas se consiguen alrededor de 130 gramos, un peso considerable para el actuador. Esta pieza tiene de referencia en Leroy Merlín 12632060 o según su etiqueta 8413023073555, en caso de que se necesite consultar otras de sus características.

Estas placas se han escogido debido a que eran un tipo de peso que se podía acumular poco a poco en la articulación para ir haciendo pruebas de cuanto se podías soportar. Al mismo tiempo, incluye agujeros por lo que es más sencillo asegurarlas a las piezas y guardarlas cuando se vea conveniente.

5.1.6 Tornillos

Debido a la necesidad de sujetar los diferentes componentes electrónicos en las dos posiciones que tiene que tener la herramienta se requiere de un método de sujeción. Para asegurar la estructura se optan por tornillos ya que en case de necesitar hacer algún cambio

de componente estos permiten ensamblar y desmontar la herramienta con relativa facilidad. La lista de tornillería utilizada es:

- Tornillos de M2 x 6 mm terminados en punta.
- Tornillo de M2.6 x 12 mm terminados en punta.
- Tornillos de M3 x 6 mm terminados en punta.
- Tornillos de M3 x 10 mm terminados en rosca.
- Tornillos de M3 x 18 mm terminados en punta.
- Tornillos de M3 x 20 mm terminados en rosca, con su correspondiente tuerca.
- Tornillos de M6 x 18 mm terminados en rosca, con su correspondiente tuerca.

5.2 Programas utilizados

Durante este trabajo se ha trabajado principalmente con tres programas: Matlab, Arduino IDE y SolidWorks. En este apartado se realiza una breve introducción a cada programa y cuál es su utilidad principal para el proyecto.

5.2.1 Matlab

Matlab es una plataforma de desarrollo y lenguaje de programación utilizada principalmente por ingenieros y científicos para el análisis de datos y creación de modelos y algoritmos. Se destaca por su capacidad para realizar cálculos avanzados y el análisis de grandes cantidades de datos, simulaciones y modelos. Además, la interfaz para controlar el servo de se ha creado a través de Matlab App Designer, una herramienta dentro del entorno de Matlab que permite diseñar y crear aplicaciones gráficas con la capacidad de computación de Matlab para matemáticas avanzadas. Esto lo convierte en una poderosa herramienta para la creación de programas e interfaces de control, razón por la que ha sido escogida:

5.2.2 Arduino IDE

Arduino IDE es un entorno de programación diseñado específicamente para programar placas Arduino. Con una aplicación sencilla de usar, su código es abierto lo que simplifica la escritura, compilación y carga de programas en las placas compatibles, como la OPENRB-150. Este entorno permite a los usuarios crear proyectos interactivos que involucren sensores, actuadores y otros dispositivos electrónicos mediante una interfaz gráfica intuitiva. Además, Arduino IDE utiliza un lenguaje de programación derivado de C/C++ simplificado para facilitar la programación. Se ha escogido esta aplicación ya que en el pasado ya se habían varios programas a través de Arduino durante el grado y se tenía un entendimiento grande del entorno.

Además de las librerías bases de Arduino, se ha utilizado la librería “Dynamixel2Arduino”, una librería desarrollada para facilitar la comunicación entre placas compatibles con el Arduino y los servomotores Dynamixel. Entre las funciones más destacadas encontramos las de marcar un valor objetivo y las de leer el valor actual del servo:

- setGoalPosition(): Ordena al servo a ir a la posición objetivo y detenerse cuando la detecte.
- getPresentPosition(): Lee el valor actual de la posición.
- setGoalVelocity(): Ordena al servo a alcanzar la velocidad objetivo y mantenerse en ella.
- getPresentVelocity(): Lee el valor actual de la velocidad.
- setGoalPWM(): Ordena al servo a alcanzar el PWM indicado y mantenerse en él. En el caso que no lo alcance irá aumentando la corriente del servo provocando picos peligrosos de intensidad.
- getPresentPWM (): Devuelve el valor actual del PWM
- setGoalCurrent(): Ordena al servo a alcanzar la corriente objetivo. Esta forma es más

seguro ya que mantiene la corriente entre los valores marcados y no crea picos de corriente.

- getPresentCurrent(): Lee el valor actual de la corriente del servo

5.2.3 SolidWorks

SolidWorks es un software de Diseño Asistido por Computadora (CAD) utilizado para el diseño 3D, planos 2D y el análisis de productos. Es conocido por su robustez y capacidad para crear modelos detallados de piezas y ensamblajes en el ámbito de la ingeniería mecánica y el diseño industrial. SolidWorks ofrece herramientas avanzadas para la creación de dibujos técnicos, pruebas de ajuste y función, y renderizado de alta calidad.

Para este proyecto se ha tenido que aprender de forma autodidacta como diseñar elementos con esta aplicación, pero su accesible interfaz y múltiples facilidades para crear piezas la convierten en una herramienta poderosa para crear piezas 3D, como se ha hecho en este proyecto. Además, su capacidad para calcular las inercias de los productos diseñados, herramienta que se ha usado en este proyecto, facilitan cálculos que pueden llegar a ser complicados.

5.3 Diseño de las piezas

El diseño final del prototipo es un cubo de piezas de metacrilato de 24 cm de largo, 24 cm de ancho y 14 cm de alto. Estas dimensiones vienen dadas debido a que la herramienta ha de poder estar en dos posiciones: posición sin gravedad (o horizontal) y posición con gravedad (o vertical). La diferencia entre estas dos posiciones es si le afecta o no la gravedad al servo, es decir, si en la trayectoria de giro del servo ha de compensar la fuerza gravitatoria. Como el prototipo ha de cambiar de una colocación a otra, todas las piezas han de estar bien sujetadas, razón por la cuál es necesario atornillar los componentes electrónicos. En las siguientes imágenes se puede observar el montaje final del diseño en sus dos posiciones:

Imagen 19. Prototipo en pocición horizontal

Imagen 20. Prototipo en posición vertical

5.3.1 Base de OPENRB-150

Esta base tiene como principal objetivo sujetar la placa OPENRB-150 a la base del prototipo. Es necesario diseñar esta pieza ya que por debajo de la placa OPEN hay una serie de pines que impiden directamente atornillarla a la base. La pieza 3D hecha con material HIPS (filamentos) cuenta con 4 huecos para poder atornillar la placa con tornillos de M2*6 mm, a la vez que cuenta con otros cuatro agujeros pasantes situados más en el exterior de la pieza para poder atornillar esta base de OPENRB-150 a la base del prototipo con tornillería de M3*10 mm.

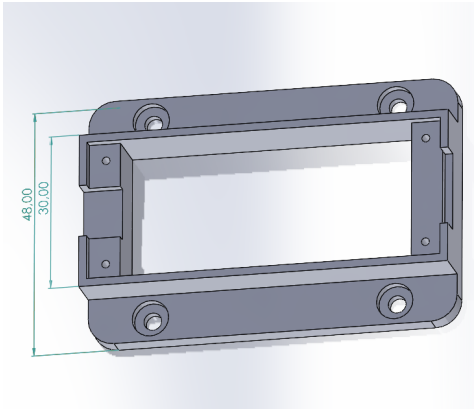
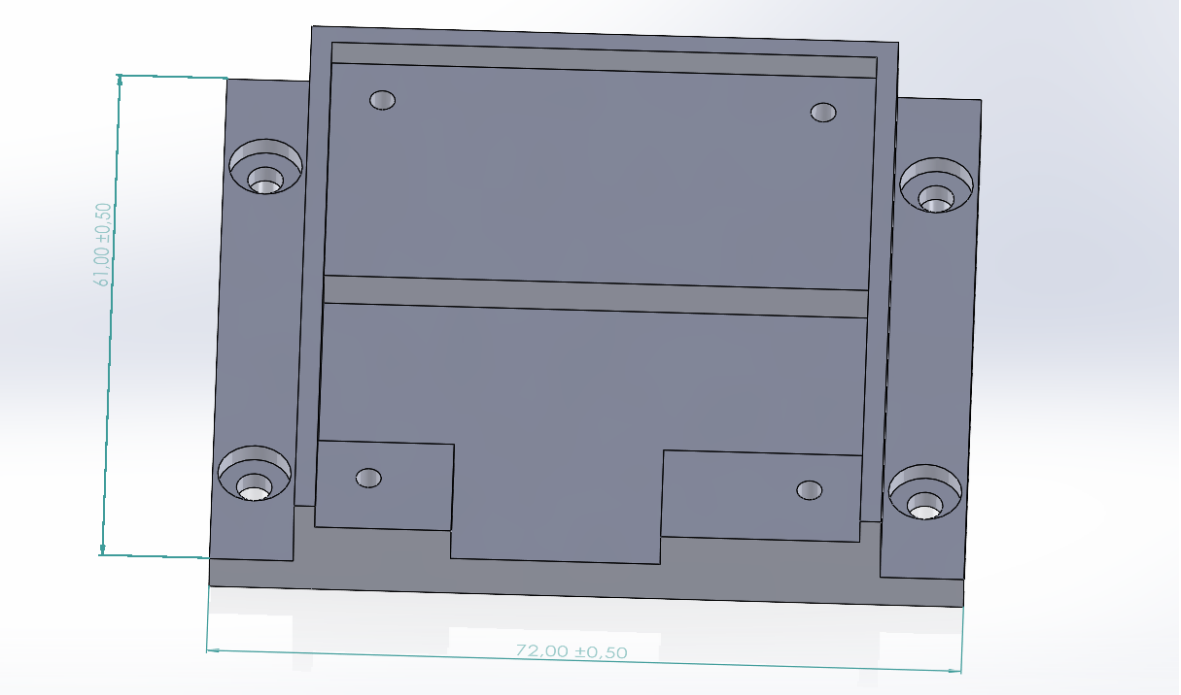


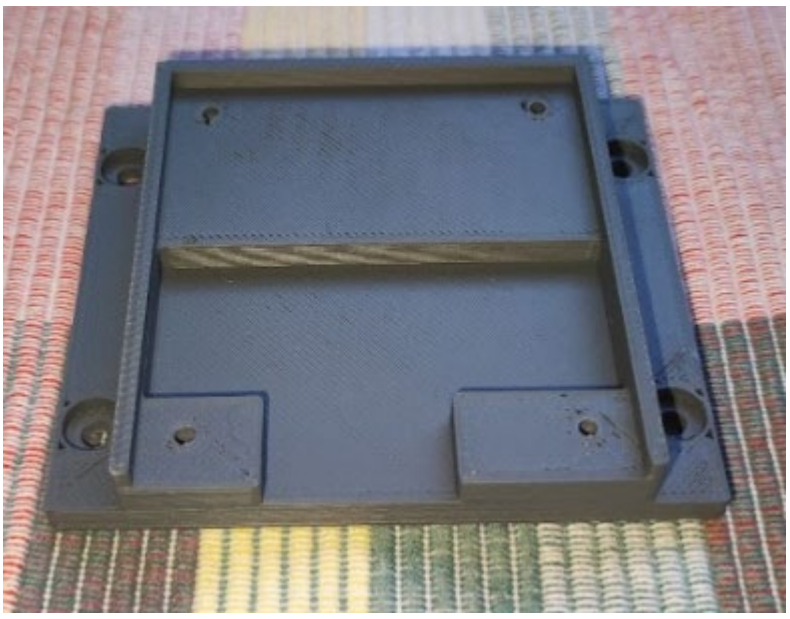
Imagen 21. Modelo en Solidworks de la pieza "Base OPENRB-150"

Imagen 22. Pieza impresa "Base OPENRB-150"

5.3.2 Base de U2D2 Power Hub

Esta base, como la pieza anterior, tiene como principal objetivo sujetar la placa U2D2 Power Hub a la base del prototipo. Ocurre el mismo caso que con el componente anterior, esta pieza es necesaria ya que en la parte inferior de la placa hay unos salientes/pines que impiden atornillarla directamente a la base. La pieza 3D hecha con material HIPS (“High Impact Polystyrene”) cuenta con 4 huecos para poder atornillar la placa con tornillos de M3*6 mm, a la vez que cuenta con otros cuatro agujeros pasantes más exterirores para poder enroscar tornillos de M3*10 mm a la base del prototipo.





5.3.3 Base del prototipo

El diseño de las paredes del prototipo es muy similar entre ellos ya que son la estructura principal de la herramienta. Empezando por la base, esta pieza se coloca en la parte más baja del prototipo y servirá como soporte. A los lados de la pieza se pueden observar unos salientes cuadrados que sirven para encajar todas las piezas como un rompecabezas. En la misma, se pueden observar dos tipos de agujeros: los agujeros redondos de métrica 3 para sujetar las bases de la electrónica, y los agujeros cuadrados para poder encajar la pared de carga que sujeta el servo y el peso de su articulación. Esta pieza, y todas las denominadas como paredes, están hechas de metacrilato y fueron cortadas por corte láser.

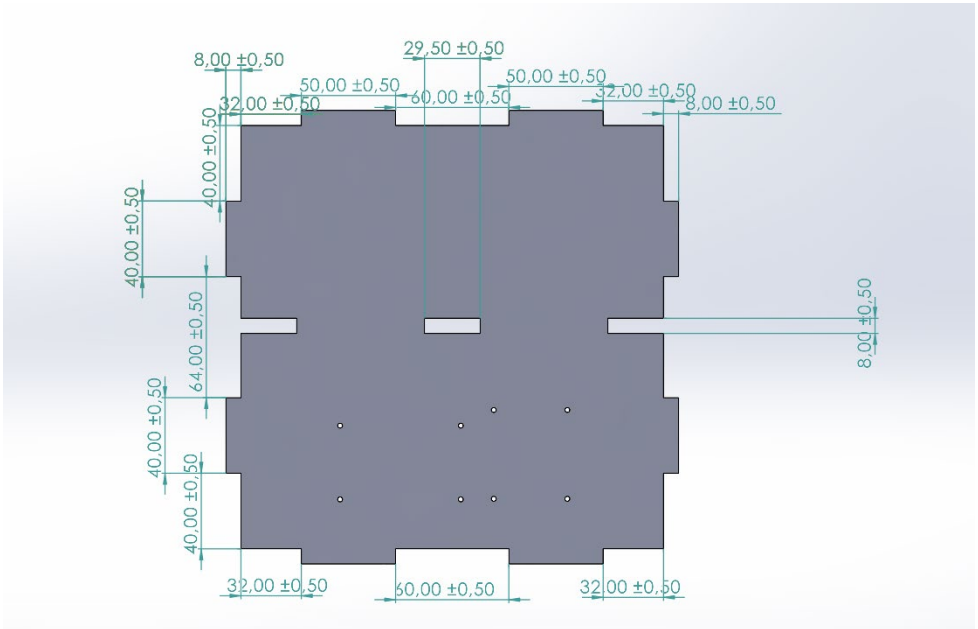


Imagen 23. "Base del prototipo" en Solidworks

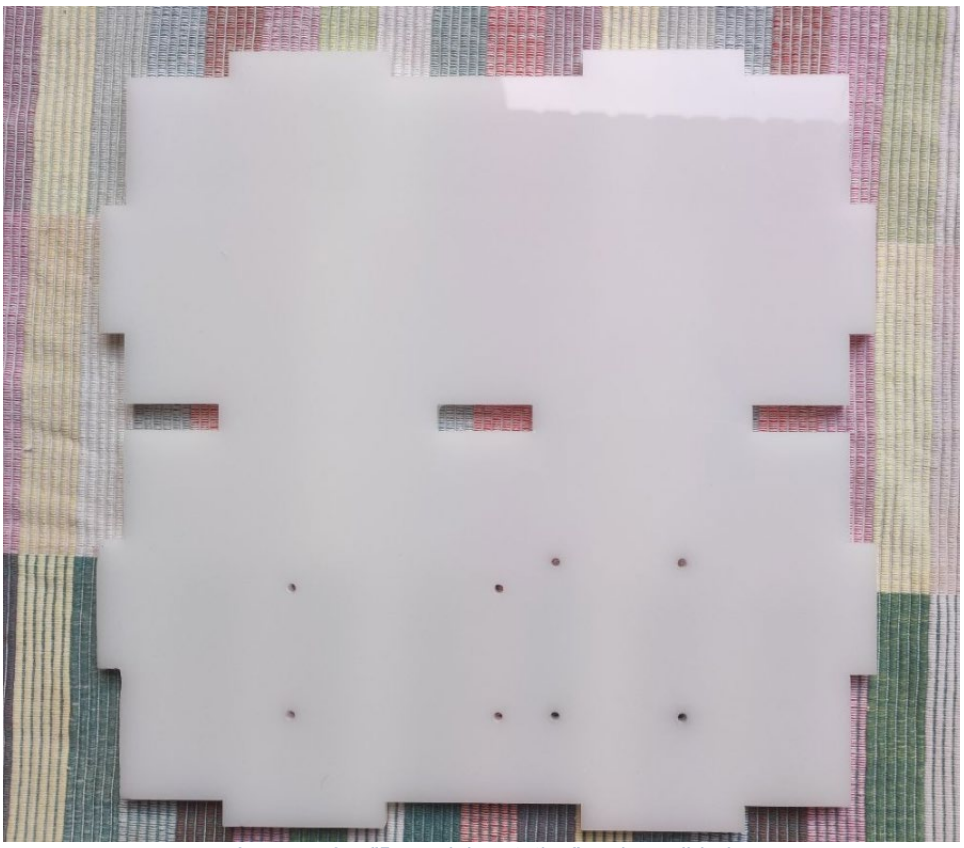


Imagen 24. "Base del prototipo" en la realidad

5.3.4 Pared de carga

La pared de carga se puede considerar la parte más crítica de la estructura. Esto se debe a que cruza por la mitad de casi todas las otras paredes, por lo que un desajuste o mal cálculo de los agujeros se puede notar a simple vista. Al mismo tiempo, actúa como soporte del servo gracias a la entrada cuadrada que se puede observar en la parte superior de la pieza y además sujeta la base del servomotor gracias a los dos agujeros redondos en medio de la pieza. Estos agujeros, de 6.8 mm de diámetro, están hechos con amplitud para el tornillo que debe colocarse en esa posición (métrica 6) pueda servir como corrector para los desajustes que puedan surgir en el montaje de la herramienta, o para las desigualdades que ocurran porque los planos de otras piezas no son correctos, como ocurrió en el desarrollo de este proyecto, que, aunque los planos dados por el fabricante marcasen unas medidas, luego en la vida real había unas tolerancias considerables.

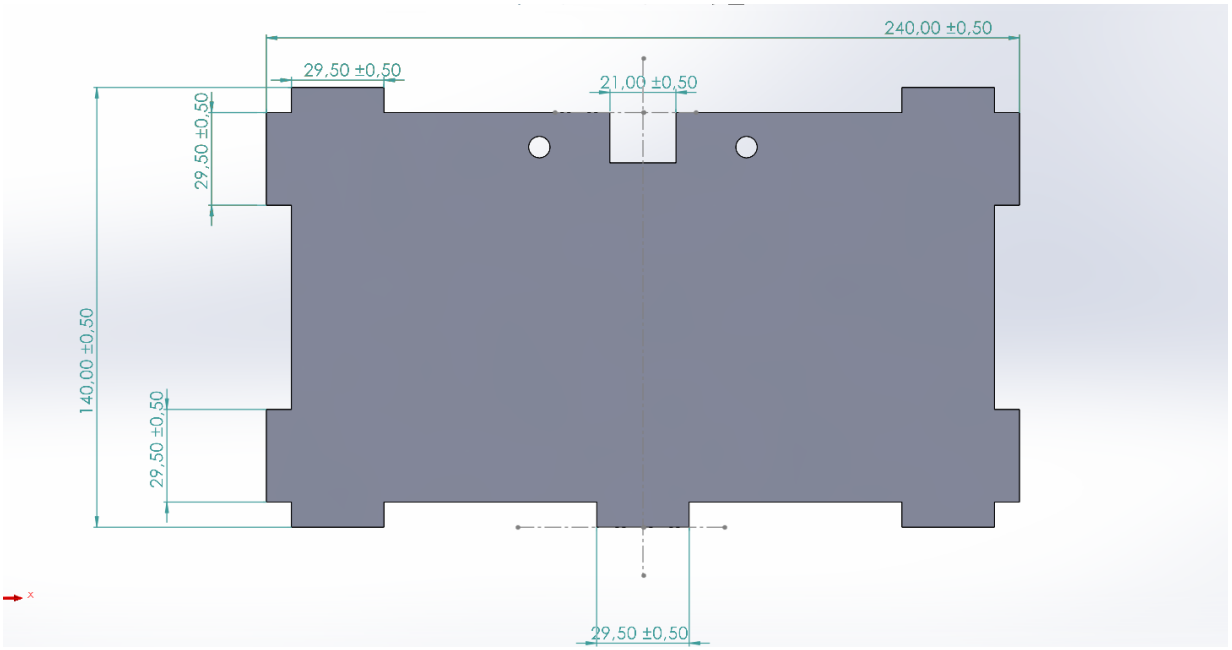


Imagen 25. Modelo la "Pared de Carga" en Solidworks

Imagen 26. "Pared de carga" en la realidad

5.3.5 Pared Izquierda/Derecha

En el diseño del prototipo, las paredes izquierda y derecha son idénticas y pueden ser intercambiadas a voluntad. Estas piezas se colocan a los lados de la estructura y es indiferente en qué lado se colocan ya que el diseño del prototipo es simétrico y los encajes son los mismo tanto por la parte superior como inferior. Al mismo tiempo, hay que tener en cuenta que al encajarlas con las demás, el saliente que sobresale a los lados ha de ir en el lado contrario de las placas.

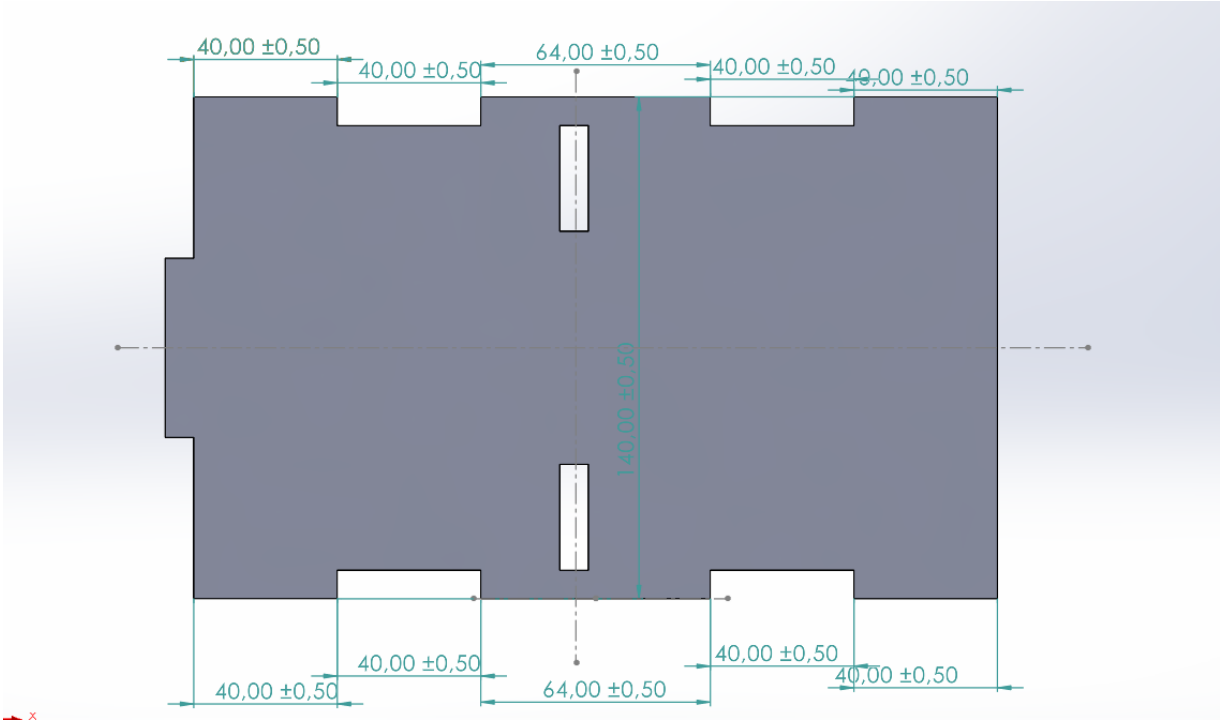


Imagen 27. Pared Izq/Der diseñada en Solidworks

Imagen 28. Pared Izq/Der en la realidad

5.3.6 Pared Fondo

El diseño de esta pieza es realmente sencillo ya que su única función es cerrar la estructura del cubo, por lo que es la única pieza de metacrilato que no se engancha con la pared de carga. La “Pared Fondo” Se coloca en el lado contrario de los componentes electrónicos para no impedir el paso de los cables.

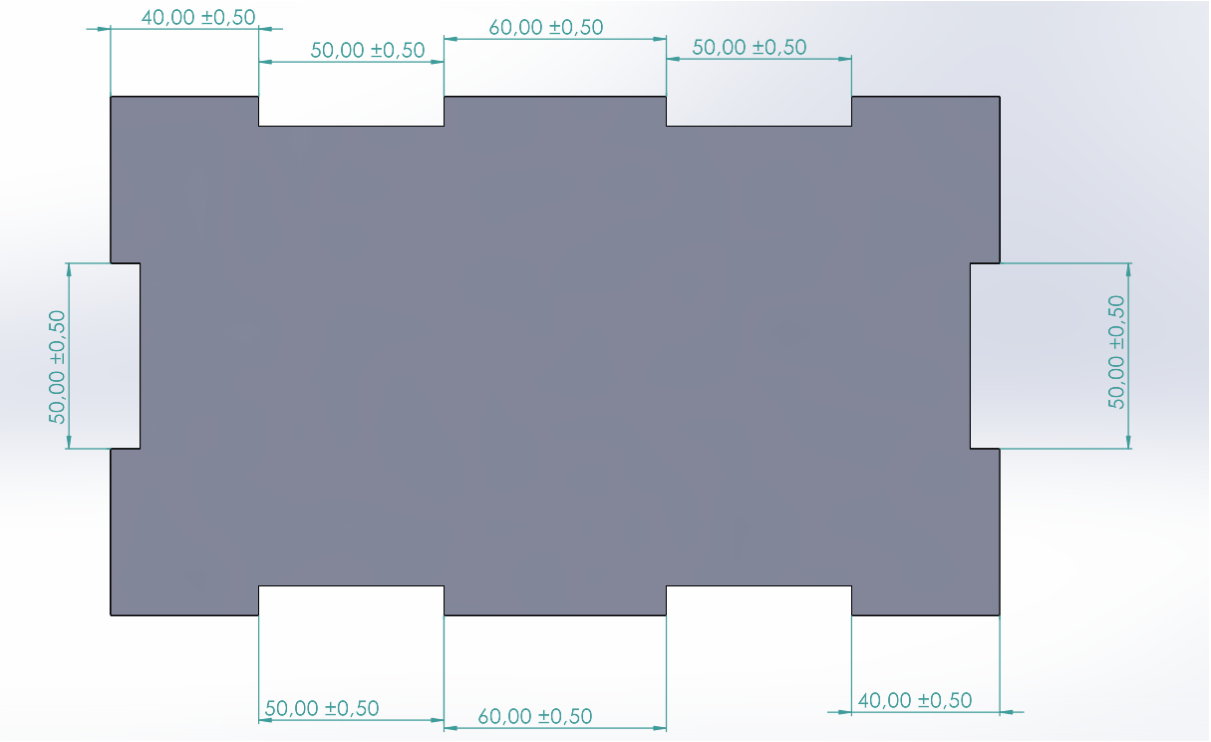


Imagen 29. Modelo "Pared Fondo" Solidworks

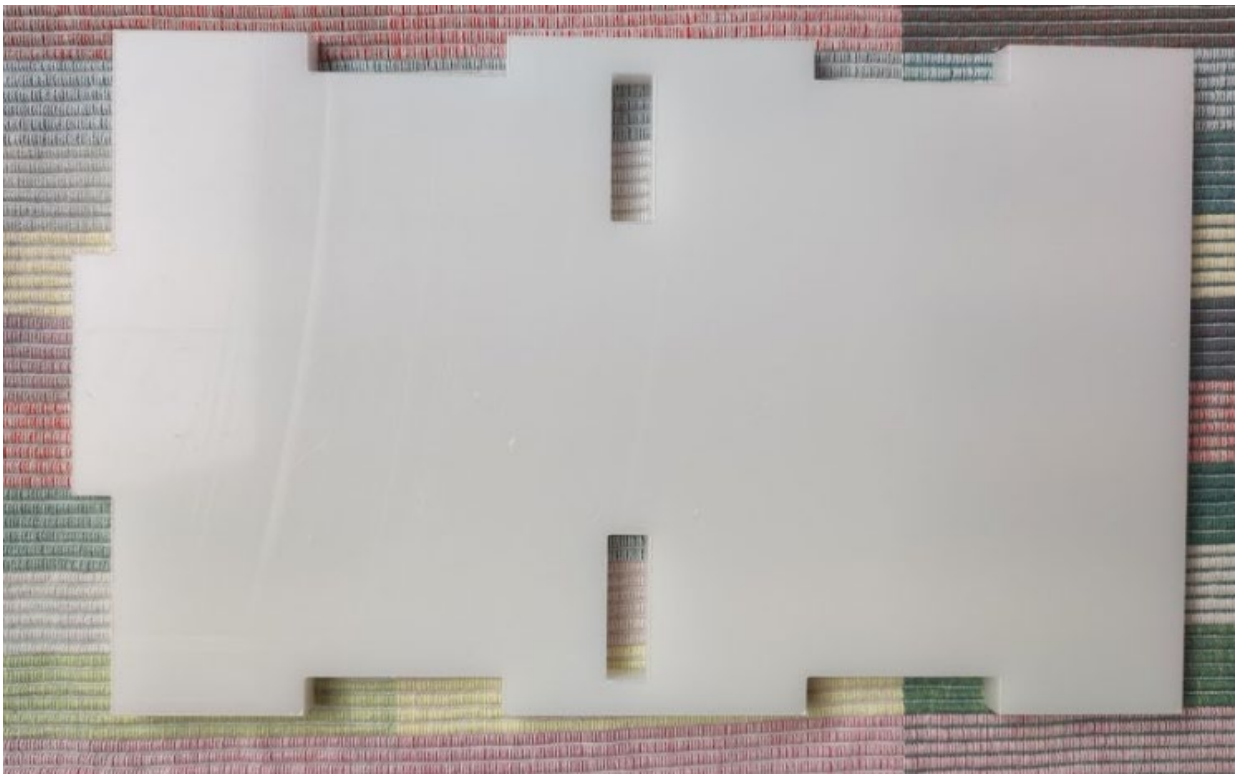


Imagen 30. "Pared Fondo" en la realidad

5.3.7 Base del Servomotor

Esta pieza 3D está hecha por HIPS y sirve para asegurar la posición del servo y fijarlo sin importar cuanta fuerza pueda estar ejerciendo el motor. Cuenta con dos agujeros pasantes de 9 mm de diámetro para conectarlo con la pared de carga y asegurar al servo en su hueco. Además, incluye cuatro agujeros en la parte superior de la pieza que tienen la función de conectar esta pieza con la parte superior de la estructura para ayudar la fijación de otra pieza

3D. Esta base del servomotor actúa como apoyo para el motor y fue diseñada sin tener en cuenta las tolerancias de las otras piezas con el objetivo de poder ajustar y fijar el servomecanismo lo máximo posible a través de la tornillería.

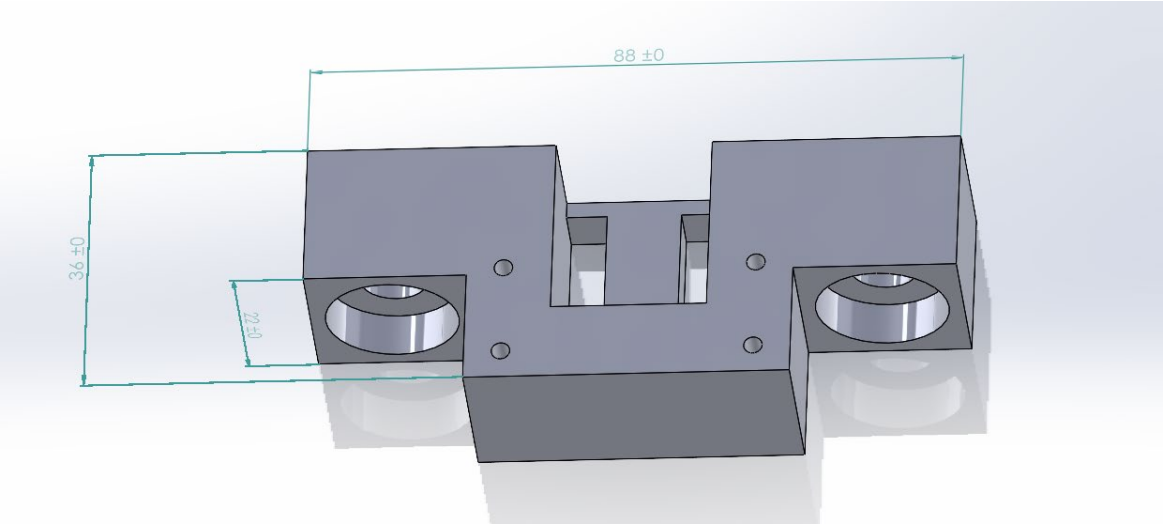


Imagen 31. "Base del Servomotor" modelada

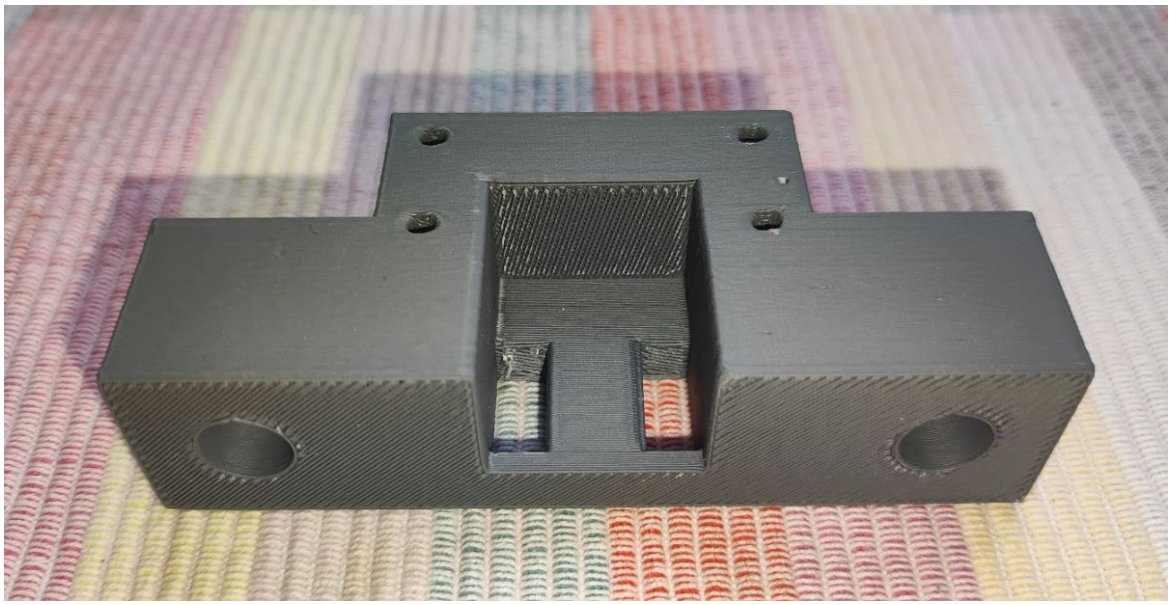


Imagen 32. "Base del Servomotor" impresa

5.3.8 Tapa

La última pieza de metacrilato es la tapa, la cual presenta un diseño muy parecido a la base de la estructura. Esta es la última pared de la estructura que debe colocarse a la pared de carga ya que tiene que encajar a la perfección con todas las demás. Se puede observar un rectángulo en el medio de la pieza que es dónde se colocará el servomotor. Al mismo tiempo, en las cercanías de ese hueco cuadrado se pueden observar 4 agujeros de métrica 3.4, que sirven para fijar el asegurador del servomotor.

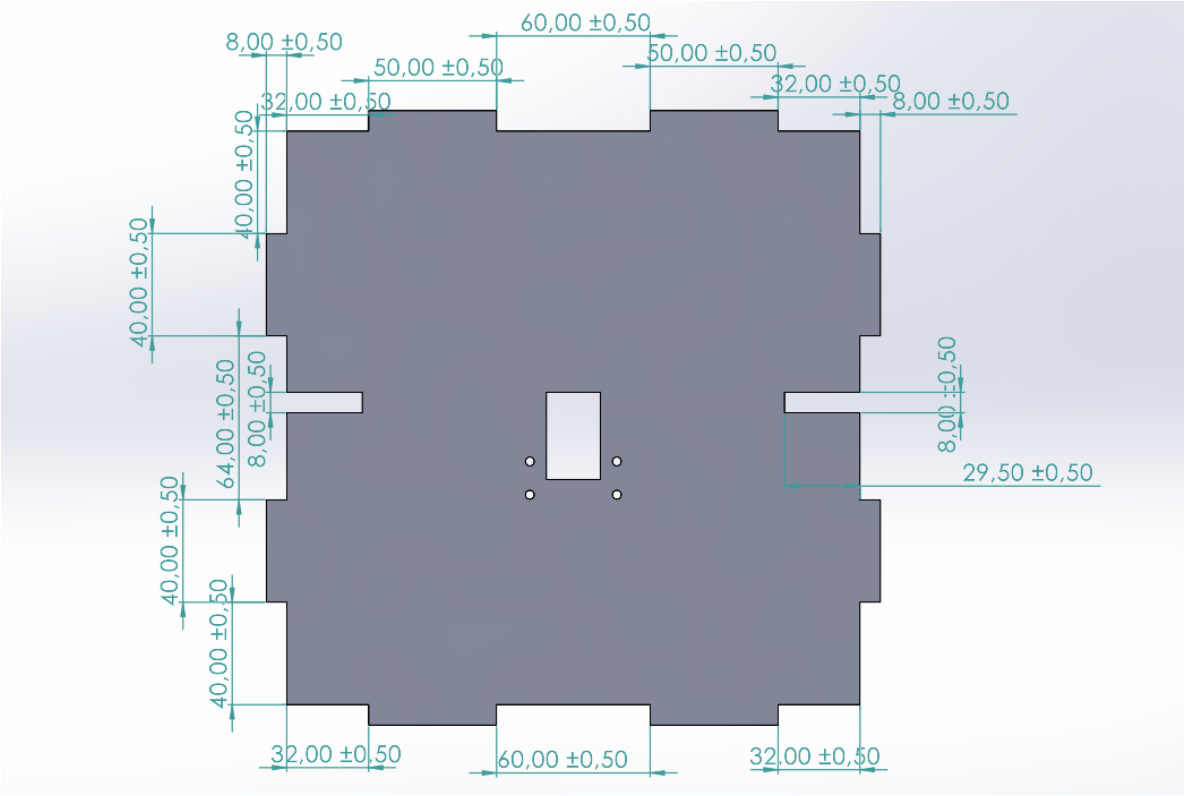


Imagen 33. "Tapa" en Solidworks

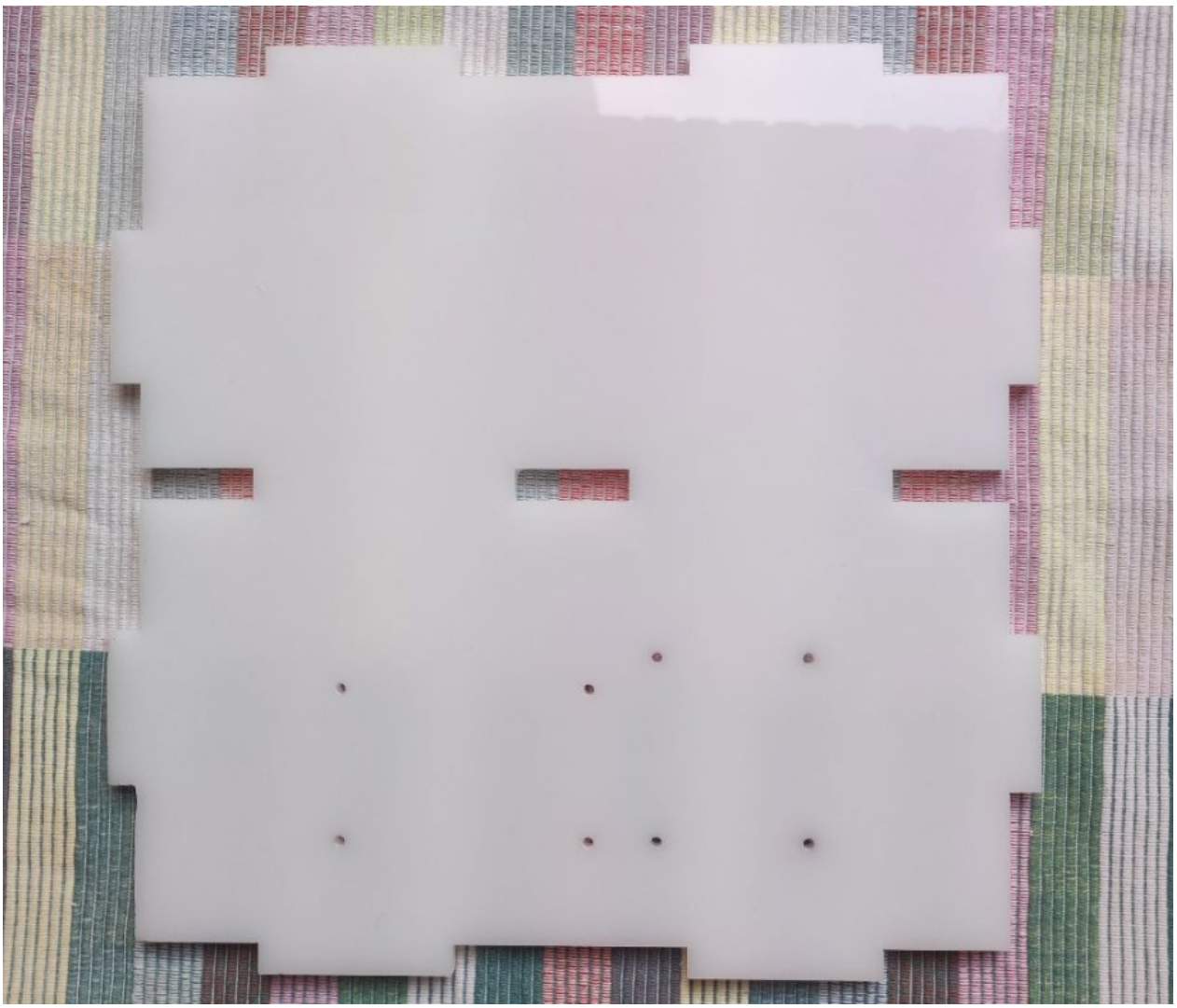


Imagen 34. "Tapa" en la realidad

5.3.9 Asegurador del Servomotor

Está pieza 3D, cuyo material es ABS (siglas de “Acrylonitrile Butadiene Styrene”), tiene la función de asegurar el servo a la herramienta en los casos que le afecte la gravedad, ya que al girar la estructura y cambiar la base a una de las paredes existe el riesgo que debido al peso de la articulación el motor ceda ante su inercia. Existiendo esta posibilidad, se ha fijado el servomotor por todos sus sectores, y esta pieza asegura al actuador gracias a sus agujeros pasantes de 3.4 mm que permiten clavar tornillos en la pieza base del servomotor.

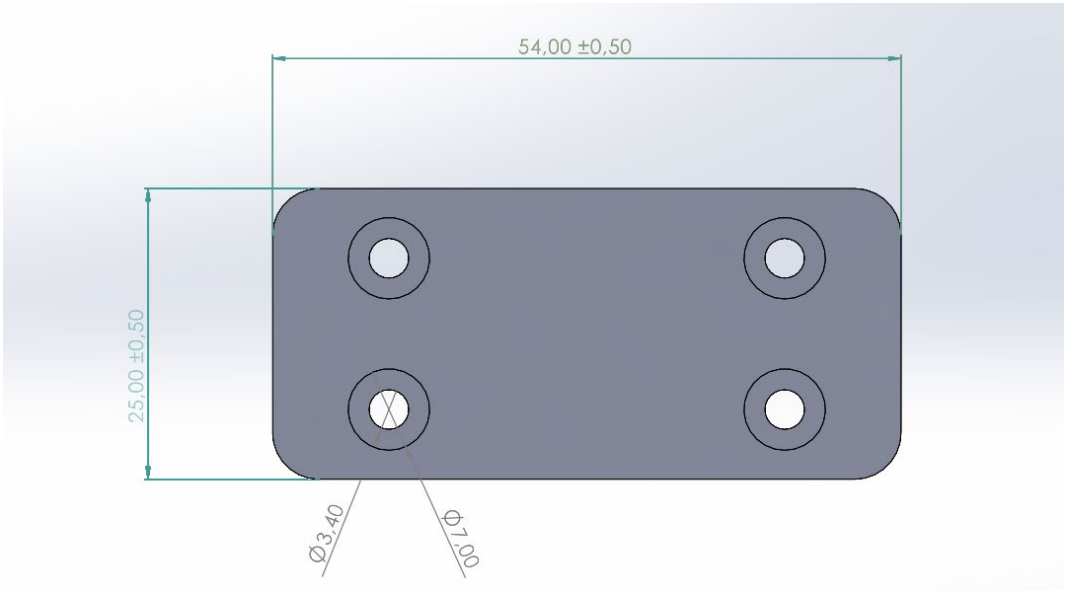


Imagen 35. Pieza "Asegurador del Servo" en Solidworks

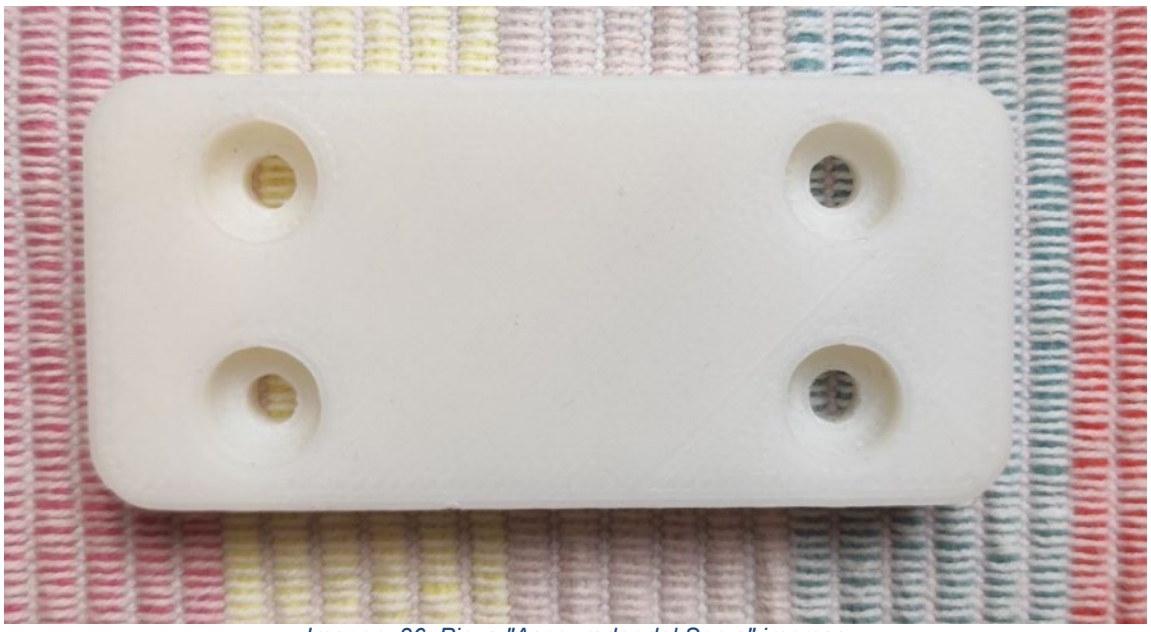


Imagen 36. Pieza "Asegurador del Servo" impresa

5.3.10 Eslabón/Articulación

En las primeras instancias del proyecto, se fijó la medida de este eslabón en alrededor de 10 cm debido a que era una longitud apreciable sin suponer demasiada carga en el servomotor, además que tenía que tener una forma rectangular y alargada para que quedase claro el funcionamiento de compensación de gravedad.

Fijar la longitud de 10 cm y la forma significa una condición de diseño debido a que si la articulación ha de girar con un radio de 10 cm ya sea en vertical o horizontal, obliga a que la estructura del proyecto, o al menos la base sobre la que se pone la articulación, ha de ser mínimo el doble de su radio. Con estas especificaciones, la siguiente dificultad surge en que las piezas de impresión 3D resultan muy ligeras y no llegan a imponer un esfuerzo en el servo, por lo que se ha de usar un peso como complemento de la articulación, que son las placas metálicas del apartado 5.1.5.

Al observar la pieza, se observan dos perforaciones en el hueco rectangular del eslabón. Estos agujeros sirven para asegurar las placas metálicas al eslabón con tornillos M3 y que no deslicen durante las trayectorias. Por otro lado, para asegurar el eslabón al servo se ha usado un tornillo de métrica 2.6 mm y de largo 12 mm. Este tornillo es imprescindible ya que el agujero del sistema de rotación del Dynamixel XL330 solo funciona con métrica 2.6 mm, no acepta 2.5 mm ya que no asegura bien la articulación. Al mismo tiempo, para facilitar la rotación del eslabón y que esté sincronizado con la rotación del servo, se han diseñado 4 salientes redondos de 1.4 mm de radio y 2 mm de largo para introducirlos en los agujeros de la rueda del servomotor. Debido al poco grosor de estos salientes existe el peligro de que estos se rompan/cedan ante el peso, por lo que en el diseño final estos han sido sustituidos por clavos de metal con sus mismas dimensiones.

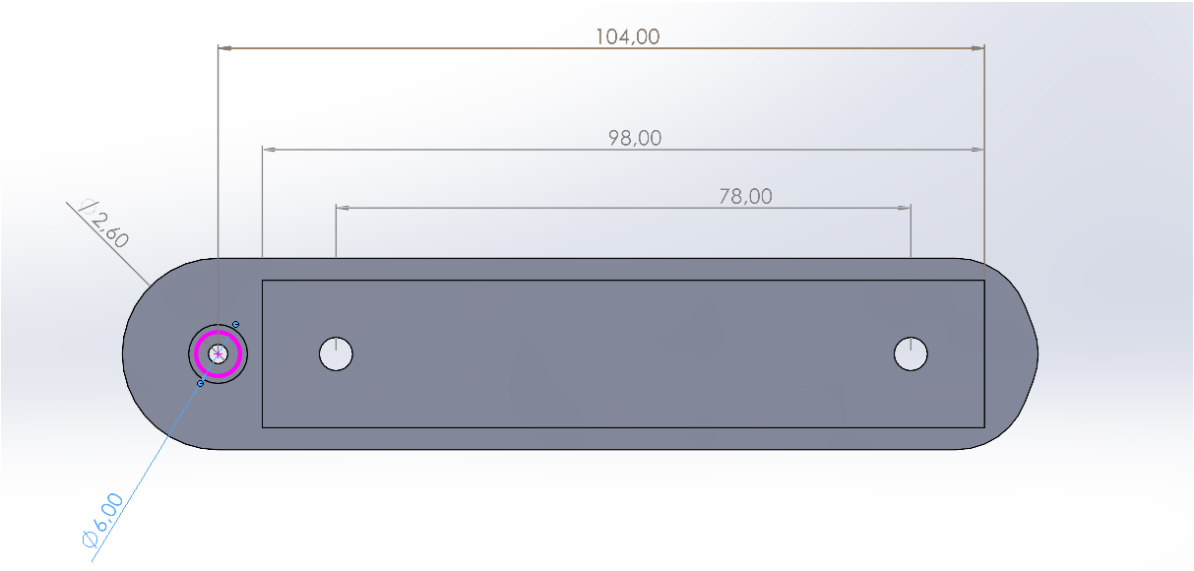


Imagen 37. Vista superior de la pieza en Solidworks

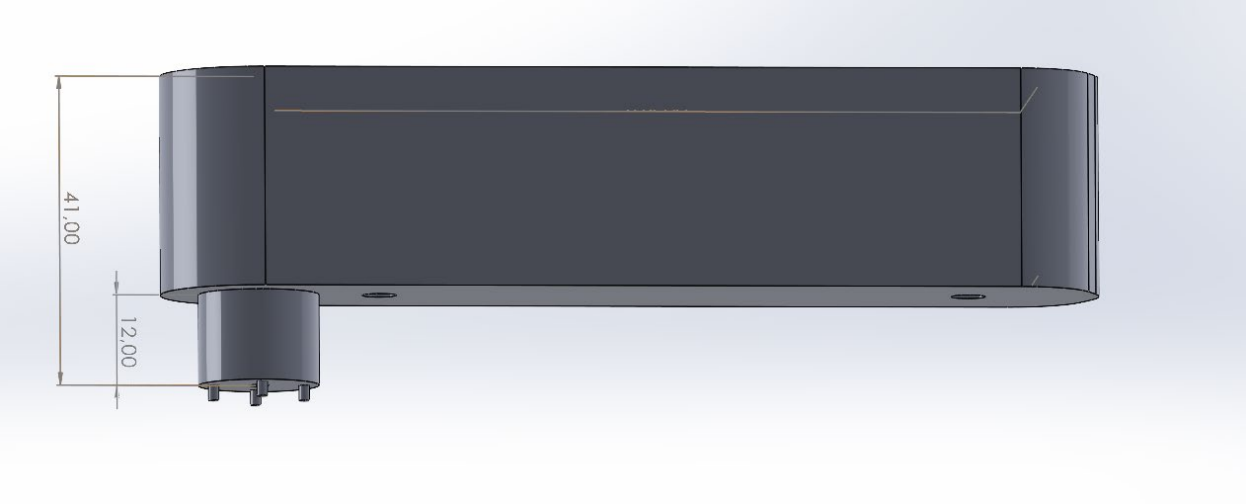


Imagen 38. Vista lateral de la pieza en Solidworks

Imagen 39. Articulación con el tornillo encajado y los salientes de metal

5.4 Modos de la herramienta

En este apartado se va a proceder a explicar los diferentes modos de configuración del servomotor que se pueden acceder a través de la aplicación desarrollada en Matlab App Designer. La herramienta ha sido configurada tal que su posición inicial es 0 º y un aumento de su posición se corresponde en una rotación en sentido contrario a las agujas del reloj. Se ha declarado de esta forma al ser un sistema de referencia bastante extendido y usado en varias asignaturas durante el grado.

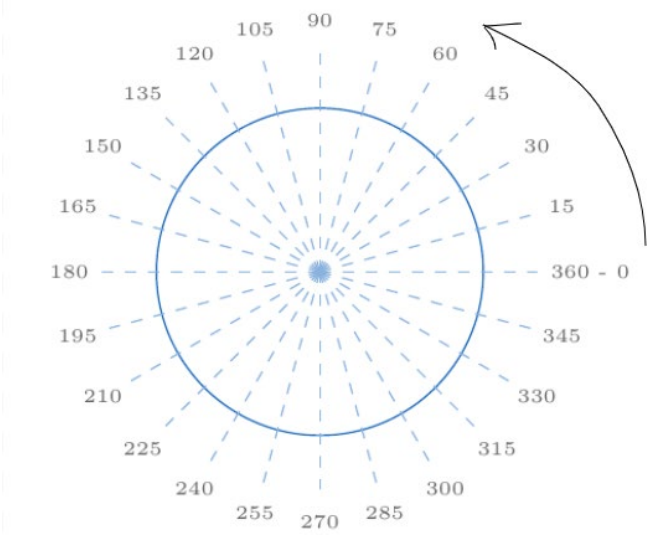


Imagen 40. Sistema de referencia

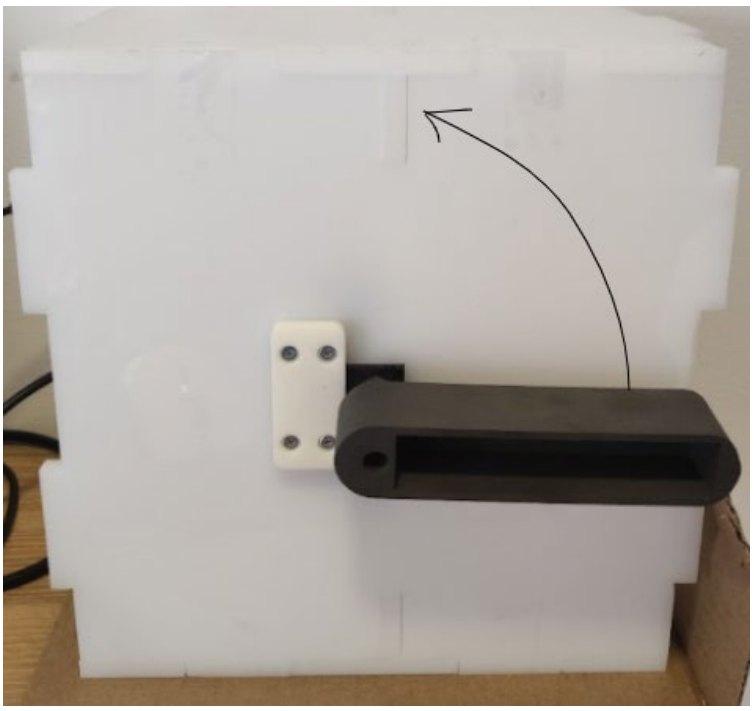


Imagen 41. Sistema de referencia aplicado al prototipo

Antes de todo, aunque no sea un modo por definición, se explicará cómo se realiza la conexión entre el servomotor y el ordenador.

5.4.1 Conectar/Desconectar el servomotor al ordenador

Un apartado clave antes de proceder a la explicación de los modos son los pasos a seguir para realizar una satisfactoria conexión entre servomotor y ordenador. La primera vez que se ejecute la aplicación aparecerá en pantalla una ventana como la siguiente imagen:

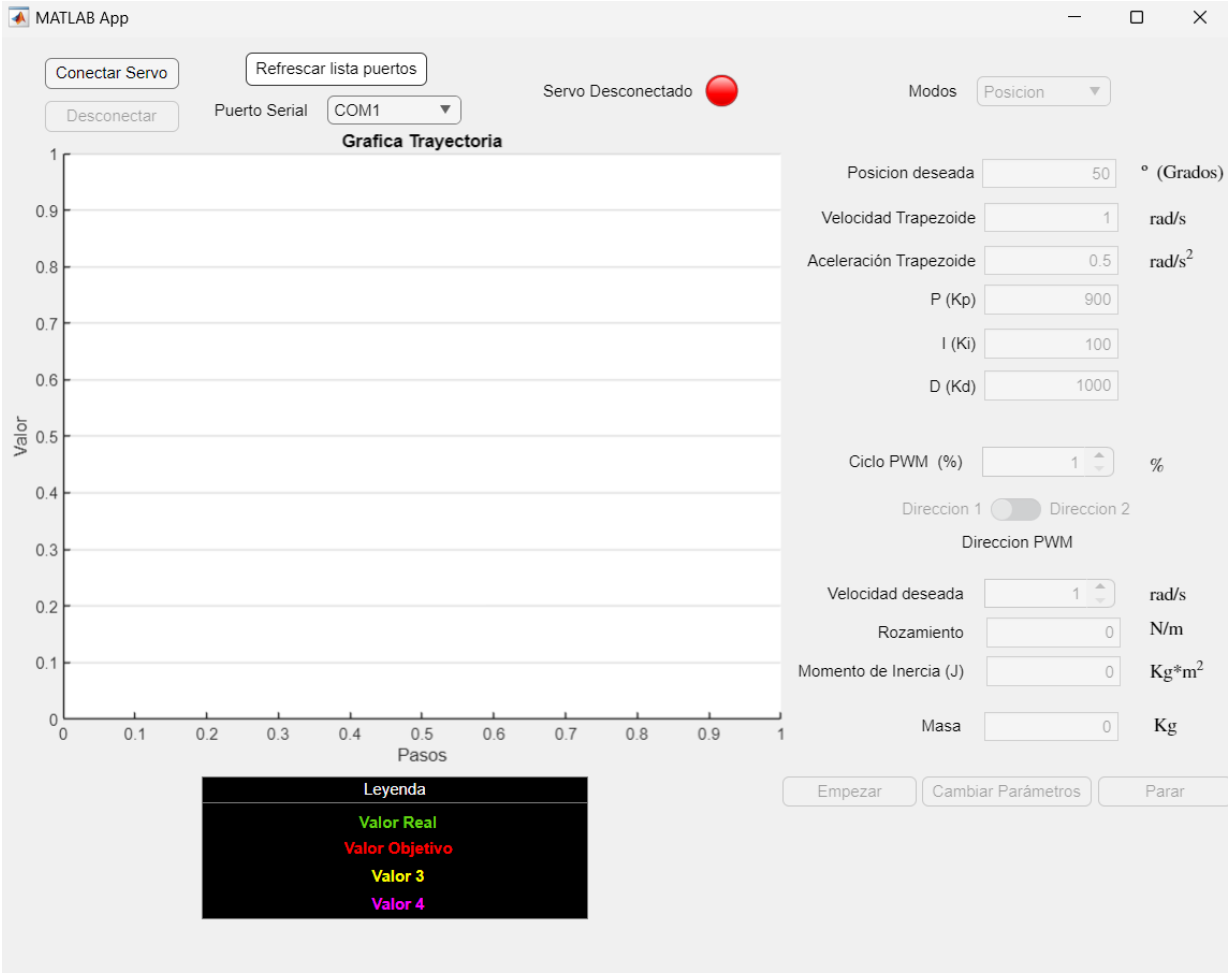


Imagen 42. Aplicación recién abierta

Como se puede observar, solo se pueden interactuar con tres elementos: “Conectar Servo”. “Refrescar lista puertos” y el desplegable de “Puerto Serial”, además, como indica el led rojo en la imagen, el servomotor se encuentra desconectado. Para realizar la conexión, primero hay que seleccionar de la lista de “Puerto Serial” el puerto COM al que está conectado la placa OPENRB-150, como se muestra en la figura:

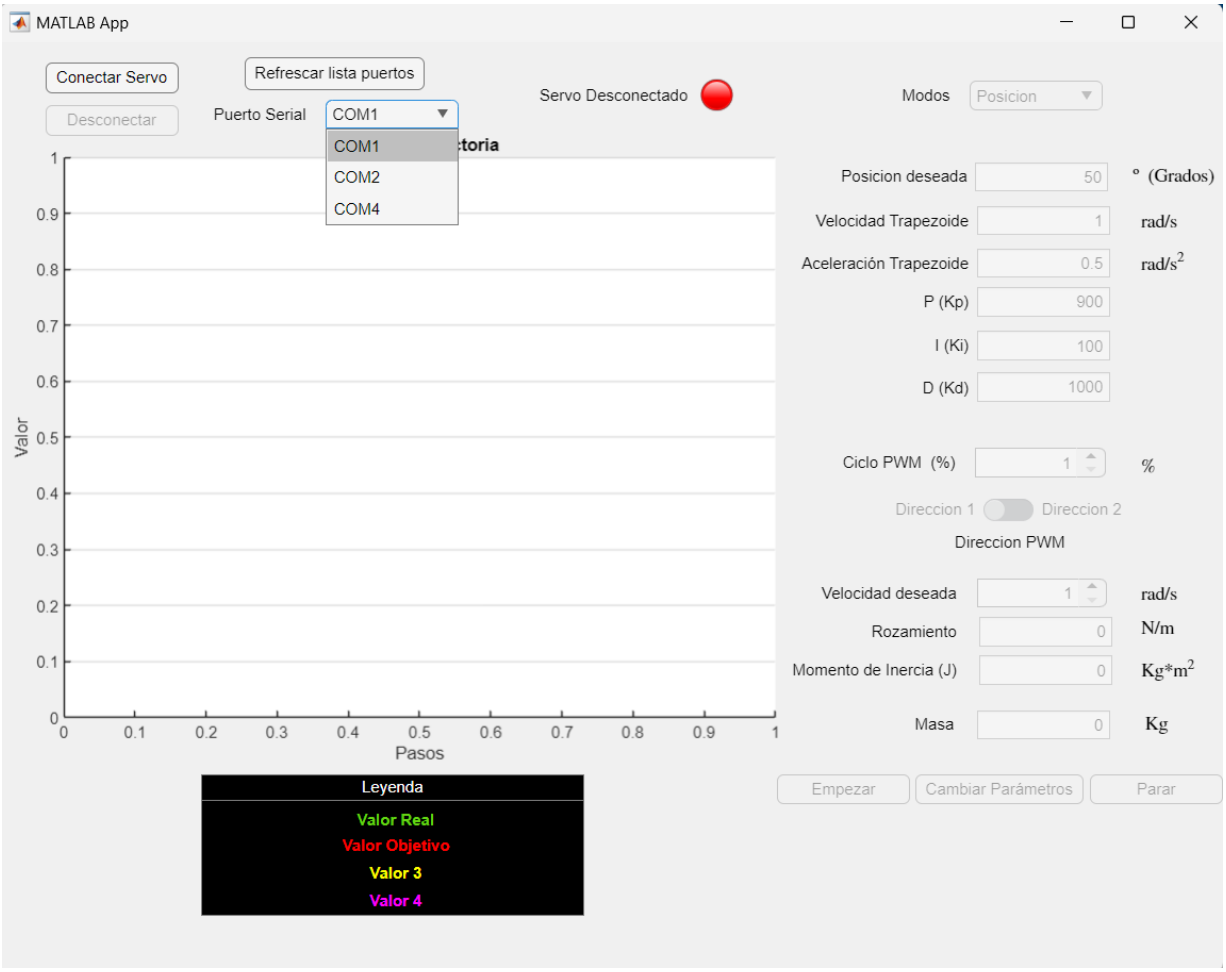


Imagen 43. Deplegable Serial

En el caso anterior, debido a pruebas anteriores, se pudo determinar que el puerto COM de la placa empleada es el COM6, pero al haberla conectado a través del cable USB al ordenador después de haber iniciado la aplicación no aparece en la lista. En el caso que la placa esté conectada al ordenador pero no aparezca su puerto se requiere pulsar el botón de “Refrescar lista puertos”. Después de haberlo pulsado, aparecen todos los puertos actualmente disponibles en el ordenador.

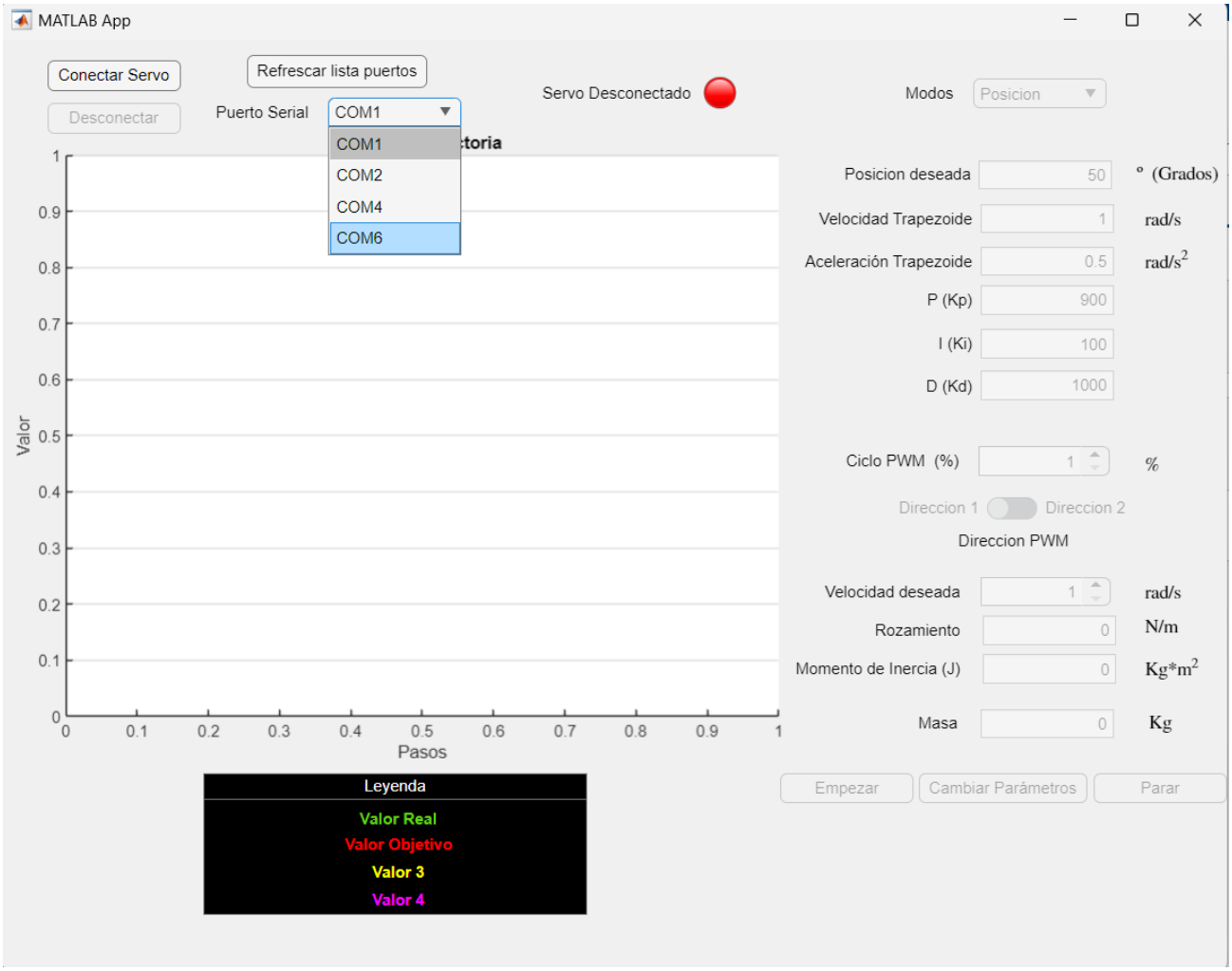


Imagen 44. Lista de puertos actualizada

Si se da la situación que no se tiene la información de que puerto es el correcto, se puede descubrir de varias maneras:

- Accediendo al “Administrador de dispositivos” a través de la barra de búsqueda del ordenador.
- Abriendo la aplicación de Arduino y viendo que puerto marca la opción de OPENRB-150, aunque si no se dispone de los drivers es posible que está opción no funcione.
- A través de la propia aplicación creada siguiendo los pasos:
  1) Desconectar la placa del ordenador.
  2) Pulsar el botón de “Refrescar lista puertos” para actualizar la lista de puertos disponibles y fijarse que puertos tiene.
  3) Volver a conectar la placa y volver a refrescar la lista.
  4) Si ha aparecido un puerto que no se encontraba en el paso anterior, ese es al que está conectado la placa.

En todo caso, en la siguiente imagen se demuestra que ocurre si se pulsa el botón de “Conectar Servo” con el puerto equivocado:

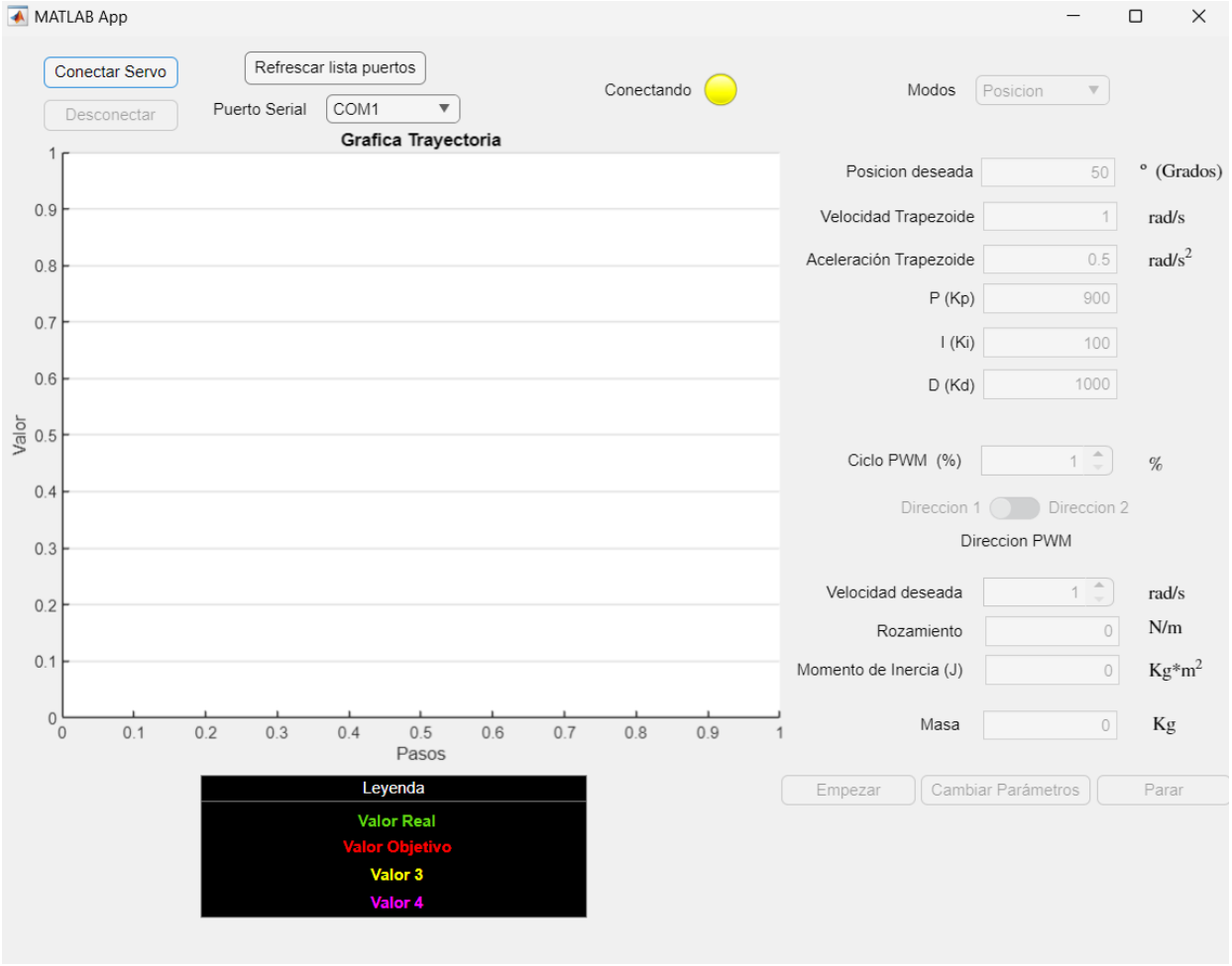


Imagen 45. Aplicación intentado conectar al puerto COM1

Como se observa en la figura, el led cambia a amarillo y aparece el mensaje de “Conectando”, pero al ser el puerto equivocado aparece el siguiente mensaje:

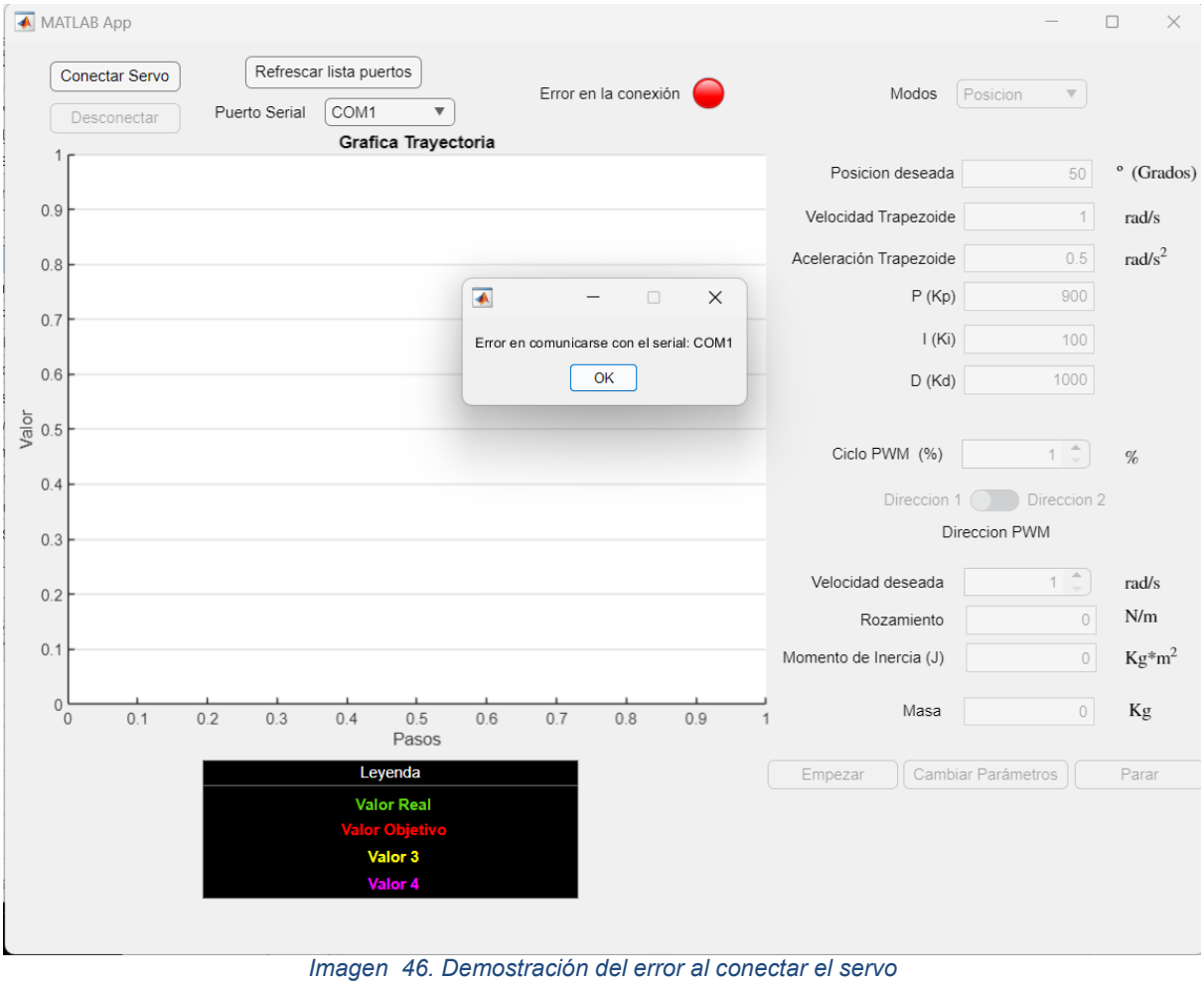


Imagen 46. Demostración del error al conectar el servo

Una vez haya terminado de intentar la conexión con ese puerto aparecerá en pantalla el mensaje de error “Error en comunicarse con el serial : COMX” (donde X es el puerto probado). Con ese mensaje en pantalla, ya se puede intentar la conexión con otro puerto, en el caso de la siguiente imagen se ha cambiado al COM6:

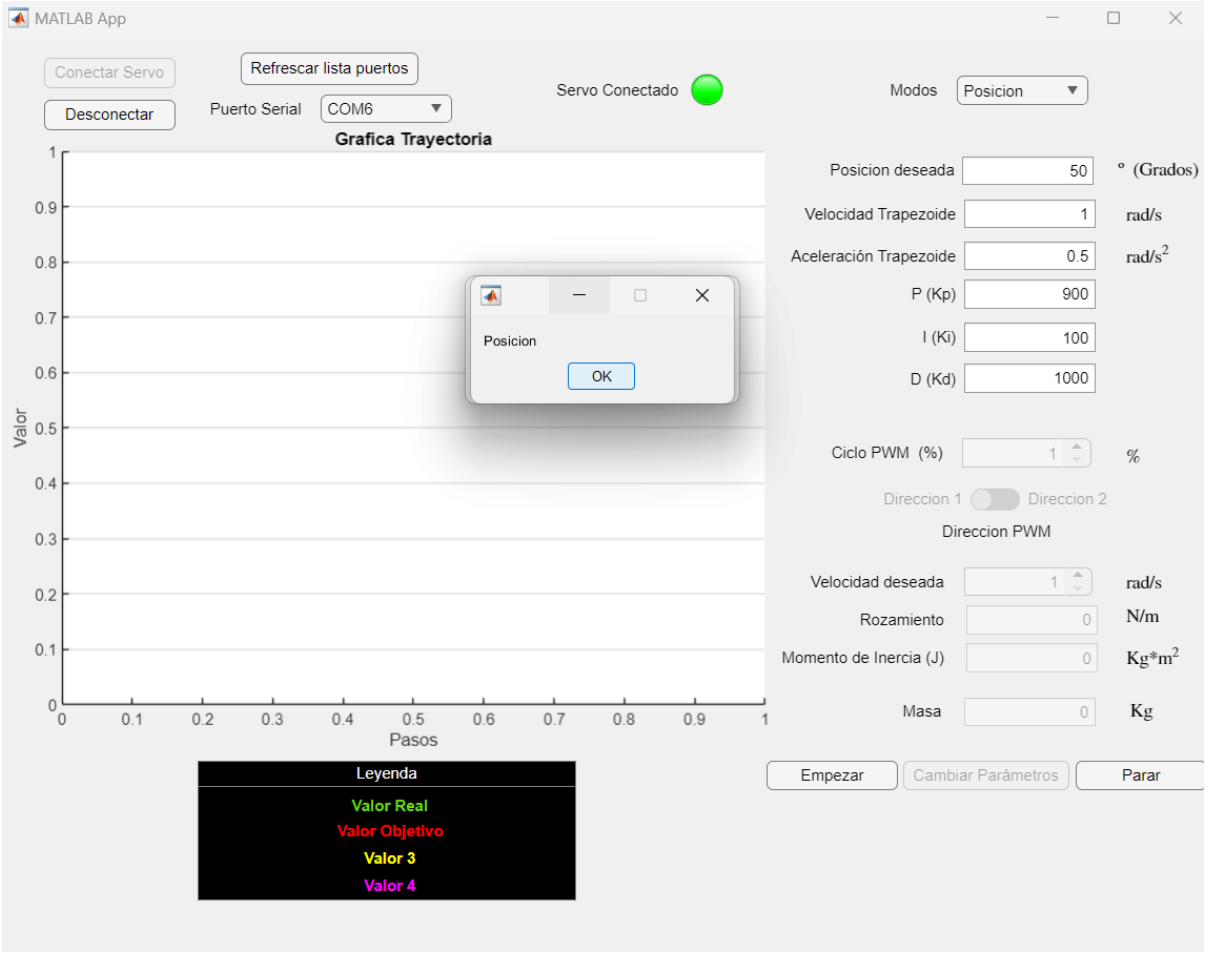


Imagen 47. Conexión satisfactoria con la placa

Con una conexión satisfactoria, la luz led cambiará a verde, aparecerá el mensaje de “Servo Conectado” y además una ventana indicará el modo actual del servomotor. Todos los modos cuentan con un campo para poner el valor objetivo que se tiene que alcanzar y alguna forma de especificación o control para llegar hasta él. Una vez ajustado todos los valores como se desean, el usuario puede pulsar el botón “Empezar” y el servomotor comenzará la trayectoria. Durante la trayectoria la gráfica se irá actualizando (con cierto retraso) según los valores que reciba del actuador, esta gráfica contará con 4 líneas y se puede comprobar en la “Leyenda” qué atributo es cada una de ellas. En el caso de apretar el botón de “Parar” la trayectoria finalizará, el servo volverá a la posición inicial y se mostrarán varias gráficas con los diferentes valores que han tenido los atributos durante todo el recorrido. Durante el movimiento también es posible apretar el botón de “Cambiar Parámetros” para que en el mismo recorrido alcance varios valores objetivos y que queden reflejados en la representación final de las gráficas.

Por último, para desconectar correctamente la aplicación de la placa es recomendable primero pulsar el botón de “Desconectar”, entonces cuando se pueda visualizar el letrero de “Servo Desconectado” y la luz led sea roja es indicación de que el canal de comunicación está cortado y se puede cerrar la aplicación de forma segura.

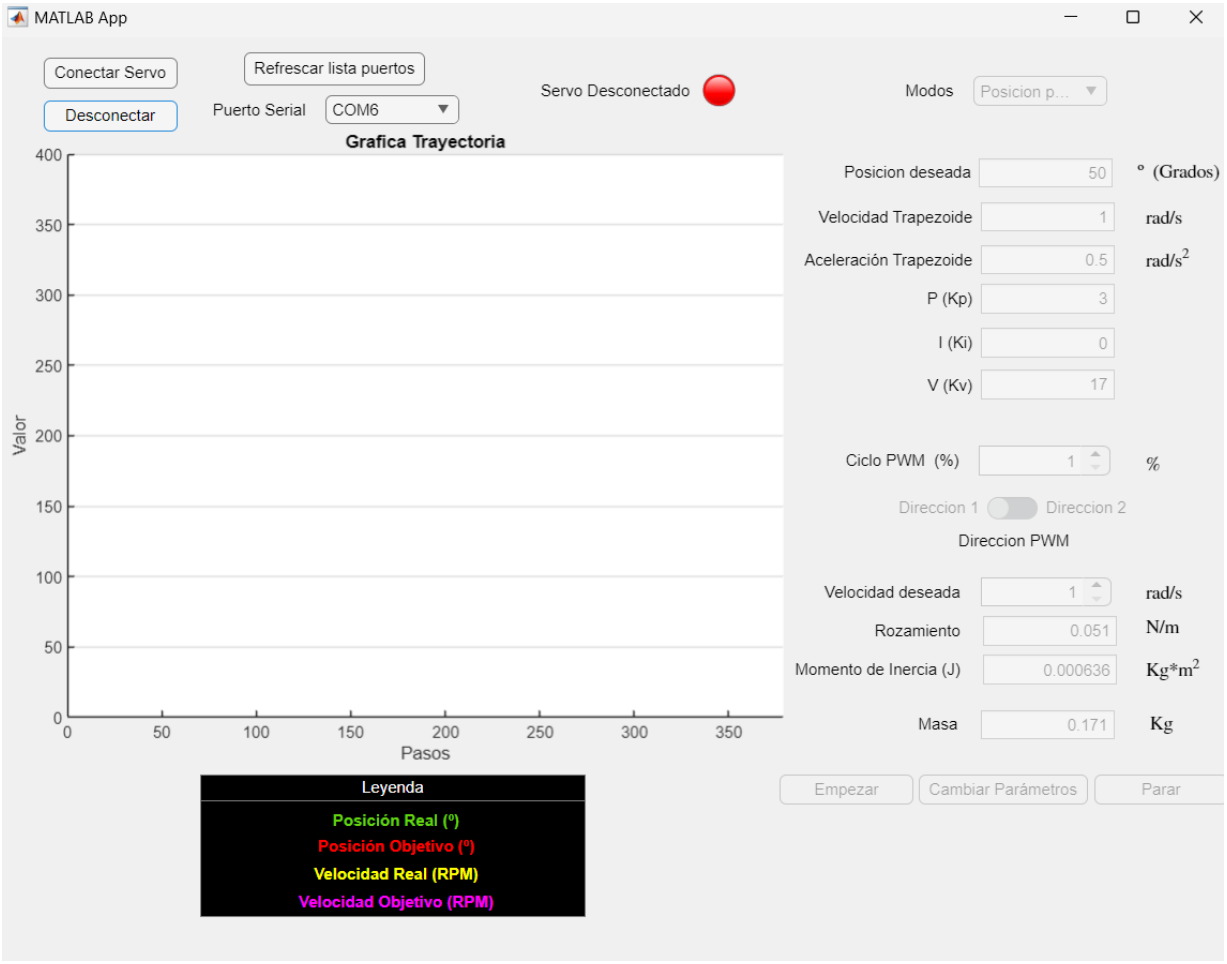


Imagen 48. Servo desconectado correctamente

5.4.2 Modo posición

Este modo usa el modo que viene preconfigurado del servomotor y tiene el objetivo de enseñar al alumnado los efectos de un control PID en una trayectoria trapezoidal sin ningún tipo de perturbación. El alumnado es capaz de modificar los siguientes parámetros:

- Posición objetivo.
- Velocidad objetivo máxima del trapezoide
- Aceleración objetiva máxima del trapezoide
- Ganancia Proporcional (Kp)
- Ganancia Integral (Ki)
- Ganancia Derivativa (Kd)

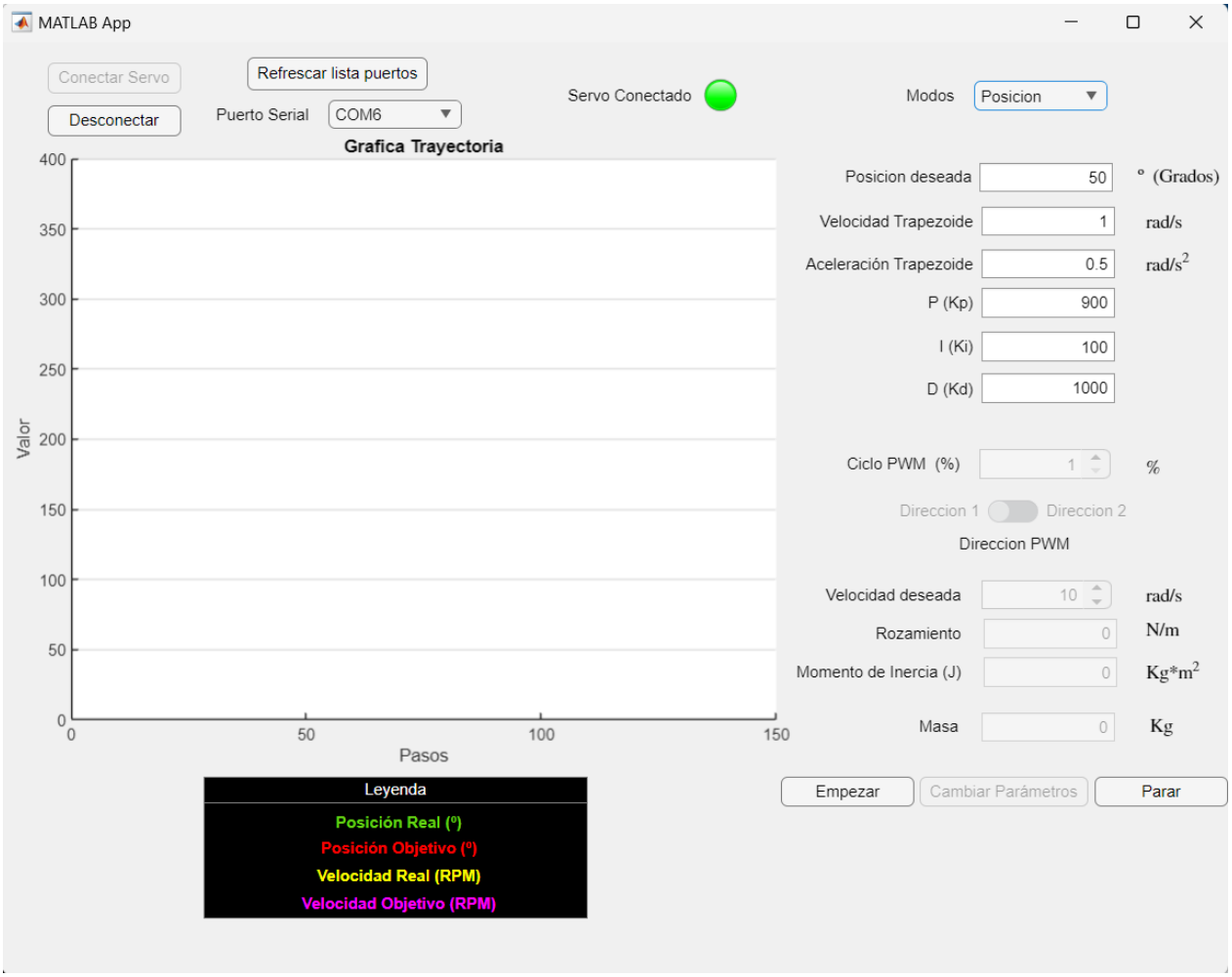


Imagen 49. Aplicación en modo posición a la espera de iniciar un recorrido

Antes de todo, una trayectoria trapezoidal es un movimiento de la posición del servo con forma de trapezoide cuando se observa la velocidad del recorrido. Para generar este tipo de trayectoria primero es necesario obtener la posición objetivo y la posición actual del servomotor, luego se calcula el tiempo de la trayectoria según la velocidad máxima y aceleración deseada. Este cálculo tiene en cuenta una aceleración inicial hasta llegar a la velocidad máxima, una etapa de velocidad constante y una desaceleración final al acercarse a la posición objetivo. Este tipo de recorridos consigue un movimiento suave que evita grandes aceleraciones que pueden ocasionar daño, además de conseguir acercarse a los objetivos con velocidades pequeñas y controladas.

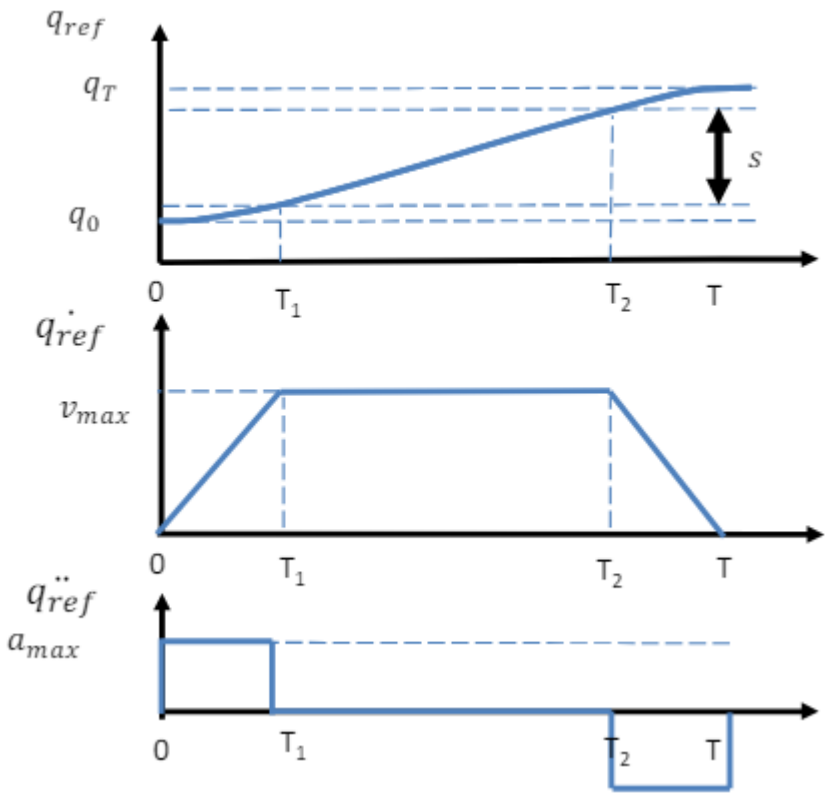


Imagen 50. Esquema de la posición, velocidad y aceleración de un recorrido con trayectoria trapezoidal

Al mismo tiempo, este modo permite cambiar los valores PID del servomotor con los efectos siguientes:

- El control proporcional (Kp) afecta al error instantáneo de la posición objetivo y aumentarlo consigue que la posición final sea más parecida a la de referencia, además que provoca que la respuesta sea más rápida. Al mismo tiempo, un valor elevado puede provocar oscilaciones e inestabilidad durante el recorrido.
- El control integral (Ki) afecta al error acumulado del recorrido y tiene también como objetivo reducir el error final de la trayectoria, pero en este caso suele afectar más a perturbaciones o errores iniciales. Un valor alto de este parámetro provoca una respuesta más lenta además de causar inestabilidad.
- El control derivativo (Kd) afecta a la velocidad de cambio del error y tiene como objetivo suavizar la respuesta del sistema y reducir oscilaciones, ya que intenta anticipar el cambio del sistema y actuar acorde a ellos. Una constante derivativa grande puede provocar ruido en el sistema y causar inestabilidad.

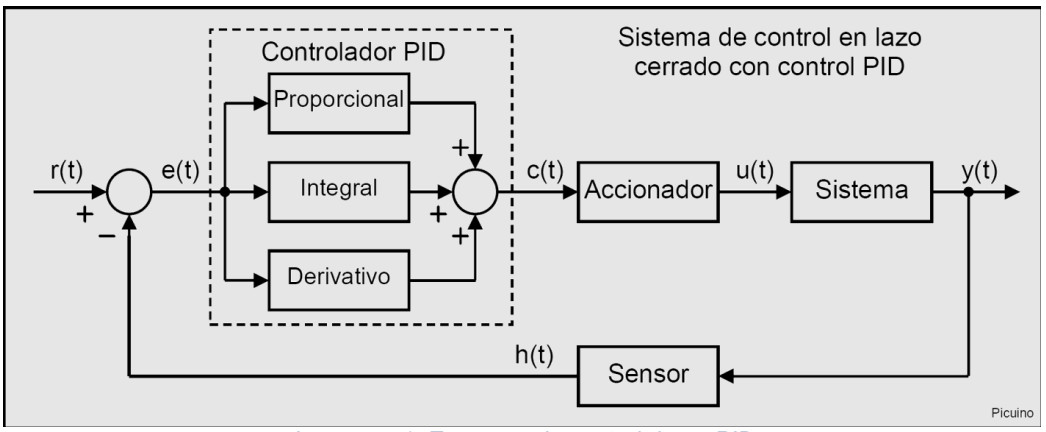


Imagen 51. Esquema de control de un PID

Explicados los conceptos, se inicia el recorrido con una posición objetivo de 50 º y solo una Kp de 20 (los valores de PID no tienen unidades). Al no tener una prealimentación, el sistema no produce respuesta a menos que en las constantes de PID haya valores diferentes de 0. Como se puede observar en las siguientes imágenes, el sistema no está correctamente ajustado ya que tiene una trayectoria lenta y un error considerable en la posición final:

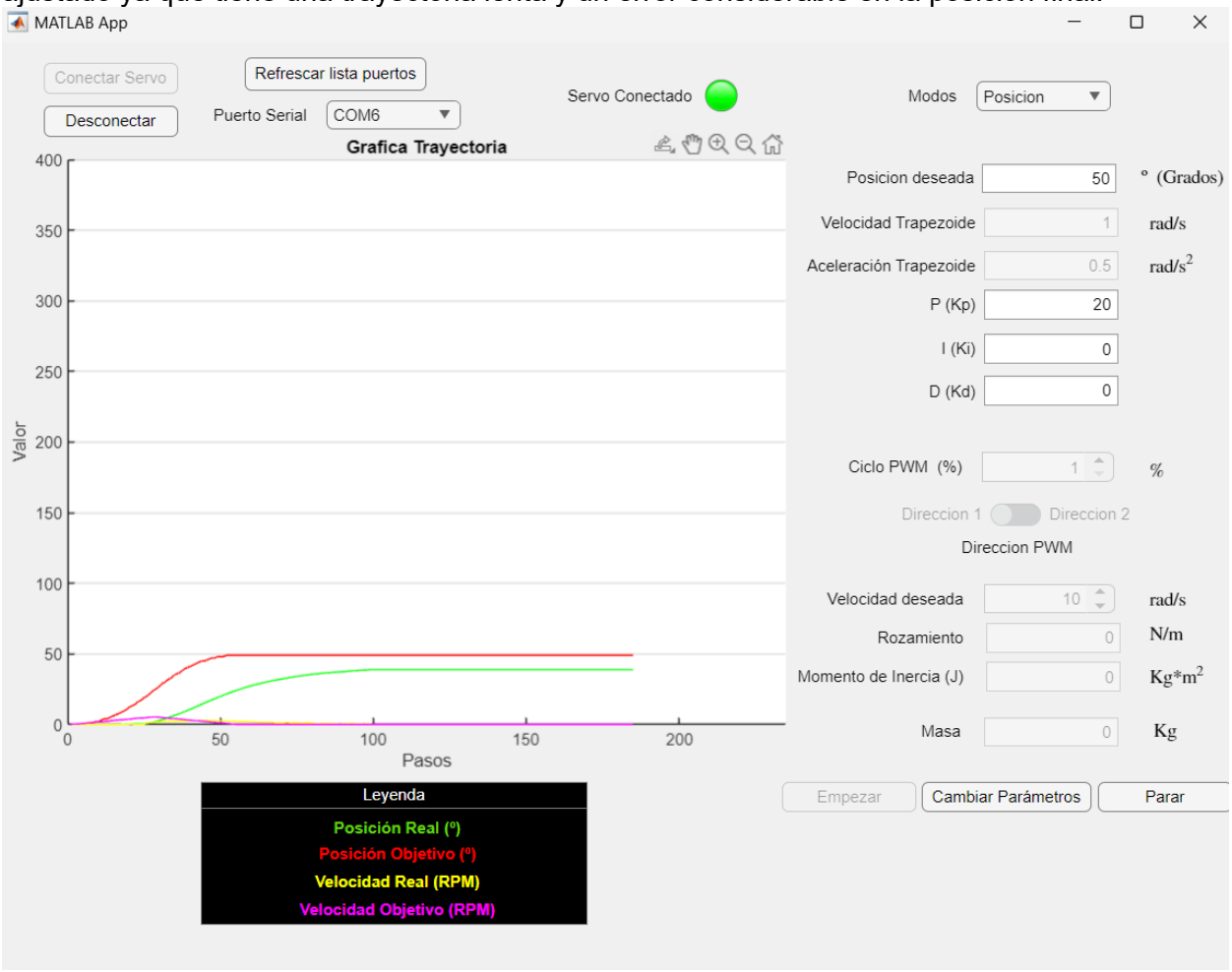


Imagen 52. Trayectoria de 50 º con Kp 20

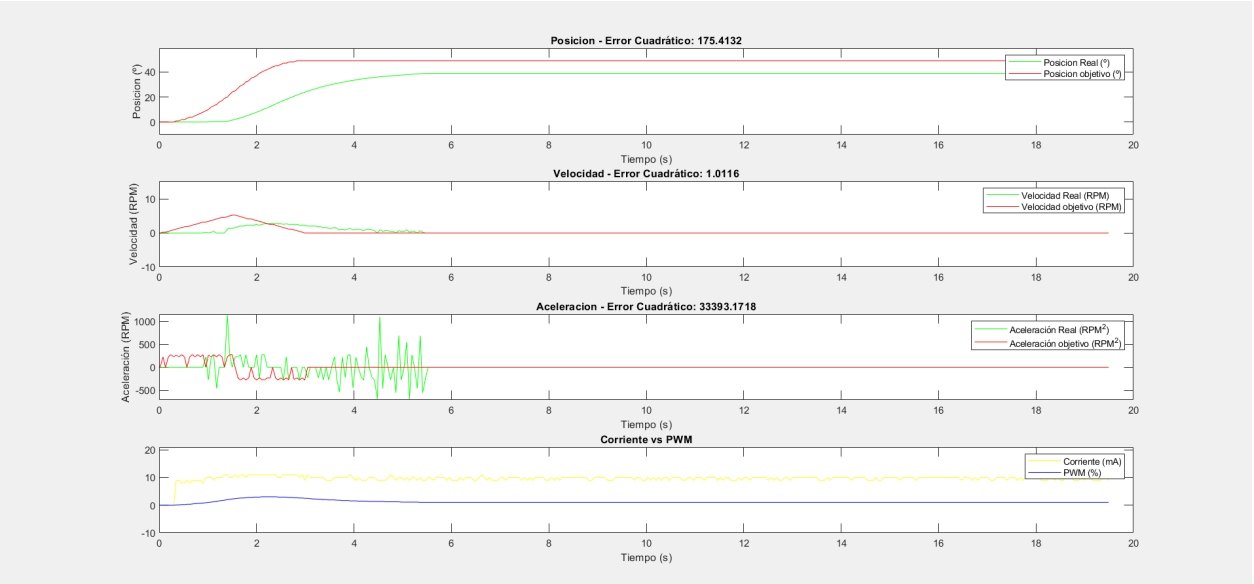


Imagen 53. Gráficasl recorrido de la Imagen 52

En el siguiente ejemplo, con los mismos valores objetivos de posición, velocidad y aceleración se consigue una respuesta mucho más cercana a la deseada con ajuste correcto de los parámetros PID del controlador:

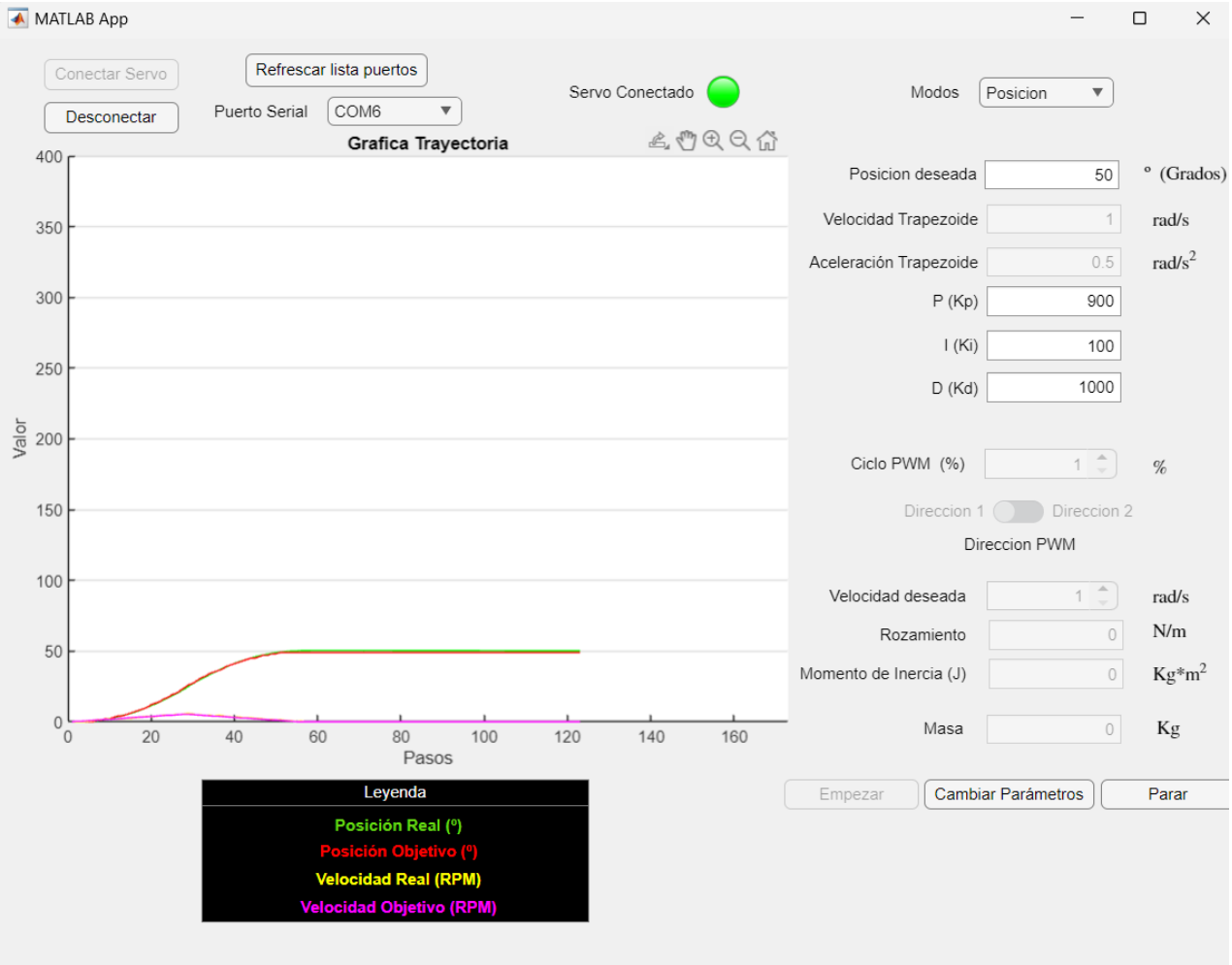


Imagen 54. Trayectoria de 50 ° con PID adecuado

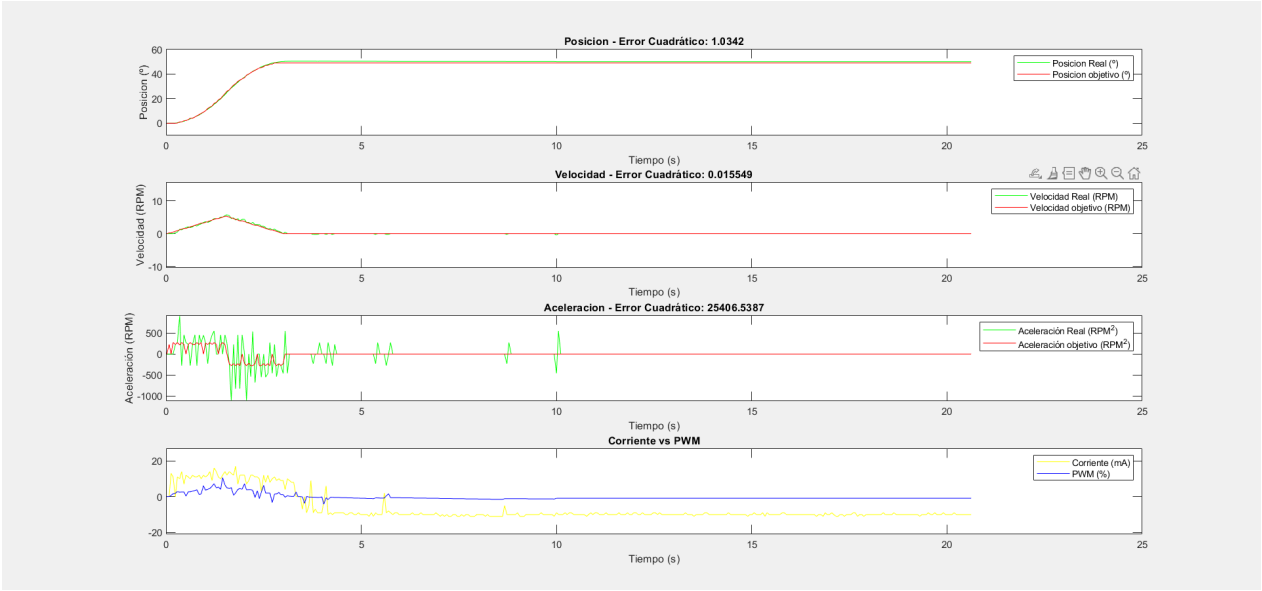


Imagen 55. Gráficas de la Imagen 54

Se puede observar el movimiento real a través de las siguientes imágenes, que ha rotado los 50 º desde su posición 0 º en sentido contrario a las agujas del reloj:

Imagen 56. Articulación a 0 º

Imagen 57. Articulación a 50 º

5.4.3 Modo tensión/PWM

El siguiente modo, también configurado de fábrica por los fabricantes, es el modo de control por tensión/PWM. En este modo el usuario es capaz de modificar los siguientes parámetros:

- El porcentaje objetivo del PWM
- La dirección en la que gira el servomotor

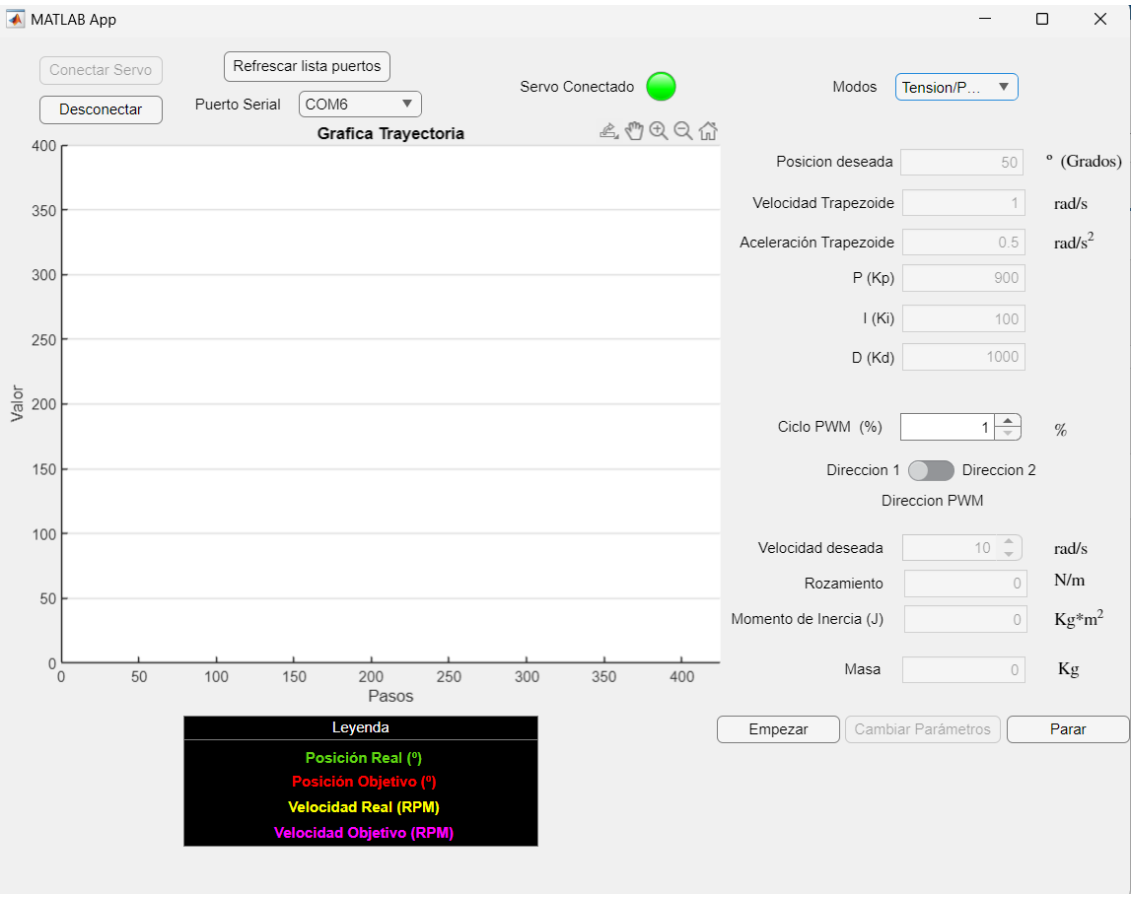


Imagen 58. Parámetros configurables en el Modo PWM

El control por tensión a través de PWM, modulación por ancho de pulso en inglés, es una técnica en la que se alterna una señal, en este caso la tensión, entre dos estados (encendido/apagado, alto/bajo) lo que modifica el valor medio que se recibe. El ciclo de trabajo de un PWM indica el tiempo o período en el que la seña se encuentra en su estado de alto, por ejemplo, un período del 25 % significa que la señal está en su período de alto un 25 % de su tiempo y la señal que se recibe es un 25% del valor de tensión máxima.

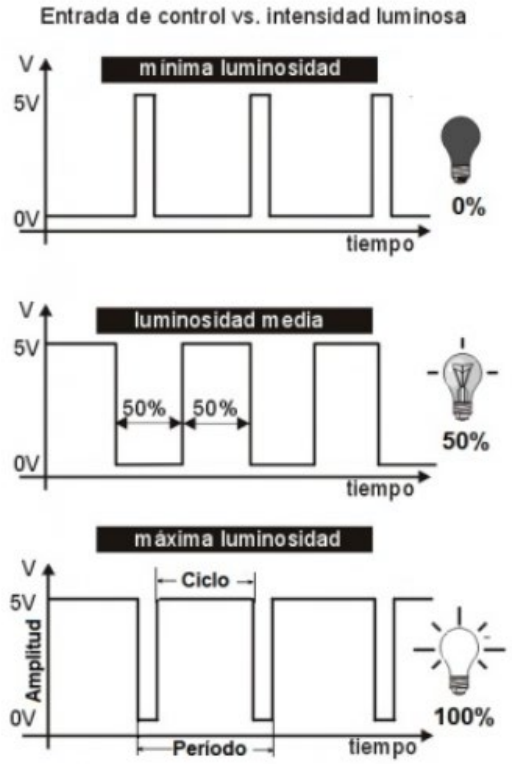


Imagen 59. Demostración de un control por PWM

Explicado lo necesario, se procede a hacer un ejemplo con un porcentaje del 10% de PWM. En este ejemplo, se interrumpe manualmente el giro del servomotor para observar su respuesta:

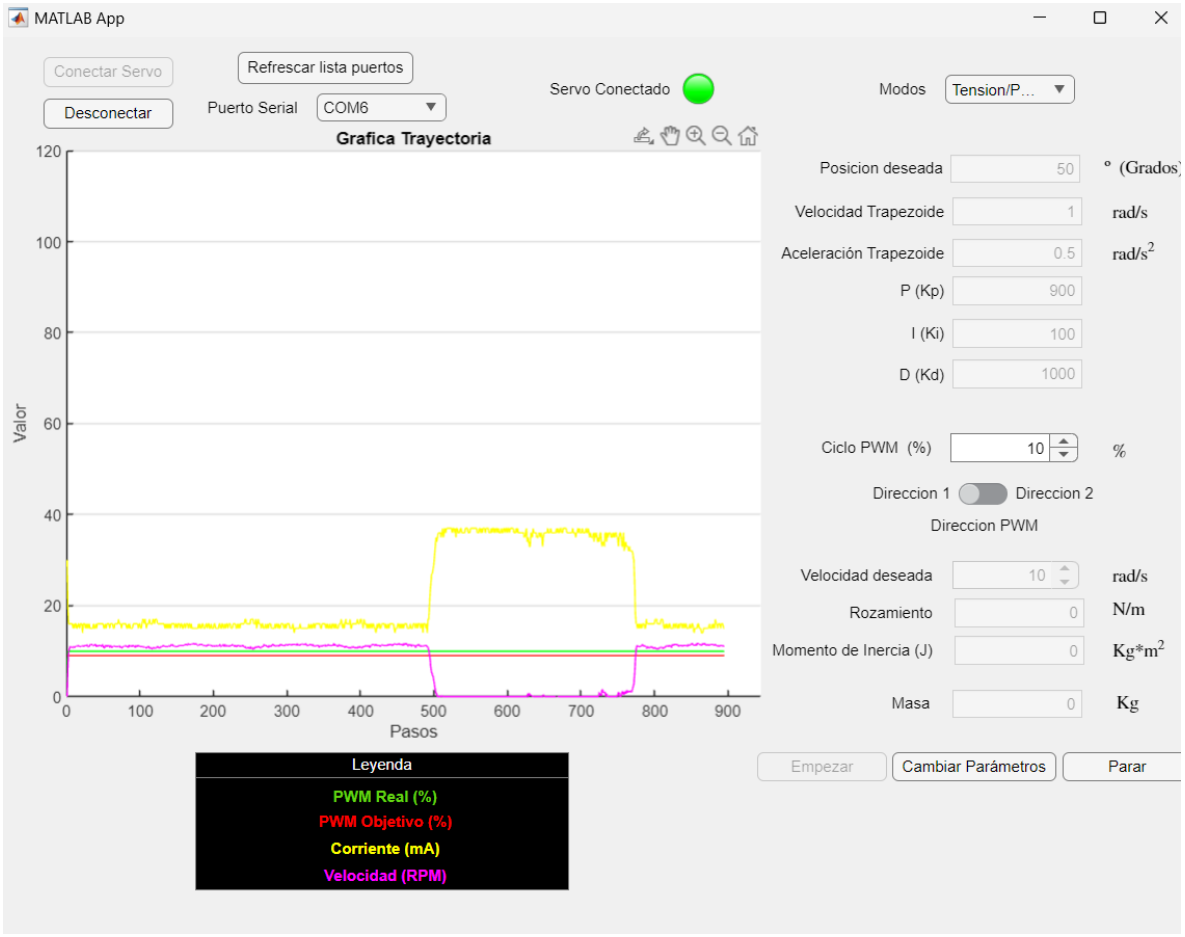


Imagen 60. Recorrido del modo PWM

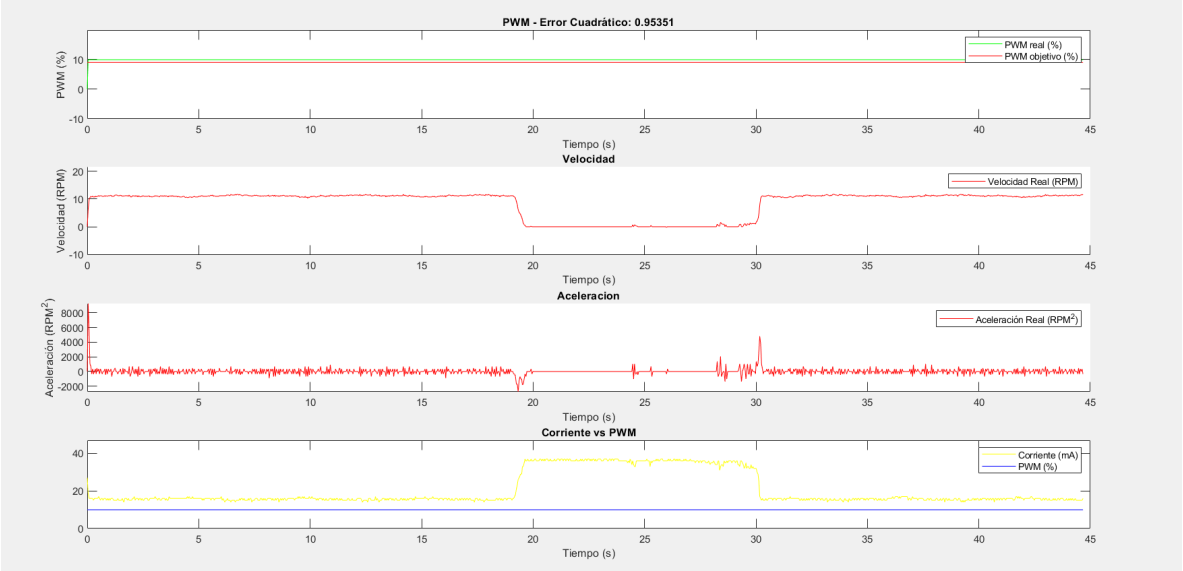


Imagen 61. Gráficas de la Imagen 60

Como se puede observar en las imágenes 60 y 61, si se manipula/perturba el movimiento, el servomotor intenta mantener su señal PWM aumentando la corriente. Esto es observable en el momento que la velocidad baja a cero al ser interrumpido manualmente al mismo tiempo que aumenta considerablemente la corriente demandada, que puede ser un peligro en cualquier tipo de aplicación.

5.4.4 Modo velocidad

Este es el último ejemplo de control proporcionado de fábrica utilizado en la herramienta. En este modo, se tiene como objetivo marcar una velocidad para que el servomotor la alcance y se mantenga. Los valores que se pueden modificar son:

- Velocidad objetivo.
- Aceleración para alcanzar la velocidad objetivo.
- Constante Proporcional (P).
- Constante Integral (I).

Entonces, con una aceleración de 1 rad/s^2 , una velocidad deseada de 3 rad/s y un PI de 10 y 10, se observa lo siguiente:

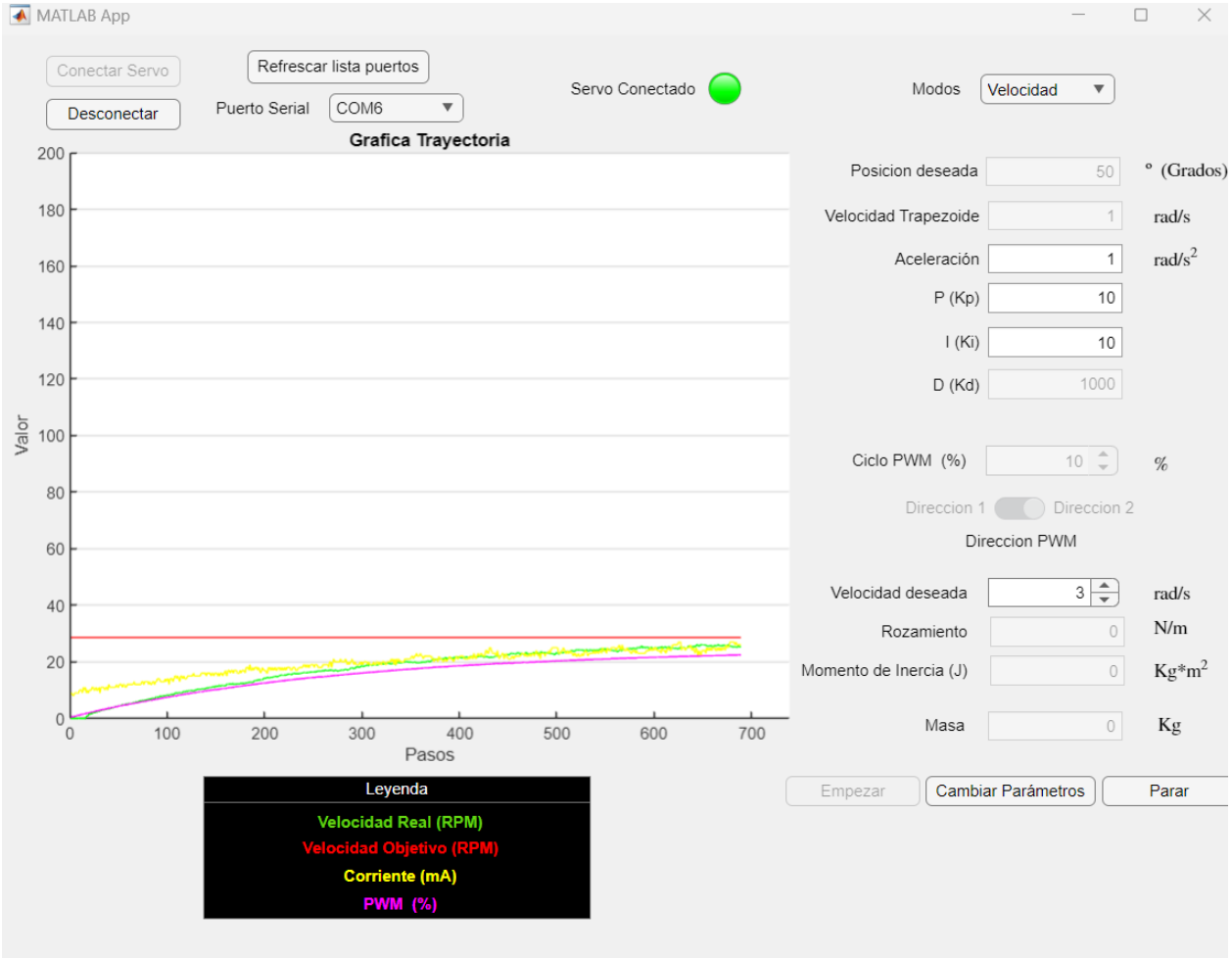


Imagen 62. Trayectoria de velocidad de P=10 e I=10

En la trayectoria de la imagen se observa una respuesta lenta del servomotor con unos valores bajos de PI. Sin embargo, aumentar la constante proporcional, creando una repuesta más rápida, también crea un pico de corriente al inicio de la trayectoria:

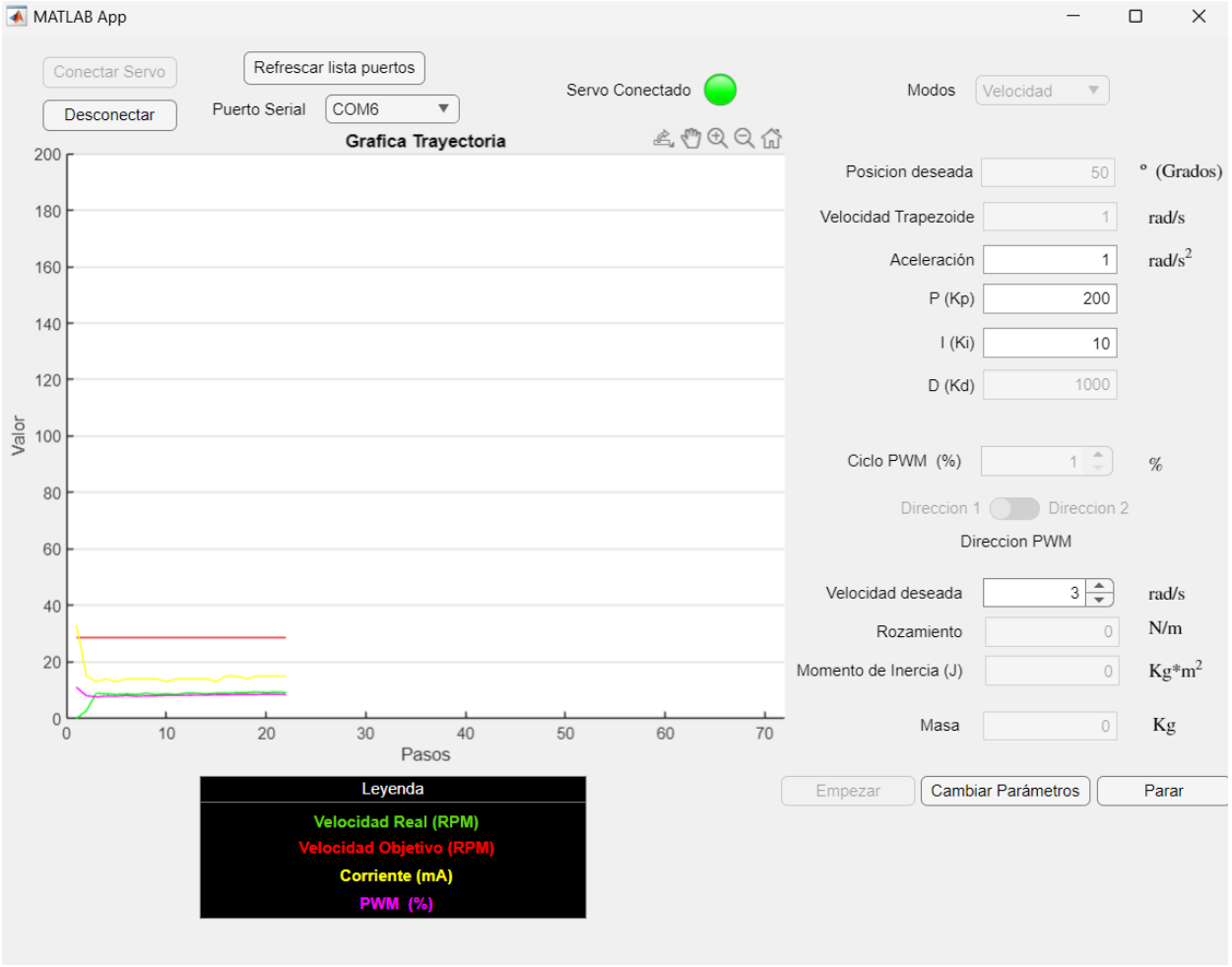


Imagen 63. Trayectoria de velocidad con P=200 e I=10

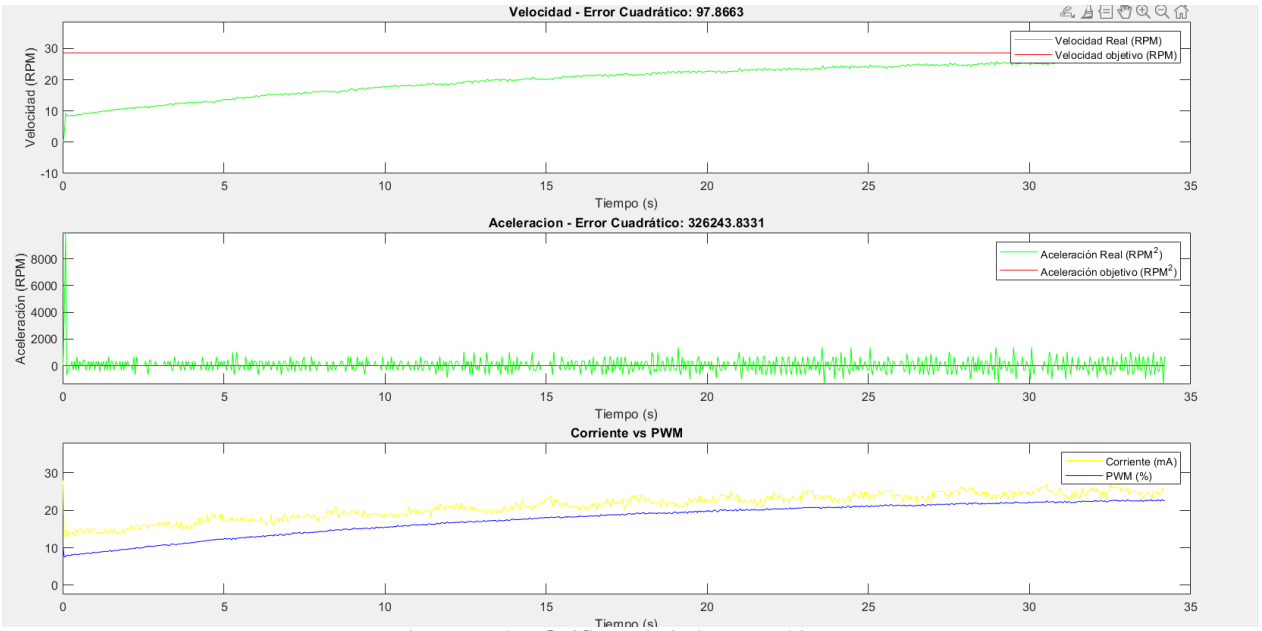


Imagen 64. Gráficas de la Imagen 63

Por este motivo, aunque se reduzca el tiempo de trayectoria al empezar desde un valor de velocidad más alto, ese pico de corriente puede ser peligroso.

5.4.5 Modo velocidad por tensión

El modo velocidad por tensión ha sido creado para obtener una respuesta mejor ante una velocidad de referencia que el modo programado para el servomotor. Este recibe una velocidad objetivo, que es tratada por el algoritmo de control y produce una velocidad instruyendo al servomotor que ciclo de PWM se necesita. Los valores que se pueden configurar son:

- Velocidad deseada
- Aceleración
- Constante proporcional (Kp)
- Constante integral (Ki)
- Constante derivativa (Kd)

Para este modo, primero se calcula la trayectoria que la velocidad ha de seguir con la función de “calculateTrajectoryVelocity”:

```
TrajectoryParamsV trajV = calculateTrajectoryVelocity( reference_velocity, initial_velocity, acceleration);
// PID parámetros
float Kp = receivedArray[4];
float Ki = receivedArray[5];
float Kd = receivedArray[6];
float v =0;
max_rate_of_change_ = 1;
while (Serial.available() <= 0) {
  // Example usage of the PID functions
  v = customPIDVelocity(trajV,  Kp, Ki, Kd );

  dxl.setGoalPWM(DXL_ID, v * K_PWM, UNIT_PERCENT);


  // Envío de datos al ordenador
```

Imagen 65. Código del bucle de control

Esta función crea una rampa sencilla en base a la aceleración deseada:

```
//Cálculo de la trayectoria para los modos de velocidad
TrajectoryParamsV calculateTrajectoryVelocity( float vmax_rad, float vini_rad, float amax_rad) {
  TrajectoryParamsV traj;
  previous_Time = millis();

  previous_error = 0;
  integral_error = 0;

  traj.vmax_rpm = vmax_rad * (60 / (2 * M_PI)); // Velocidad en RPM
  traj.amax_rpm = amax_rad * (60 / (2 * M_PI));
  traj.vini_rpm = vini_rad * (60 / (2 * M_PI));
  previous_output = traj.vini_rpm;

  float v_diff = traj.vmax_rpm - traj.vini_rpm;
  float dir = (v_diff > 0) ? 1 : (v_diff < 0 ? -1 : 0);

  v_diff = fabs(v_diff);

  traj.T = v_diff / traj.amax_rpm;
  traj.amax_rpm =  traj.amax_rpm*dir;
  float t0 = millis() / 1000.0f;
  traj.t0 = t0;
  return traj;
}
```

Imagen 66. Función que crea la trayectoria de velocidad

Con la trayectoria calculada, se entra en un bucle donde primero se calcula que valor se ha de enviar al servomotor según los valores PID recibidos. Esto se consigue con la función de “customPIDVelocity”:

```
// Función de control del modo velocidad
float customPIDVelocity(const TrajectoryParamsV &traj,  float kp, float ki, float kd) {

  float t = (millis() / 1000.0f) - traj.t0;

  float velocity = calculateVelocity(traj, t);

  float current_velocity = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);

  // Cáclulo de la diferencia de tiempo
  float current_Time = millis();
  float deltaT = (current_Time - previous_Time) / 1000.0f;  // Diferencia en segundos

  // Calcular errores y salida del PID
  float error = velocity - current_velocity;
  integral_error = integral_error + error * deltaT;
  float derivative_error = (error - previous_error) / deltaT;
  float control_output = kp * error + ki * integral_error + kd * derivative_error;

  // Limita el valor de salida de control para que sea cercano al anterior
  float delta_output = control_output - previous_output;
  if (fabs(delta_output) > max_rate_of_change_) {
    delta_output = max_rate_of_change_ * (delta_output / fabs(delta_output));
  }
  control_output = previous_output + delta_output;

  // Se actualizan los valores anteriores
  previous_output = control_output;
  previous_error = error;
  previous_Time = current_Time;

  ArraytoSend[1]= dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
  ArraytoSend[2]= velocity;

  // Se devuelve la acción de control calculada
  return control_output;

}
```

Imagen 67. Función del cálculo de la acción de control

En esta función se hacen unos cálculos sencillos de los diferentes errores teniendo en cuenta el valor que debería estar en este mismo momento (función de “calculateVelocity”, la cual almacena la velocidad de referencia en “velocity”) y el valor actual que detecta el servo que tiene (“current_velocity” en revoluciones por minuto).

El valor obtenido de este bucle de control se multiplica por la constante “K_PWM” que es una constante calculada que relaciona la velocidad con el valor de PWM del servomotor. Explicada la programación, se procede a hacer una prueba del modo buscando una velocidad de 2 rad/s (aproximadamente 19 rpm, la representación se realiza en rpm ya que sus valores son más altos que los equivalentes de rad/s). Durante la trayectoria también se procederá a bloquear el giro para observar cómo se comporta el control:

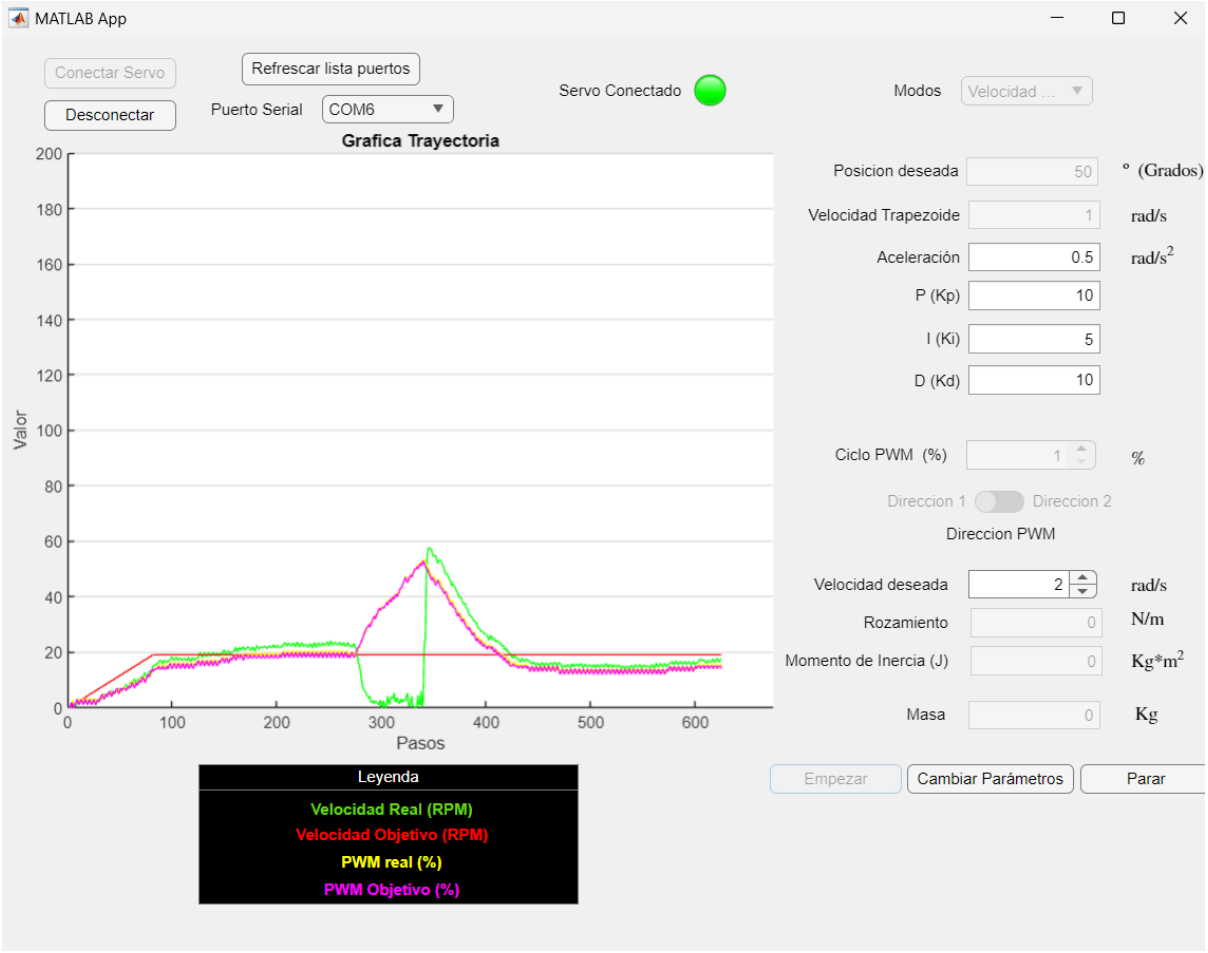


Imagen 68. Trayectoria del modo velocidad por tensión

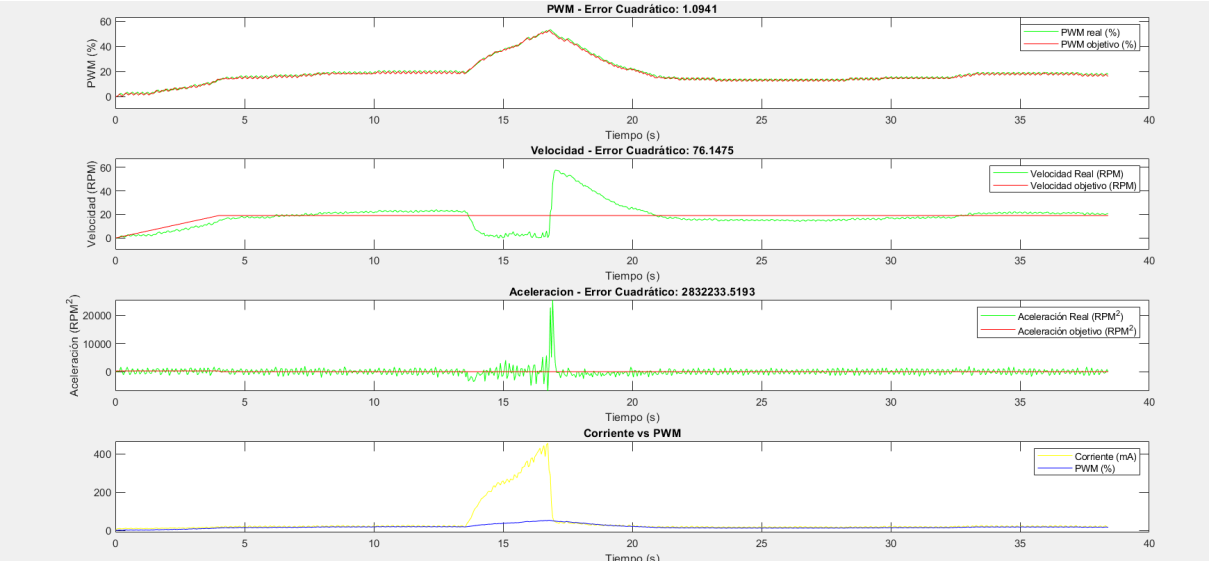


Imagen 69. Gráficas de la Imagen 68

Como se puede observar en las gráficas, este modo funciona de forma mucho más satisfactoria que el mostrado anteriormente de fábrica, ya que el control programado consigue seguir correctamente la velocidad con una respuesta mucho más rápida y cercana. Además, se comprueba una razón por la que el control por tensión puede ser peligroso, ya que, al bloquear el giro, el control aumenta la corriente para mantener el PWM y la velocidad, provocando un pico de corriente de más de 400 mA. En la realidad, esto se transmite en el

servo haciendo un esfuerzo excesivo que puede causar daños.

5.4.6 Modo velocidad por corriente

El otro modo de velocidad programado se corresponde al modo de velocidad por corriente. En este, el usuario selecciona la velocidad objetivo y el servomotor la consigue a través de un control de la corriente calculada. En este modo, se puede configurar los siguientes parámetros:

- Velocidad objetivo
- Aceleración objetivo
- Constante Proporcional (Kp)
- Constante Integral (Ki)
- Constante derivativa (Kd)

La programación es casi idéntica que el modo de velocidad por tensión ya que utiliza las mismas funciones:

```
float reference_velocity = receivedArray[2];
float acceleration = receivedArray[3];

TrajectoryParamsV trajV = calculateTrajectoryVelocity( reference_velocity, initial_velocity, acceleration);
// PID parameters
float Kp = receivedArray[4]; //1000
float Ki = receivedArray[5]; //100
float Kd = receivedArray[6]; //700
float v =0;
max_rate_of_change_ = 1;
while (Serial.available() <= 0) {
  // Example usage of the PID functions
  v = customPIDVelocity(trajV,  Kp, Ki, Kd );

  dxl.setGoalCurrent(DXL_ID, v * K_current, UNIT_MILLI_AMPERE);

  // Envío de datos al ordenador
```

Imagen 70. Bucle de control del modo velocidad por corriente

La gran diferencia es que la constante de velocidad es “K_current”, ya que esta es la que relaciona la velocidad con la corriente del servomotor. Entonces, se procede a hacer un recorrido de demostración en el que se volverá a bloquear el giro para observar la diferencia con el modo de control anterior.

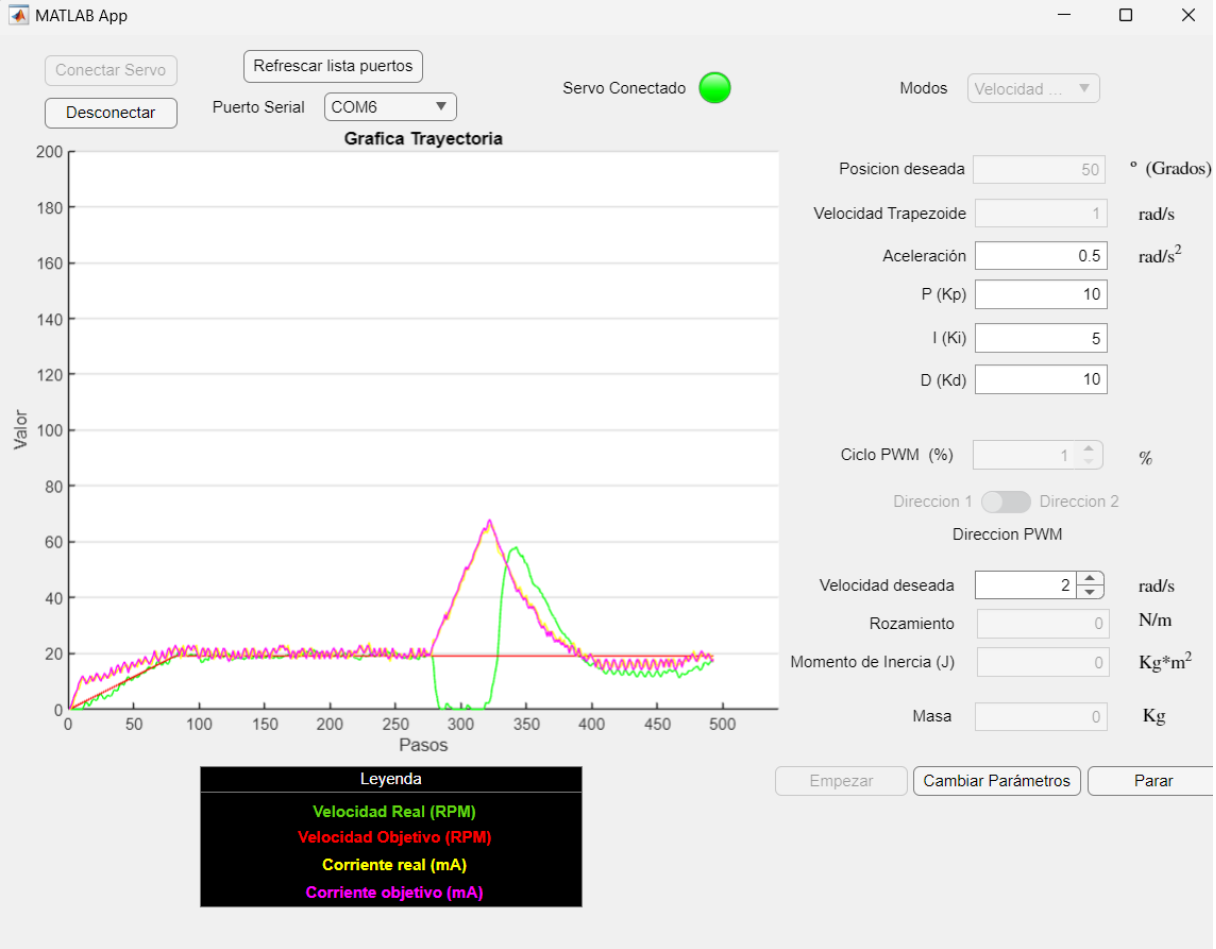


Imagen 71. Trayectoria modo velocidad por corriente

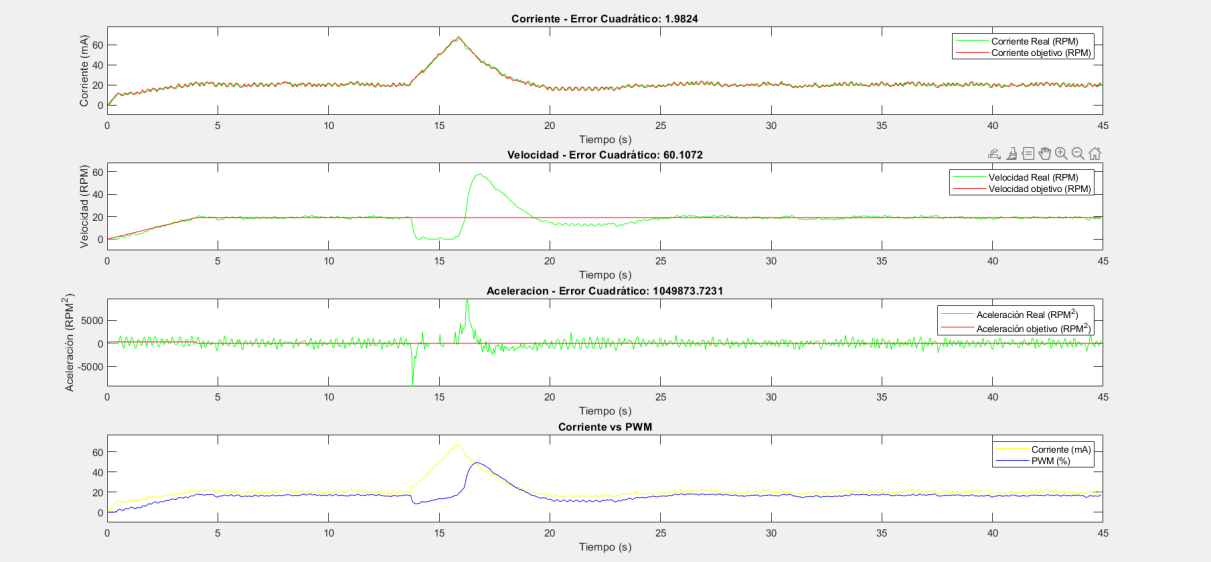


Imagen 72. Gráficas de la Imagen 71

En las gráficas 71 y 72 se puede observar como el modo por corriente consigue un valor más ajustado de la velocidad deseada al obtener un error cuadrático más pequeño que al controlar la tensión en el caso de observar el parámetro de control (gráfica superior). Además, se puede observar que ante un bloqueo el control intenta aumentar la corriente, pero el valor es mucho menor que en el caso anterior (alrededor de 60 mA comparados a los más de 400 mA del caso por tensión). Esto se transmite que al bloquear el servo no se ha sentido ni la mitad de

fuerza percibida que en el caso de tensión, ya que una corriente menor es directamente proporcional a un par de fuerza menor. Debido a casos como este, en la robótica se usa más ampliamente el control por corriente que el de tensión.

5.4.7 Modo posición por corriente

Este modo ha sido programado para crear un método en el que el servo controle la corriente proporcionada y eso ajuste la posición final. En este modo los valores configurables son:

- Posición deseada
- Velocidad del trapezoide
- Aceleración del trapezoide
- Constante Proporcional (Kp)
- Constante Integral (Ki)
- Constante Derivativa o de velocidad (Kv)
- Constante de rozamiento del servomotor
- Momento de Inercia de la pieza (J)
- Masa de la pieza

Este modo ha sido programado para poder ajustar el control de posición y que perturbaciones como la gravedad puedan ser solventadas. Primero de todo, se muestra como se llama al generador de trayectorias (función “calculateTrajectory”) en el programa:

```
// Recoge los valores mandados por el ordenador

float q0_deg = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
float qT_deg = receivedArray[1];
float vmax_rad = receivedArray[2];  // Velocidad mandada en radianes
float amax_rad = receivedArray[3];  // Aceleración mandada en radianes
float Kp = receivedArray[4];
float Ki = receivedArray[5];
float Kv = receivedArray[6];
float Friction = receivedArray[8];  // Fricción recibida
float J = receivedArray[9];  // Inercia recibida
float mass = receivedArray[10];  // Masa recibida

TrajectoryParams traj = calculateTrajectory(q0_deg, qT_deg, vmax_rad, amax_rad);

// Ejecutar el bucle de control según los parámetros recibidos
controlLoop(DXL_ID, traj, Kp, Ki, Kv, J, mass, Friction);


do {


  // Envío de datos al ordenador
```

Imagen 73. Código que muestra el orden de acción del control (recepción de valores, cálculo de trayectoria y bucle de control)

Esta función “calculateTrajectory” crea una trayectoria trapezoidal para llegar a la posición objetivo según la posición actual del servomotor, la posición objetivo y la velocidad y aceleración solicitadas por el usuario:

```
// Cálculo de la trayectoria trapezoidal
TrajectoryParams calculateTrajectory(float q0_deg, float qT_deg, float vmax_rad, float amax_rad) {
  TrajectoryParams traj;

  traj.q0_deg = q0_deg;
  traj.qT_deg = qT_deg;
  traj.vmax_deg = vmax_rad * (180.0 / M_PI);
  traj.amax_deg = amax_rad * (180.0 / M_PI);

  float q_diff = traj.qT_deg - traj.q0_deg;
  traj.dir = (q_diff > 0) ? 1 : (q_diff < 0 ? -1 : 0);
  q_diff = fabs(q_diff);

  traj.T1 = traj.vmax_deg / traj.amax_deg;
  float q_accel = 0.5 * traj.amax_deg * traj.T1 * traj.T1;

  if (q_accel * 2 > q_diff) {
    traj.T1 = sqrt(q_diff / traj.amax_deg);
    traj.T2 = traj.T1;
    traj.vmax_deg = traj.amax_deg * traj.T1;
  } else {
    traj.T2 = (q_diff - 2 * q_accel) / traj.vmax_deg + traj.T1;
  }

  traj.T = 2 * traj.T1 + (traj.T2 - traj.T1);

  return traj;
}
```

Imagen 74. Función "calculateTrajectory"

Además, esta función primero calcula si se llegará en algún momento a la velocidad máxima, ya que, si la aceleración es demasiado baja, la velocidad objetivo muy alta o la posición a la que se pretende ir es muy cercana, en vez de hacer un trapezoide la velocidad tomará forma de triángulo.

Después de calcular la trayectoria se entra en el bucle de control, que es la función "controlLoop". En este bucle primero se establecen los valores necesarios para el cálculo de la acción de control:

```
void controlLoop(const uint8_t DXL_ID, const TrajectoryParams& traj, float Kp, float Ki, float Kv, float J, float mass, float Friction) {
  float t0 = millis() / 1000.0f;
  float t = 0;

  long previousTime = millis();
  float ePrevious = 0;
  float eIntegral = 0;
  float e_velocity = 0;
  const float G = 9.81f; // Gravedad
  const float Xc = 0.112f; //112 mm (longitud del eslabón)
  const float Kt = 0.354f; // Constante del par
```

Imagen 75. Inicio de la función "controlLoop"

Después se entra en el bucle que tiene dos opciones para finalizar: o ha recibido la orden de parar o el tiempo de la trayectoria ha acabado (se deja más tiempo de trayectoria para ver el error estacionario y dejar al servomotor alcanzar el objetivo, ya que la respuesta no es inmediata).

```
while (Serial.available() <= 0 && t < (traj.T * 2)) {
  t = (millis() / 1000.0f) - t0;

  PositionParam pos = calculatePosition(traj, t);

  // Calcular cuanto tiempo ha pasado en segundos del anterior ciclo
  float currentTime = millis();
  float deltaT = (currentTime - previousTime) / 1000.0f;

  //Lecturas del servo
  float q_real = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
  float v_real = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);

  // Calculo del error
  float e = pos.q_actual - q_real ;

  eIntegral = eIntegral + e * deltaT;
  e_velocity = (pos.v_actual * 60 / 360) - v_real ;

  // Calculo del par necesario para mover el servo

  float gravity_compensation = mass * G * Xc * cos(q_real * (M_PI / 180.0));
  float acceleration = J * (traj.amax_deg * (M_PI / 180.0) );
  float Friction_compensation = - Friction * sign(v_real); // Da igual unidades dado que la funcion signo solo devuelve el valor
  float control_output = (Kp * e) + (Kv * e_velocity) + (Ki * eIntegral) + Friction_compensation + gravity_compensation + acceleration ; //torque


  // Limita el valor de salida de control para que sea cercano al anterior

  float delta_output = control_output - previous_output;
  if (fabs(delta_output) > max_rate_of_change) {
    delta_output = max_rate_of_change * (delta_output / fabs(delta_output));
  }

  control_output = previous_output + delta_output;


  // Actualización de los valores
  previous_output = control_output;
  previousTime = currentTime;
  ePrevious = e;

  // Instrucciones al servo

  float current_goal = control_output / Kt; // Se convierte el par necesario a la corriente necesaria
  dxl.setGoalCurrent(DXL_ID, current_goal, UNIT_MILLI_AMPERE);
```

Imagen 76. Bucle de control en la función "controlLoop"

En este bucle además se pueden destacar varios elementos. El primero de todo es la función de “calculatePosition” que devuelve que valor de posición debería tener el servo en ese tiempo “t”:

```
// Función para calcular la posición en el tiempo t según la trayectoria
PositionParam calculatePosition(const TrajectoryParams& traj, float t) {
  PositionParam pos;
  if (t < traj.T1) { // Etapa de aceleración
    pos.q_actual = traj.q0_deg + 0.5f * traj.dir * traj.amax_deg * t * t;
    pos.v_actual = traj.dir * traj.amax_deg * t;
  } else if (t < traj.T2) { // Etapa de velocidad constante
    pos.q_actual = traj.q0_deg + 0.5f * traj.dir * traj.amax_deg * traj.T1 * traj.T1 + traj.dir * traj.vmax_deg * (t - traj.T1);
    pos.v_actual = traj.dir * traj.vmax_deg;
  } else if (t < traj.T) { //Etapa de desaceleración
    float t_dec = t - traj.T2;
    pos.q_actual = traj.q0_deg + 0.5f * traj.dir * traj.amax_deg * traj.T1 * traj.T1 + traj.dir * traj.vmax_deg * (traj.T2 - traj.T1) + traj.dir * traj.vmax_deg * t_dec - 0.5f * traj.dir * traj.amax_deg * t_dec * t_dec;
    pos.v_actual = traj.dir * traj.vmax_deg - traj.dir * traj.amax_deg * t_dec;
  } else if (t > traj.T) { //Etapa donde se debería haber llegado ya a la posición objetivo
    pos.q_actual = traj.qT_deg;
    pos.v_actual = 0;
  }
  return pos;
}
```

Imagen 77. Función "calculatePosition"

Después de calcular que posición y velocidad objetivo que debería haber en ese instante, se detectan la posición y la velocidad reales. Con estos valores se calcula el error de posición (posición de referencia – posición real), se añade el error de posición a la acumulación del error integral y por último se calcula el error de velocidad/derivativo. El error derivativo equivale a calcular el error de velocidad ya que la derivada de la posición equivale a la velocidad:

ERROR/ VARIABLE CONTOLADA	PROPORCIONAL	INTEGRAL	DERIVATIVO
POSICIÓN	$q_{ref} - q$	$\int q_{ref} - q)dt$	$\dot{q}_{ref} - \dot{q}$
VELOCIDAD	$\dot{q}_{ref} - \dot{q}$	$q_{ref} - q$	$\ddot{q}_{ref} - \ddot{q}$

Imagen 78. Equivalencia entre el control de posición y velocidad

Por último, después de calcular los errores y añadir las compensaciones, ese error es el cálculo del par de fuerza ya que la compensación de gravedad, aceleramiento y rozamiento afectan a la fuerza que ha de ejercer el motor. Este valor ha de ser convertido a la corriente necesario se ha de transmitir al motor, por lo que es necesario dividir el valor calculado entre la constante de par (Kt). Para clarificar, se ha diseñado una aproximación de la función de control en un esquema de bloques realizado en Simulink:

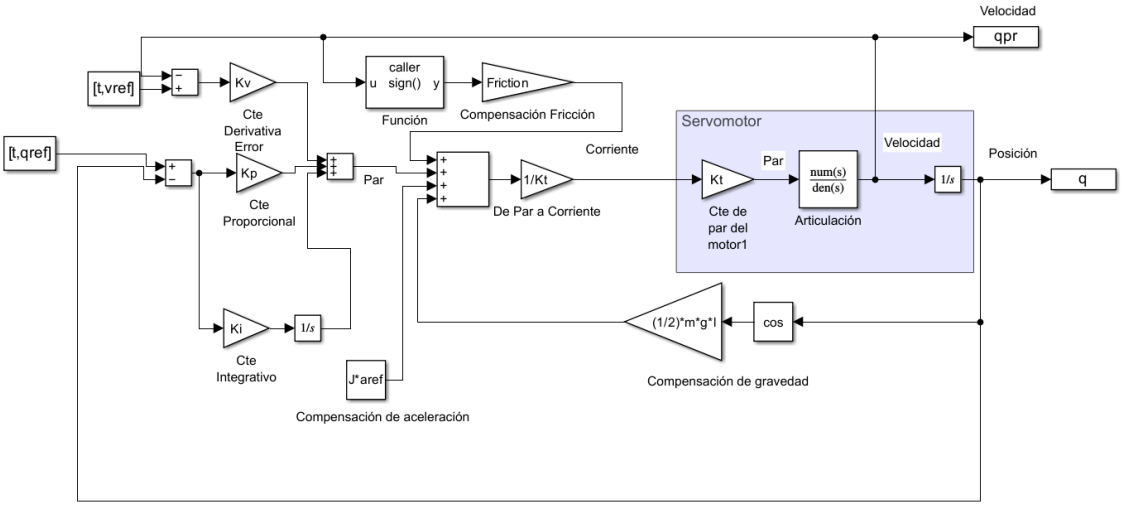


Imagen 79. Esquema del control Posición por corriente

Entonces se hace una demostración para ir a la posición 120 º con una P de 8, una I de 1 (la acción integral produce muchas oscilaciones, por lo que tener un valor bajo ayuda a la estabilidad de la trayectoria) y un valor de Kv de 20.

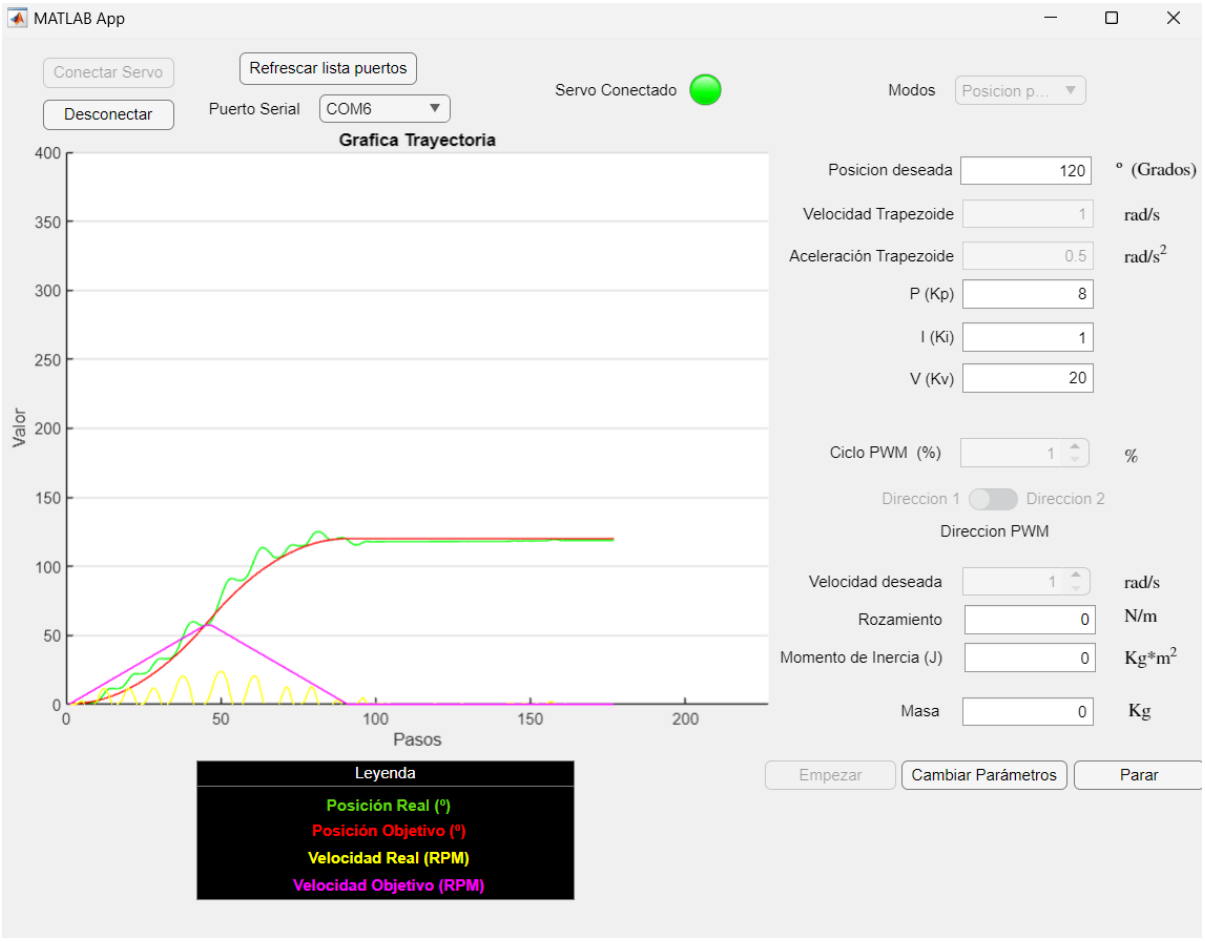


Imagen 80. Trayectoria de Posición por corriente a 120 º sin compensación

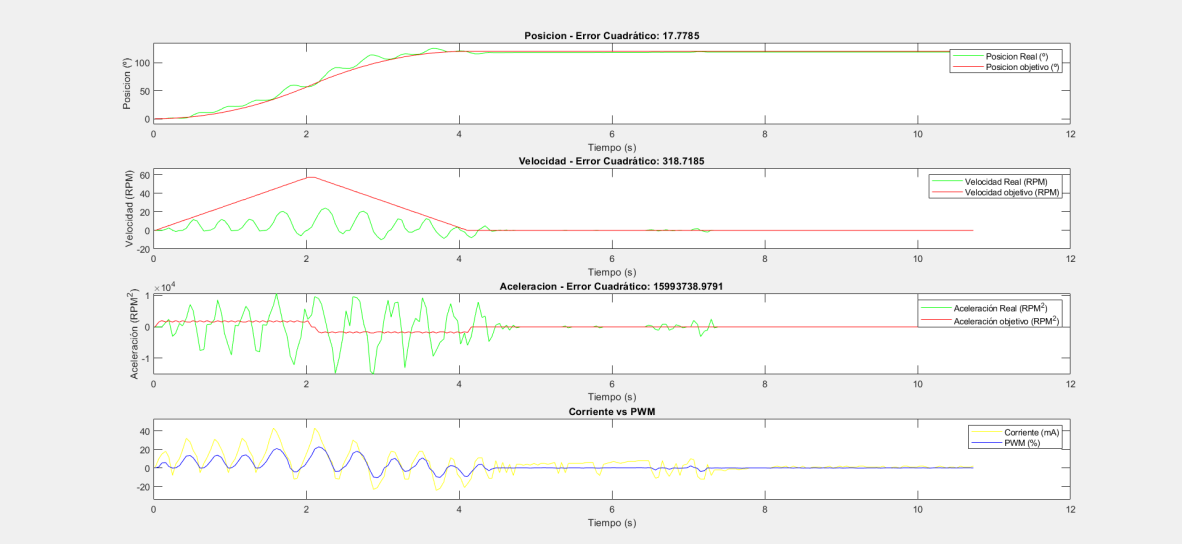


Imagen 81. Gráficas de la Imagen 80

Como se puede observar en las gráficas, se consigue hacer un control aproximado de la posición, pero al comparar las velocidades no es muy preciso, esto se debe a que este es un control de la posición y para conseguir un valor decente de la velocidad habría que hacer un control de ella también, como pueden ser los modos anteriores ya explicados. Esta solución no se ha implementado debido a que la práctica que se pretende simular es un control simple de la posición, por lo que el control implementado ha de ser relativamente parecido al implementado en el laboratorio.

En el caso anterior no se había introducido ninguna compensación (los valores de Rozamiento y Momento de Inercia estaban a 0), por lo que ahora se demuestra el efecto que tienen en la trayectoria:

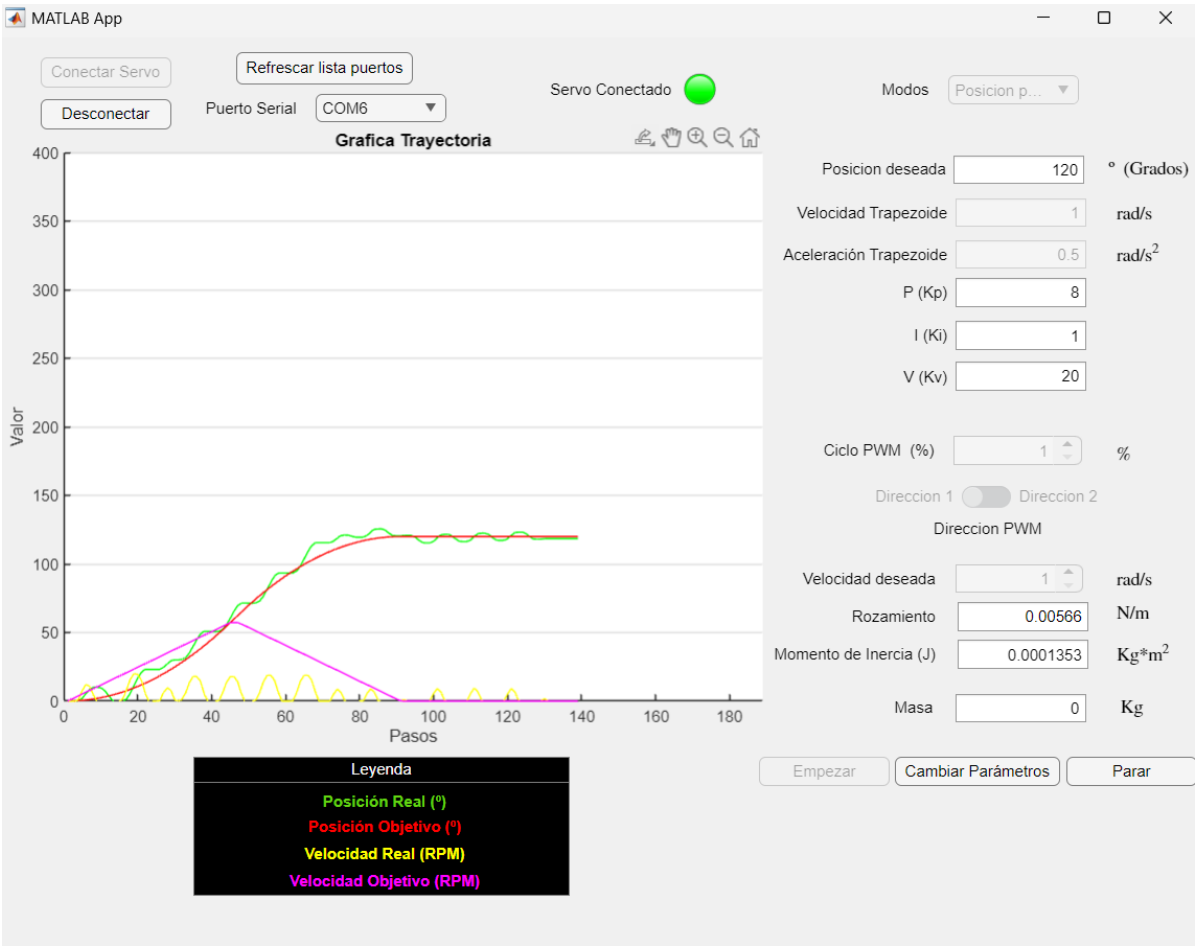


Imagen 82. Trayectoria de 120 º con Compensación de aceleración y rozamiento

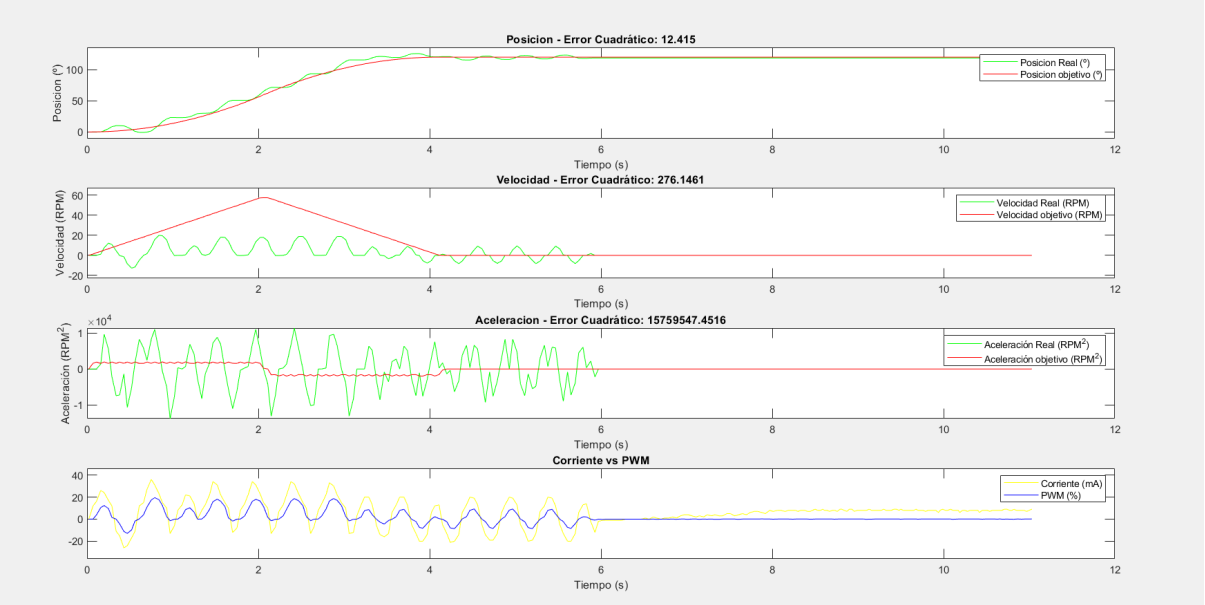


Imagen 83. Gráficas de la Imagen 82

Gracias a las compensaciones se obtiene una respuesta más estable, como se comprueba en las gráficas de las imágenes 82 y 83.

En este modo además hay una compensación de la gravedad que no se ha introducido en los ejemplos anteriores, ya que todas las trayectorias eran sin gravedad. Una trayectoria sin gravedad se refiere a que la gravedad afecta equitativamente en todos los puntos del recorrido, esto sucede cuando la herramienta está en posición horizontal, es decir, la base del prototipo actúa como la parte inferior de la estructura:

Imagen 84. Prototipo modo horizontal

En el siguiente caso por probar de trayectoria, la herramienta está apoyada sobre una de sus paredes, en concreto sobre el lado derecho, como en la imagen:

Imagen 85. Prototipo modo vertical

Además, se le ha introducido una masa de 131 gramos:

Imagen 86. Medida de la masa de 5 placas metálicas

Entonces, se procede al recorrido con la compensación activa de la gravedad a una posición algo complicada, 120 º:

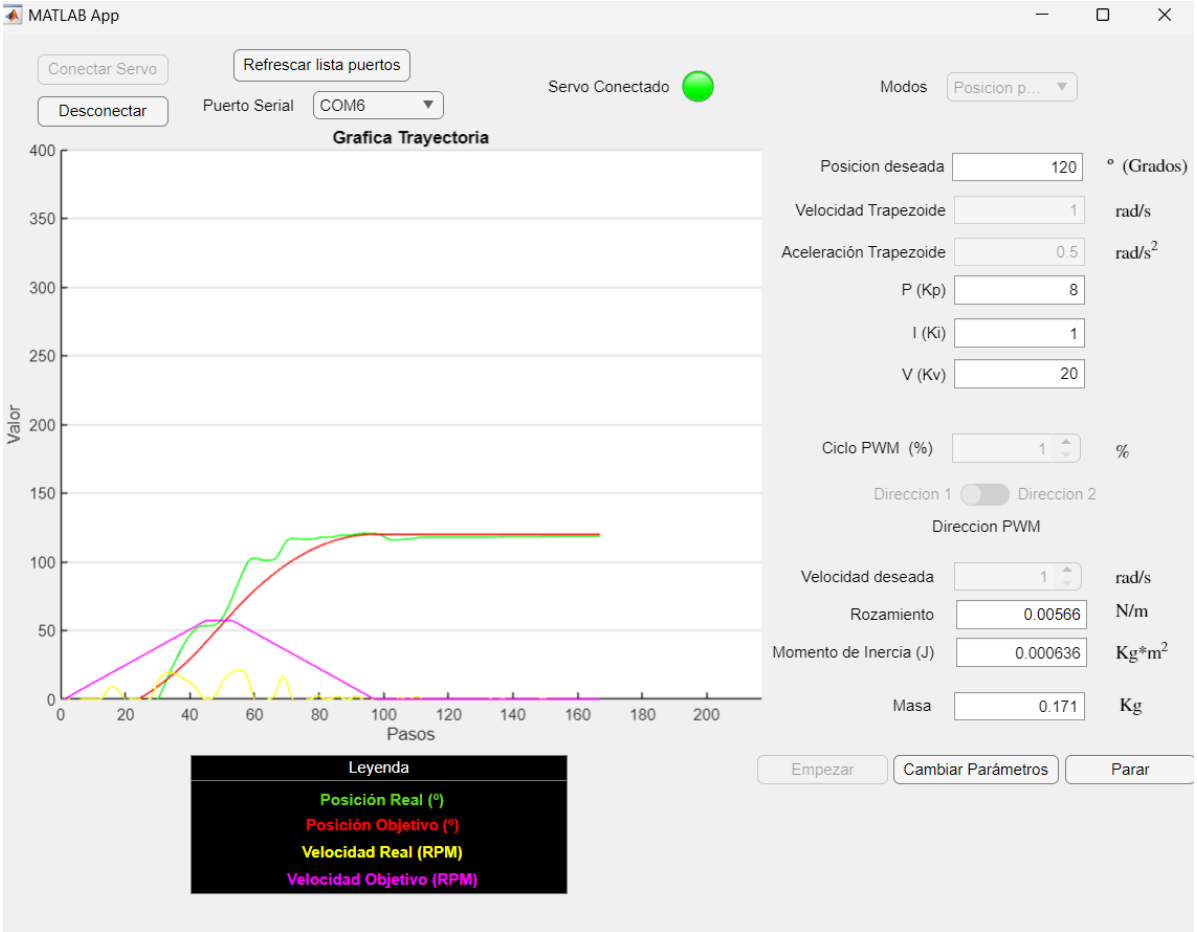


Imagen 87. Trayectoria a 120 º compensando la gravedad

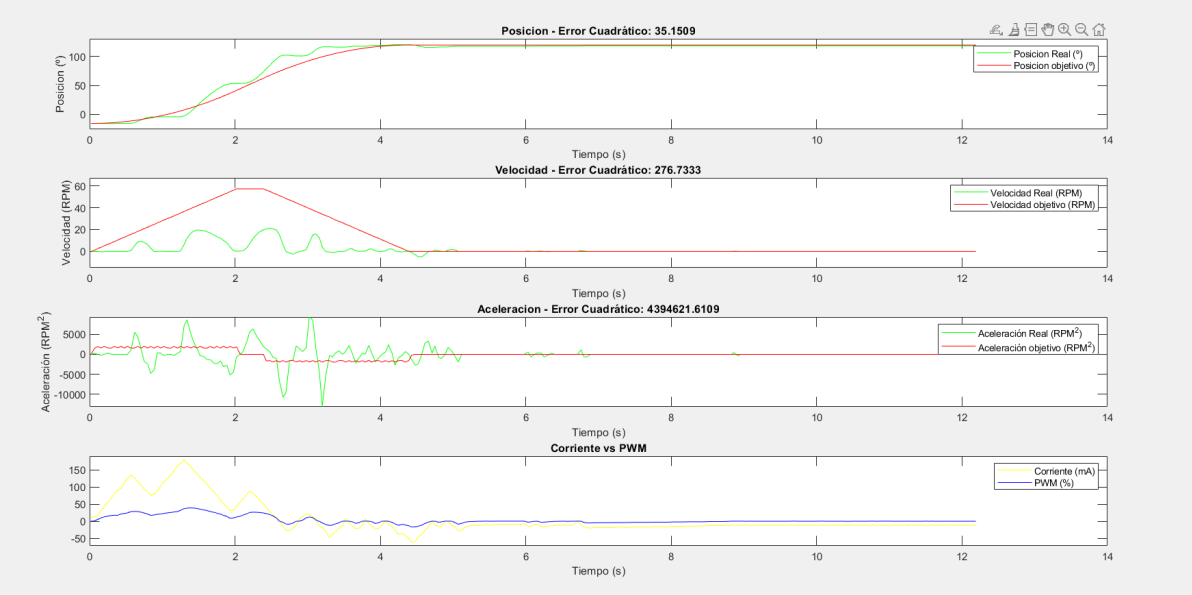


Imagen 88. Gráfica Imagen 87

Como se puede observar en base a las gráficas 87 y 88, el servomotor ha hecho el movimiento correctamente, teniendo una sobreoscilación relativamente baja sobretodo considerando que hay muy poca oscilación por encima de la posición final. En la realidad, el servomotor ha hecho el siguiente movimiento en sentido contrario a las agujas del reloj:

Imagen 89. Servomotr en posición 0 °

Imagen 90. Servomotor posición 120 º

5.4.8 Modo posición por corriente (PV)

Este modo es una modificación del modo por corriente anterior, por lo que es un modo programado para crear un control de posición en el que el servomotor controle la corriente proporcionada y eso ajuste la posición final. En este modo los valores configurables son:

- Posición deseada
- Velocidad del trapezoide
- Aceleración del trapezoide
- Constante Proporcional (Kp)
- Constante Derivativa o de velocidad (Kv)
- Constante de rozamiento del servomotor
- Momento de Inercia de la pieza (J)
- Masa de la pieza

Este modo surge debido a que en robótica normalmente con control PD es realmente suficiente para conseguir una respuesta satisfactoria de los actuadores. En este modo, además, se usa una versión del control en la que no se tiene en cuenta la velocidad de referencia, por eso es llamado PV. Este tipo de control produce una respuesta más estable y con menos sobreoscilación. El esquema de bloques actualizado se corresponde a este:

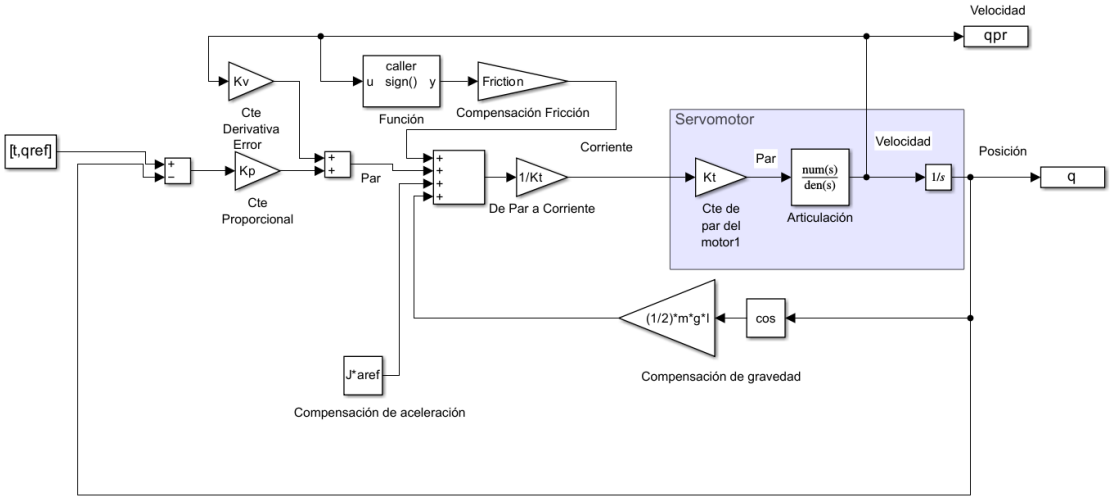


Imagen 91. Esquema de bloques del control PV

El esquema de la Imagen 91 se traduce en la acción de control programado de siguenta manera en la función “controlLoopV”:

```
while (Serial.available() <= 0 && t < (traj.T * 2)) {
  t = (millis() / 1000.0f) - t0;

  PositionParam pos = calculatePosition(traj, t);

  // Calcular cuanto tiempo ha pasado en segundos del anterior ciclo
  float currentTime = millis();
  float deltaT = (currentTime - previousTime) / 1000.0f;

  //Lecturas del servo
  float q_real = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
  float v_real = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);

  // Calculo del error
  float e = pos.q_actual - q_real ;

  //eIntegral = eIntegral + e * deltaT;

  e_velocity = (v_real)*0.01;

  // Calculo del par necesaria para mover el servo

  float gravity_compensation = 0.5*mass * G * Xc * cos(q_real * (M_PI / 180.0));
  float acceleration = J * (traj.amax_deg * (M_PI / 180.0));
  float Friction_compensation = - Friction * sign(v_real); // da igual unidades dado que la funcion signo solo devuelve el valor
  float control_output = (Kp * e) + (Kv * e_velocity) + Friction_compensation + gravity_compensation + acceleration ; //torque


  // Limita el valor de salida de control para que sea cercano al anterior

  float delta_output = control_output - previous_output;
  if (fabs(delta_output) > max_rate_of_change) {
    delta_output = max_rate_of_change * (delta_output / fabs(delta_output));
  }
  control_output = previous_output + delta_output;


  // Actualización de los valores
  previous_output = control_output;
  previousTime = currentTime;
  ePrevious = e;

  // Instrucciones al servo

  float current_goal = control_output / Kt; // Se pasa de par a corriente
  dxl.setGoalCurrent(DXL_ID, current_goal, UNIT_MILLI_AMPERE);
```

Imagen 92. Código de la función “controlLoopV”

Como se puede observar, “e_velocity” solo tiene en cuenta la velocidad real pero, además, está multiplicado por 0.01 para limitar que efecto tiene en la acción, ya que sin este el valor que se tendría que poner en la acción de control ha de ser más pequeño. Esta multiplicación

actúa como una clarificación/simplificación a la hora de usar la herramienta.

Entonces, poniendo la herramienta en modo vertical y con 171 gramos de peso total en el eslabón (40 g del eslabón más 131 de las placas metálicas) se obtiene la siguiente gráfica:

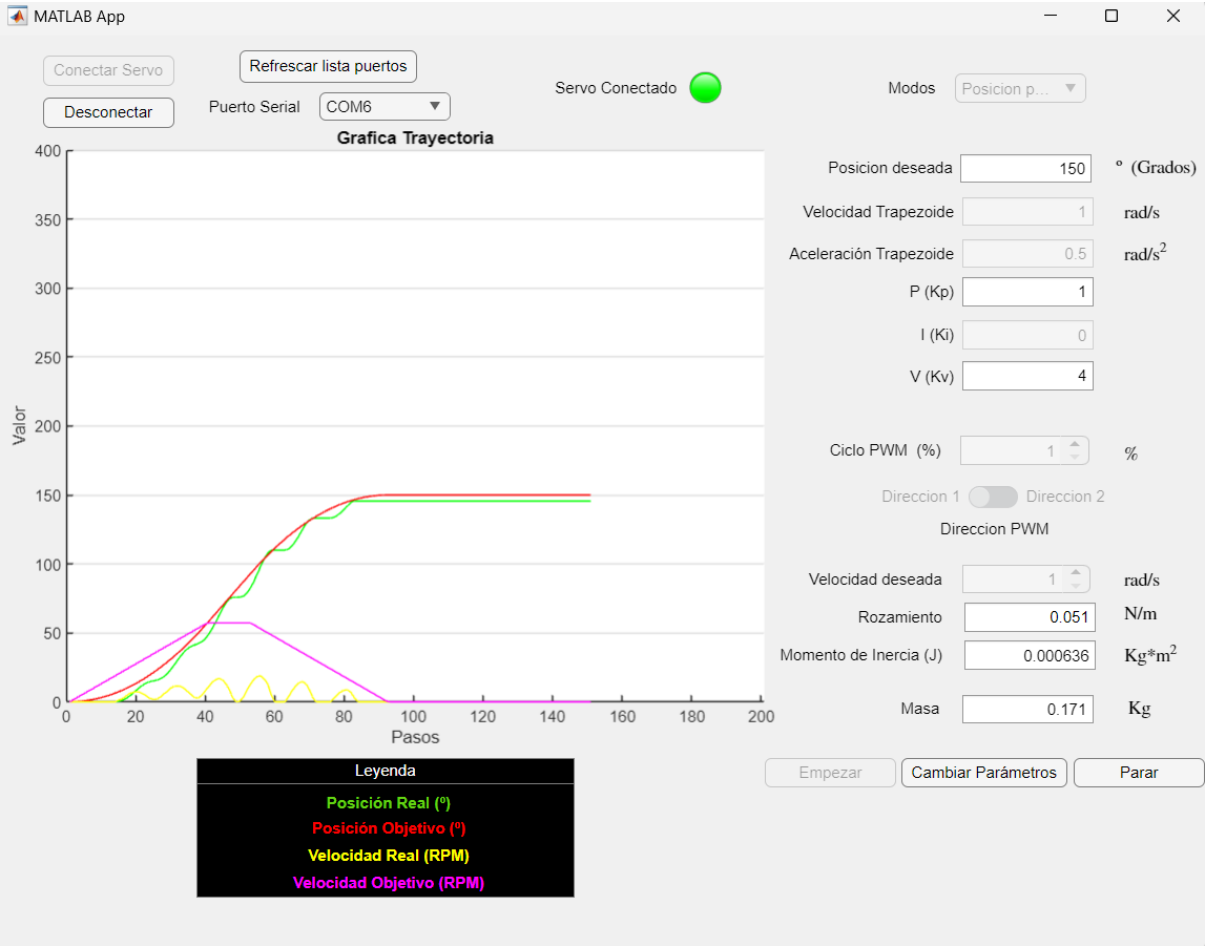


Imagen 93. Trayectoria del control PV a 150 º

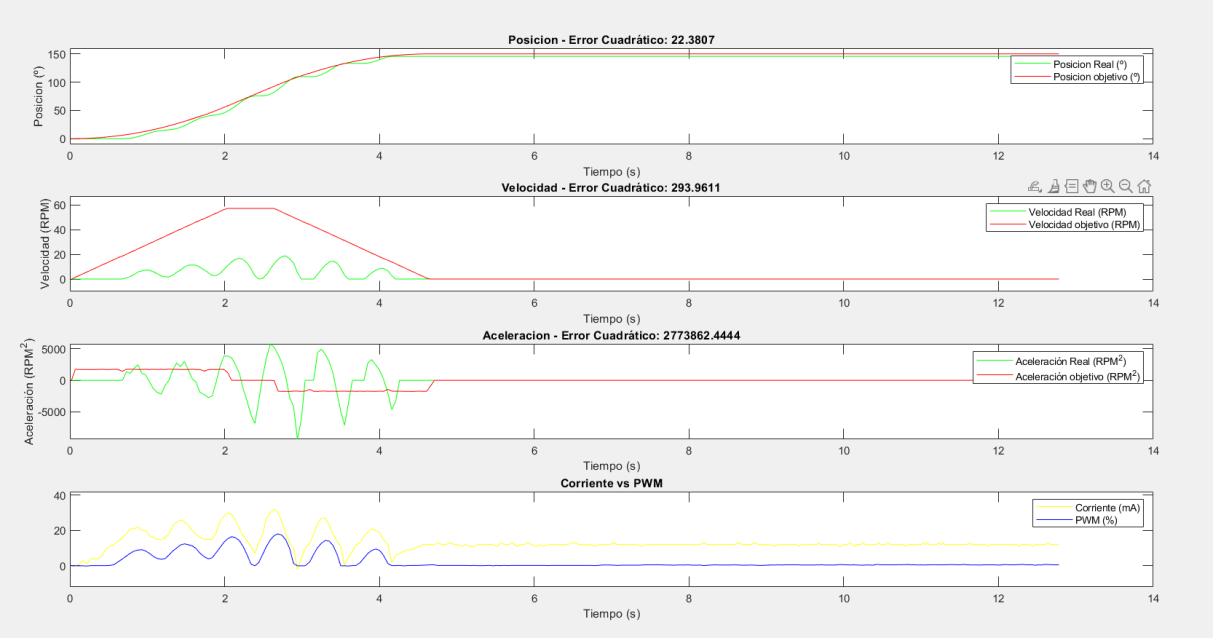


Imagen 94. Gráficas Imagen 93

Como se observa en las gráficas, se obtiene una respuesta más amortiguada, algo muy deseable en aplicaciones robóticas donde las sobreoscilaciones se tienden a evitar.

6 CÁLCULO DE LOS VALORES DE LA SOLUCIÓN ADOPTADA

Una vez presentadas todas las funcionalidades y capacidades de la herramienta de control y cómo se comporta el servomotor, se procede a explicar cómo se ha llegado a los valores de las constantes que aparecen en el programa mostrado.

6.1 Cálculo de la constante de par

La constante de par de un motor, generalmente conocida como Kt, es un valor que relaciona el par de fuerza del motor (llamado torque muchas veces) con su intensidad, por lo que es una medida de la eficiencia con la que el motor convierte la corriente eléctrica en par mecánico. La constante de par suele tener las unidades de N.m/A. La ecuación que relaciona el par del motor con su constante de par viene dada en la forma de:

$$\tau = K_t * I$$

τ: Par de motor (N. m)
K_t: Constante de par
I: Intensidad (A)

Usualmente, suele venir dada en las especificaciones técnicas de los motores, pero en el caso del XL330-M288-T no viene dada. Igualemente, se puede estimar a partir del par de parada que sí viene la hoja técnica del servomotor. El par de parada del servomotor es 0.52 N.m con una corriente de 1.47 A, por lo que se llega a la ecuación de:

$$K_t = \frac{\tau\ (N.m)}{I\ (A)} = \frac{0.52\ N.m}{1.47\ A} = 0.354\ N.m/A$$

Gracias a esta ecuación se puede calcular que intensidad es necesaria para los métodos de control programados:

$$Intensidad\ necesaria = \frac{\tau\ (par\ necesario\ calculado)}{K_t}$$

6.2 Cálculo del peso máximo del eslabón

El cálculo de que masa puede soportar el eslabón es uno de los más críticos de este proyecto, ya que una masa excesiva no solo puede dañar la herramienta sino poner en riesgo la seguridad de quién la emplee. Gracias al par de parada (0.52 N.m) se obtiene el esfuerzo que puede ejercer el servomotor a un metro de distancia. Esto, juntado al conocimiento de que cualquier fuerza es la masa del objeto por su aceleración, conduce a la ecuación:

$$\tau = G * m * d$$

τ: Par de motor (N. m)
G: Aceleración de la Gravedad ($\frac{\mathrm{m}}{s^2}$)
m: masa del objeto (Kg)
d: distanciaa a la que se encuentra el objeto (m)

Dividiendo el par de fuerza entre el efecto de la gravedad, obtenemos la masa que soporta el peso relacionada con su distancia:

$$\frac{\tau}{G} = m * d = \frac{0.52}{9.81} = 0.053\ Kg.m$$

Esta ecuación, combinada con el hecho de que el valor inicial para la longitud del eslabón eran 10 cm, nos lleva a calcular que masa puede soportar el servo a esa distancia:

$$0.053\ Kg.m * \frac{1\ cm}{0.01\ m} = 5.3\ Kg.cm$$

$$5.3\ Kg.cm * \frac{1}{10\ cm} = 0.53\ Kg$$

Teóricamente, el servomotor es capaz de soportar 530 gramos a 10 cm de distancia. Sin embargo, debido a que el eslabón tuvo que ser adaptado para poder encajar las placas metálicas, la longitud final del eslabón es 11.2 cm desde el centro de giro, por lo que el nuevo valor es:

$$5.3\ Kg.cm * \frac{1}{11.2\ cm} = 0.47\ Kg$$

Con el último resultado calculado se obtiene que como máximo se pueden soportar 470 g, por lo que la masa final utilizada es 171 g (5 placas metálicas más el peso del propio eslabón), un valor que obliga un esfuerzo al servomotor pero que no lo lleva a sus valores límites. Este valor es el usado en modos de posición por corriente, ya que para calcular la compensación de la gravedad es necesaria la masa del eslabón. Esta compensación es calculada para la acción de control en el modo de posición por corriente:

$$\tau_{control} = PID + \ compensación\ de\ fuerzas\ externas$$

$$compensación\ de\ fuerzas\ externas = c.rozamiento + c.gravedad + p.aceleración$$

$$compensación\ de\ gravedad = \frac{1}{2} * m * G * L * \cos(q)$$

τ: Par de motor necesario (N. m)
G: Aceleración de la Gravedad ($\frac{\mathrm{m}}{s^2}$)
m: Masa del objeto (Kg)
L: Longitud del eslabón

q: Ángulo actual del giro
$c.$: compensación de
$p.$: prealimentación de

La acción de control que regula la gravedad se corresponde con el comportamiento de la función de coseno:

- Cuando el ángulo es 0 es el momento donde más compensación debe suceder, por lo que el valor del coseno es 1, es decir, el valor máximo de la compensación.
- Cuando el coseno está entre los valores de 0 a 90 º y de 270 º a 360º la acción ha de ser positiva ya que es necesaria más compensación debido a que el eslabón está ascendiendo.
- En el caso contrario de 90 º a 270 º el servo está realizando un descenso, por lo que el valor de la compensación ha de ser negativo.

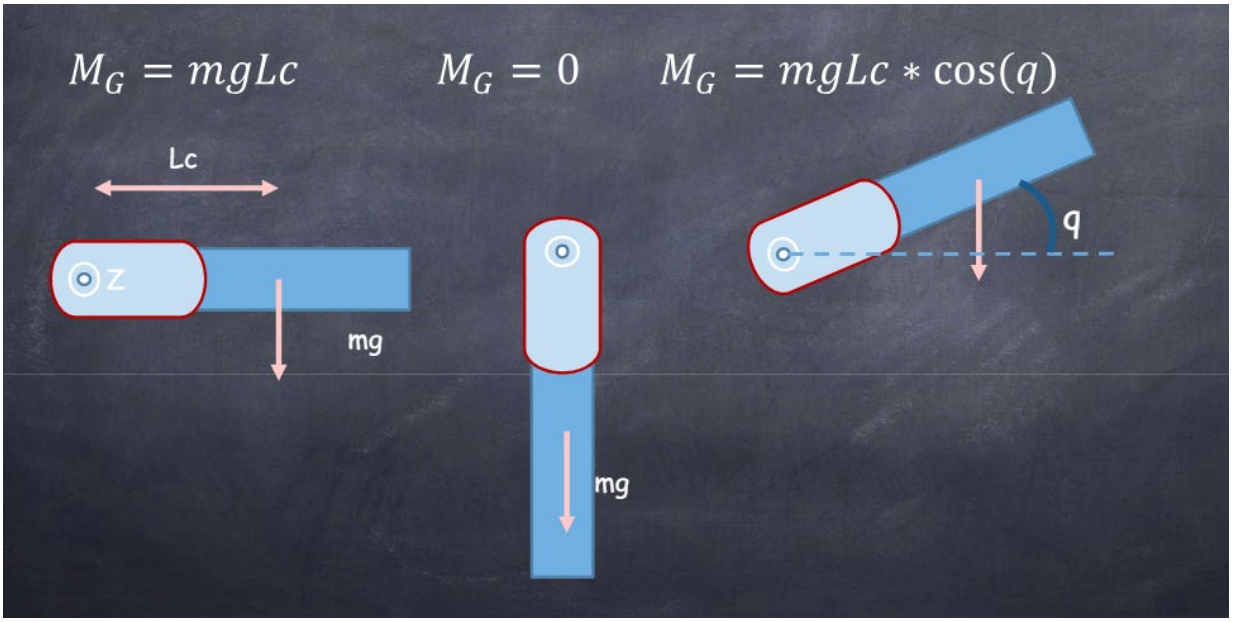


Imagen 95. Efecto de la fuerza de la gravedad y como lo compensa la función coseno

6.3 Cálculo de las inercias

Otra de las compensaciones que se introduce al servomotor en el modo de posición por corriente es la prealimentación de la aceleración, que viene dada por esta fórmula:

$$\tau_{control} = PID + \ compensación\ de\ fuerzas\ externas$$

$$compensación\ de\ fuerzas\ externas = c.rozamiento + c.gravedad + p.aceleración$$

$$prealimentación\ de\ aceleración = \ J * a$$

τ: Par de motor necesario (N. m)
a: Aceleración de referencia ($\frac{\mathrm{m}}{s^2}$)
J: Inercia del objeto ($\mathrm{Kg} * \frac{\mathrm{m}}{s^2}$)
$c.$: compensación de
$p.$: prealimentación de

Esta inercia ha sido calculada a través de Solidworks usando su cálculo de propiedades físicas. Primero de todo se hizo con la figura de la articulación, dando los siguientes resultados:

```
Propiedades de masa de Brazov2
    Configuración:  Predeterminado
    Sistema de coordenadas:  -- predeterminado --

Densidad = 969.259426 kilogramos por metro cúbico

Masa (reemplazada por el usuario) = 0.040000 kilogramos

Volumen = 0.000041 metros cúbicos

Área de superficie = 21770.292197  milímetros cuadrados

Centro de masa: ( milímetros )
     X = 41.647118
     Y = 25.082376
     Z = 0.000000

Ejes principales de inercia y momentos principales de inercia: ( kilogramos *  milímetros cuadrados )
Medido desde el centro de masa.
      Ix = ( 0.999956,  0.009390,  0.000000)          Px = 6.922812
      Iy = (-0.009390,  0.999956,  0.000000)          Py = 65.908447
      Iz = ( 0.000000,  0.000000,  1.000000)          Pz = 66.522711

Momentos de inercia: ( kilogramos *  milímetros cuadrados )
Obtenidos en el centro de masa y alineados con el sistema de coordenadas de resultados. (Usando notación tensorial positiva.)
     Lxx = 6.928013          Lxy = 0.553844          Lxz = 0.000000
     Lyx = 0.553844          Lyy = 65.903246         Lyz = 0.000000
     Lzx = 0.000000          Lzy = 0.000000          Lzz = 66.522711

Momentos de inercia: ( kilogramos *  milímetros cuadrados)
Medido desde el sistema de coordenadas de salida. (Usando notación tensorial positiva.)
     Ixx = 32.093036         Ixy = 42.338191         Ixz = 0.000000
     Iyx = 42.338191         Iyy = 135.282543        Iyz = 0.000000
     Izx = 0.000000          Izy = 0.000000          Izz = 161.067031
```

Imagen 96. Cálculo de las propiedades del eslabón realizado por Solidworks

De la figura anterior se obtiene que la inercia de la pieza (llamada Brazov2) es 135.28 Kg.mm^2 (valor de Iyy), o en unidades internacionales, 0.000135 kg*m^2.

En el caso de la pieza con las placas metálicas se construyó el ensamble del eslabón con 5 piezas (como se puede observar en la imagen):

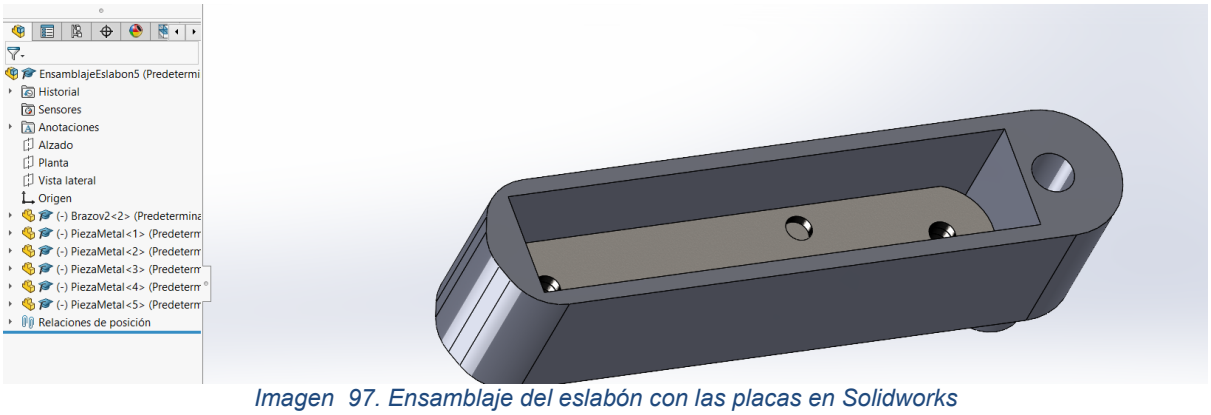


Imagen 97. Ensamblaje del eslabón con las placas en Solidworks

Del modelo de la Imagen 97 se volvió a calcular su inercia a través de SolidWorks:

```
Propiedades de masa de EnsamblajeEslabon5
    Configuración:  Predeterminado
    Sistema de coordenadas:  -- predeterminado --

Masa = 0.17 kilogramos

Volumen = 0.00 metros cúbicos

Área de superficie = 41719.15  milímetros cuadrados

Centro de masa: ( milímetros )
     X = 51.89
     Y = 20.72
     Z = 0.00

Ejes principales de inercia y momentos principales de inercia: ( kilogramos *  milímetros cuadrados )
Medido desde el centro de masa.
      Ix = ( 1.00, -0.01,  0.00)        Px = 12.22
      Iy = ( 0.00,  0.00, -1.00)        Py = 171.05
      Iz = ( 0.01,  1.00,  0.00)        Pz = 172.98

Momentos de inercia: ( kilogramos *  milímetros cuadrados )
Obtenidos en el centro de masa y alineados con el sistema de coordenadas de resultados. (Usando notación tensorial positiva.)
     Lxx = 12.24                Lxy = -1.82                Lxz = 0.11
     Lyx = -1.82                Lyy = 172.96               Lyz = 0.00
     Lzx = 0.11                 Lzy = 0.00                 Lzz = 171.05

Momentos de inercia: ( kilogramos *  milímetros cuadrados)
Medido desde el sistema de coordenadas de salida. (Usando notación tensorial positiva.)
     Ixx = 86.04                Ixy = 183.03               Ixz = 0.11
     Iyx = 183.03               Iyy = 635.97               Iyz = 0.00
     Izx = 0.11                 Izy = 0.00                 Izz = 707.85

Uno o más componentes han reemplazado las propiedades de masa:
     Brazov2<2><Predeterminado>
```

Imagen 98. Cálculo de las propiedades del eslabón con las placas realizado por Solidworks

Como se observa en la Imagen 98, la masa en este caso es 0.17 kg y se obtiene la inercia de la pieza en el valor de Iyy, que es 635.97 Kg.mm^2, o en unidades internacionales, 0.000636 kg*m^2.

6.4 Cálculo de las relaciones de tensión y corriente con la velocidad

La herramienta tiene dos modos personalizados en los que el usuario escoge la velocidad de referencia y entonces a través de un control PID la acción de control ajusta los valores de tensión o corriente para conseguir la velocidad objetivo. Para hacer este tipo de control más preciso es necesario encontrar como esos parámetros se relacionan con la velocidad, como el procedimiento para los dos es el mismo se explican en el mismo apartado. Todos los datos conseguidos son con una carga en el servo de 171 g, ya que se ha comprobado que estos valores son correctos también con solo la carga del eslabón (40 g).

El primer paso es explicar cómo se consigue la constante de velocidad de la corriente (“K_current” en el programa final) de forma experimental. Este se lleva a cabo creando un programa de Arduino en el que se compare la corriente con la velocidad a la vez que se incrementa la corriente, como en el de esta imagen:

```
for (int current = 0; current <= 500; current = current+25) {
  dxl.setGoalCurrent(DXL_ID, current,UNIT_MILLI_AMPERE);
  delay(1000); // Tiempo para asegurarse que el valor de corriente se ha alcanzado
  float velocity = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
  Serial.print("Corriente: ");
  Serial.print(current);
  Serial.print(" -> Velocidad: ");
  Serial.println(velocity);
  showcurrent[a]=current;
  showvelocity[a]=velocity;
  a++;
}
delay(1000);
//Se reduce la velocidad a 0 de forma incremental
for (float current = 500; current >= 0; current= current-50){
dxl.setGoalCurrent(DXL_ID, current,UNIT_MILLI_AMPERE);
delay(150);
}
  Serial.print("current_values = [");
  for (int i=0; i <a; i++){
    Serial.print(showcurrent[i]);
    Serial.print(", ");
    }
    Serial.println("];");
  Serial.print("velocity_values_current = [");
  for (int i=0; i <a; i++){
    Serial.print(showvelocity[i]);
    Serial.print(", ");
    }
    Serial.println("];");
```

Imagen 99. Función while del código de "CurrentvsVelocity"

Este código, llamado “CurrentvsVelocity”, incrementa la corriente objetivo en saltos de 25 mA cada segundo hasta llegar a 500 mA, entoncecs los almacena en una variable que después envía al monitor Serial:

```
Empieza
Corriente: 0 -> Velocidad: 0.00
Corriente: 25 -> Velocidad: 20.15
Corriente: 50 -> Velocidad: 39.16
Corriente: 75 -> Velocidad: 64.58
Corriente: 100 -> Velocidad: 85.42
Corriente: 125 -> Velocidad: 103.51
Corriente: 150 -> Velocidad: 106.03
Corriente: 175 -> Velocidad: 107.63
Corriente: 200 -> Velocidad: 107.17
Corriente: 225 -> Velocidad: 104.88
Corriente: 250 -> Velocidad: 104.65
Corriente: 275 -> Velocidad: 106.71
Corriente: 300 -> Velocidad: 107.17
Corriente: 325 -> Velocidad: 106.03
Corriente: 350 -> Velocidad: 103.74
Corriente: 375 -> Velocidad: 105.80
Corriente: 400 -> Velocidad: 107.17
Corriente: 425 -> Velocidad: 106.71
Corriente: 450 -> Velocidad: 104.88
Corriente: 475 -> Velocidad: 103.74
Corriente: 500 -> Velocidad: 105.80
current_values = [0.00, 25.00, 50.00, 75.00, 100.00, 125.00, 150.00, 175.00, 200.00, 225.00, 250.00, 275.00, 300.00, 325.00, 350.00, 375.00, 400.00, 425.00, 450.00, 475.00, 500.00, ];
velocity_values_current = [0.00, 20.15, 39.16, 64.58, 85.42, 103.51, 106.03, 107.63, 107.17, 104.88, 104.65, 106.71, 107.17, 106.03, 103.74, 105.80, 107.17, 106.71, 104.88, 103.74, 105.80, ];
```

Imagen 100. Captura del monitor Serial de la Imagen 99

Cómo se observa en la imagen, a partir de 200 mA ya se alcanza la velocidad máxima del servomotor por lo que a partir de ese valor no es significativo.

El método con la tensión es el mismo, un incremento del PWM y que se guarden los valores de la velocidad en cada incremento, que es lo que realiza el código de “PWMvsVelocity”:

```
for (float pwm = 0; pwm <= 100; pwm= pwm+10) {
  dxl.setGoalPWM(DXL_ID, pwm,UNIT_PERCENT);
  delay(1000); // Tiempo para asegurarse que el valor del PWM se ha alcanzado
  float velocity = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
  Serial.print("PWM: ");
  Serial.print(pwm);
  Serial.print(" -> Velocidad: ");
  Serial.println(velocity);
  Serial.print(" -> Corriente: ");
  Serial.println(dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE));
  showpwm[a]=pwm;
  showvelocity[a]=velocity;
  a++;
}
delay(1000);
//Se reduce la velocidad a 0 de forma gradual
for (float pwm = 100; pwm >= 0; pwm= pwm-10){
dxl.setGoalPWM(DXL_ID, pwm,UNIT_PERCENT);
delay(150);
}
  Serial.print("pwm_values = [");
  for (int i=0; i <a; i++){
    Serial.print(showpwm[i]);
    Serial.print(", ");
    }
    Serial.println("];");
  Serial.print("velocity_values_pwm = [");
  for (int i=0; i <a; i++){
    Serial.print(showvelocity[i]);
    Serial.print(", ");
    }
    Serial.println("];");
```

Imagen 101. Código que muestra el bucle de "PWMvsVelocity"

```
Empieza
PWM: 0.00 -> Velocidad: 0.00
 -> Corriente: 0.00
PWM: 10.00 -> Velocidad: 8.24
 -> Corriente: 17.00
PWM: 20.00 -> Velocidad: 21.76
 -> Corriente: 24.00
PWM: 30.00 -> Velocidad: 30.23
 -> Corriente: 41.00
PWM: 40.00 -> Velocidad: 43.28
 -> Corriente: 48.00
PWM: 50.00 -> Velocidad: 54.50
 -> Corriente: 59.00
PWM: 60.00 -> Velocidad: 65.04
 -> Corriente: 67.00
PWM: 70.00 -> Velocidad: 74.20
 -> Corriente: 86.00
PWM: 80.00 -> Velocidad: 87.02
 -> Corriente: 88.00
PWM: 90.00 -> Velocidad: 94.12
 -> Corriente: 125.00
PWM: 100.00 -> Velocidad: 106.03
 -> Corriente: 119.00
pwm_values = [0.00, 10.00, 20.00, 30.00, 40.00, 50.00, 60.00, 70.00, 80.00, 90.00, 100.00, ];
velocity_values_pwm = [0.00, 8.24, 21.76, 30.23, 43.28, 54.50, 65.04, 74.20, 87.02, 94.12, 106.03, ];
```

Imagen 102. Salida del Monitor Serial de la Imagen 101

En el caso de PWM, no hay ningún caso poco significativo ya que se percibe un incremento en cada aumento de PWM.

Con los valores experimentales conseguidos, se procede al código de Matlab de "Calculo_constantes":

```
% Datos
pwm_values = [0.00, 10.00, 20.00, 30.00, 40.00, 50.00, 60.00, 70.00, 80.00, 90.00, 100.00];
velocity_values_pwm = [0.00, 8.24, 21.76, 30.23, 43.28, 54.50, 65.04, 74.20, 87.02, 94.12, 106.03];

current_values = [0.00, 25.00, 50.00, 75.00, 100.00, 125.00, 150.00, 175.00, 200.00];
velocity_values_current = [0.00, 20.15, 39.16, 64.58, 85.42, 103.51, 106.03, 107.63, 107.17];

%Calculo de K_current
sum=0;

for i=2:1:length(pwm_values) %El primer valor es 0 por lo que no es necesario contabilizarlo, además crea una número NaN
    relation=pwm_values(i)/velocity_values_pwm (i);
    sum= sum+ relation;
end

K_PWM= sum/length(pwm_values);

%Calculo de K_current
sum=0;

for i=2:1:length(current_values)
    relation=current_values(i)/velocity_values_current(i);
    sum= sum+ relation;
end

K_current= sum/length(current_values);

% Resultados de constantes
fprintf('K_PWM= %.3f\n', K_PWM);
fprintf('K_current= %.3f\n', K_current);
```

Imagen 103. Código de Matlab de "Calculo_constantes"

Este es un simple código que divide cada valor de corriente y tensión por su valor de velocidad correspondiente y se hace la media de todos esos valores. Este código tiene como salida:

```
Command Window
   K_PWM= 0.877
   K_current= 1.218
fx >>
```

Imagen 104. Salida de la Imagen 103

Por lo que se obtienen las dos ecuaciones de:

$$I\ (mA)\ = Acción\ de\ control * K_{current}$$

$$PWM\ (\%)\ = Acción\ de\ control * K_{PWM}$$

I: Intensidad objetivo que se transmite al servo (mA)
PWM: PWM objetivo que se transmite al servo (%)

6.5 Cálculo de la compensación de rozamiento

El rozamiento de un motor es una las perturbaciones más comunes en un sistema de aplicación real ya que siempre que se usen dos mecanismos o un movimiento, existirá un rozamiento que hace falta tener en cuenta si se quiere tener un sistema preciso. Debido a este motivo, a la acción de control PID del modo de posición por corriente se le añade la compensación del rozamiento. Como aclaración, las fuerzas externas es siempre la compensación del rozamiento y las demás (en el caso aceleración y gravedad) se añaden en sus casos específicos. En la acción de control calculada para el sistema de posición por corriente, la fórmula del par de control sigue la siguiente forma:

$$\tau_{control} = PID + \ compensación\ de\ fuerzas\ externas$$

$$compensación\ de\ fuerzas\ externas = c.rozamiento + c.gravedad + p.aceleración$$

$$compensación\ de\ rozamiento = \ -Fc * sign(v)$$

Fc: Constante de rozamiento (N. m)
v: velocidad actual del servomotor
$sign$: Función matemática que devuelve el signo
$c.$: compensación de
$p.$: prealimentación de

Esta fórmula incluye la función de sign(), que en práctica lo que hace es devolver el signo de la velocidad siguiendo esta lógica:

- Si el valor de v es mayor que 0 la función devuelve 1
- Si el valor de v es 0 la función devuelve 0
- Si el valor de v es menor que 0 la función devuelve -1

```
// Función que devuelve el signo de un valor
float sign(float value) {
  float x;
  if (value > 0) {
    x = 1;
  } else if ( value < 0 ) {
    x = -1;
  } else {
    x = 0;
  }
  return x;
}
```

Imagen 105. Código de Arduino de la función Sign

El término interesante y necesario de conseguir es la constante de rozamiento, que se ha obtenido de forma similar que las constantes de velocidad. Para conseguir el valor de la constante se han hecho incrementos del ciclo del PWM hasta que se ha detectado velocidad en el servomotor. Teniendo en cuenta que en el apartado anterior se ha observado que entre un ciclo del 0% y del 10% el servomotor vence el rozamiento, se modifica el código de "PWMvsVelocity" para hacer el incremento hasta 10:

```
for (float pwm = 0; pwm <= 10; pwm= pwm+1) {
  dxl.setGoalPWM(DXL_ID, pwm,UNIT_PERCENT);
  delay(1000); // Tiempo para asegurarse que el valor del PWM se ha alcanzado
  float velocity = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
  Serial.print("PWM: ");
  Serial.print(pwm);
  Serial.print(" -> Velocidad: ");
  Serial.println(velocity);
  Serial.print(" -> Corriente: ");
  Serial.println(dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE));
  showpwm[a]=pwm;
  showvelocity[a]=velocity;
  a++;
}
delay(1000);
//Se reduce la velocidad a 0 de forma gradual
for (float pwm = 10; pwm >= 0; pwm= pwm-1){
dxl.setGoalPWM(DXL_ID, pwm,UNIT_PERCENT);
delay(150);
}
  Serial.print("pwm_values = [");
  for (int i=0; i <a; i++){
    Serial.print(showpwm[i]);
    Serial.print(", ");
    }
    Serial.println("];");
  Serial.print("velocity_values_pwm = [");
  for (int i=0; i <a; i++){
    Serial.print(showvelocity[i]);
    Serial.print(", ");
    }
    Serial.println("];");
```

Imagen 106. Código de Arduino "PWMvsVelocity" modificado

Ejecutando el código anterior, se obtiene el siguiente resultado:

```
Empieza
PWM: 0.00 -> Velocidad: 0.00
 -> Corriente: 0.00
PWM: 1.00 -> Velocidad: 0.00
 -> Corriente: 12.00
PWM: 2.00 -> Velocidad: 0.00
 -> Corriente: 13.00
PWM: 3.00 -> Velocidad: 0.00
 -> Corriente: 15.00
PWM: 4.00 -> Velocidad: 2.75
 -> Corriente: 16.00
PWM: 5.00 -> Velocidad: 4.81
 -> Corriente: 14.00
PWM: 6.00 -> Velocidad: 5.27
 -> Corriente: 17.00
PWM: 7.00 -> Velocidad: 5.72
 -> Corriente: 17.00
PWM: 8.00 -> Velocidad: 5.95
 -> Corriente: 20.00
PWM: 9.00 -> Velocidad: 8.47
 -> Corriente: 19.00
PWM: 10.00 -> Velocidad: 10.08
 -> Corriente: 20.00
pwm_values = [0.00, 1.00, 2.00, 3.00, 4.00, 5.00, 6.00, 7.00, 8.00, 9.00, 10.00, ];
velocity_values_pwm = [0.00, 0.00, 0.00, 0.00, 2.75, 4.81, 5.27, 5.72, 5.95, 8.47, 10.08, ];
```

Imagen 107. Salida del monitor serial de la Imagen 106

Viendo los resultados obtenidos se determina que, con una carga de 171 g ,cuando se instruye un ciclo de trabajo del 4 % el servo vence al rozamiento inicial del sistema y necesita una corriente de 16 miliamperios. Entonces, con la constante de par del motor Kt se convierte esa corriente en el par necesario para solventar el rozamiento. La fórmula final de la conversión es:

$$Fc = K_t * corriente\ de\ cambio = 0.354 * \frac{16\ mA}{1000} = 0.00566\ N.m$$

7 CONCLUSIONES Y PERSPECTIVA DE FUTURO

Con todo el desarrollo del TFG expuesto, se llega a la conclusión de que la implementación de la herramienta es satisfactoria y que llegará a ser una herramienta útil para prácticas donde se enseñe el control PID, ya que se han diseñado y programado todos los elementos y modos propuestos al inicio del proyecto obteniendo el producto solicitado al principio de la oferta pública.

Este trabajo de fin de grado ha constado de una etapa de aprendizaje y búsqueda sobre el marco teórico del control automático, una etapa del diseño de materiales y piezas que se ajustasen a las especificaciones para los casos necesarios, una etapa de programación que combina los dos lenguajes de programación más extendidos durante el grado y una etapa de resolución de problemas cuando el resultado experimental difería del resultado esperado, teniendo muchas veces que repetir todas las etapas mencionada. Debido a la complejidad de estas etapas, este proyecto puede considerarse uno integral para un ingeniero electrónico, ya que se ha requerido la aplicación de todas las habilidades adquiridas durante el grado para poder realizarlo con éxito.

Con perspectiva de futuro y comprendiendo los puntos débiles del proyecto, hay varias opciones para mejorarlo o ampliarlo. La primera mejora a implementar es usar sistemas de control más avanzados como un control de velocidad a la vez que el de posición o agregar

modificaciones avanzadas al control PID como “anti-windup”. Estos no han sido implementados en este proyecto ya que se han utilizado controles simples para mantener la coherencia con la práctica objetivo, pero sistemas más complejos podrían proporcionar mayor estabilidad y claridad en la observación y tratamiento de los sistemas si ese es el resultado más deseado.

Otra mejora posible es la implementación de varios servos para llegar a imitar el funcionamiento de una articulación robótica si fuese deseado. Esta mejora ayudaría a entender a los estudiantes como se relacionan varios servos en un sistema y como se han de modificar los valores de control para que todos los actuadores trabajen en armonía.

La última ampliación que se considera útil es usar un servomotor de mayor coste para tener resultados más precisos. Se podría plantear uno que tuviese un codificador de aceleración para que los resultados de las gráficas fuesen más reales. Otro motivo para el cambio es debido a la configuración interna del servomotor en la que para cambiar el modo hace falta desactivar su par de fuerza, lo que supone un peligro en situaciones con carga o donde haya gravedad.

En conclusión, este ha sido un trabajo que ha requerido un esfuerzo considerable y muchas horas de diseño y desarrollo, proporcionando un aprendizaje completo de todos los aspectos de la ingeniería electrónica y lo que fomenta el interés por seguir en la industria de la electrónica y, especialmente, en la robótica.

8 BIBLIOGRAFÍA

- *Amazon.es*. (s. f.). https://www.amazon.es/FandWay-Phillips-Rounded-Laptop-Repair/dp/B07XCB4MMW/ref=sr_1_4?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=1DQMFU0ZA8M3J&dib=eyJ2IjoiMSJ9.abNOWISliIPzg20JZHJA0MK0o_3Pt6ag2esJQ0cVLkqmb3nnrDw4q9fK22xMsIkIycGYtvfeV1EpofYDIhdgKY-4u110y8R4MyTKJBSr8CcVYoQA3ALka4HymgBvtcgg1xDQ2M_YHGE-jYHjJ5dqe_4jWEOiAvgqdIfRFbxNg-CzUiMYFv0BPWNyw9mvHCguNTXwk1IXVBfILN6_T1sX_G_UmeEcz6RvPvegcKOE5LY.WgTKnW0VquF977J3IgiXrfr5YKHkwAnbxpRS6xSGQ0s&dib_tag=se&keywords=m2%2C6&qid=1715629359&s=tools&sprefix=m2%2B6%2Cdiy%2C97&sr=1-4&th=1
- Anders, V. (s. f.). *SERVOMOTOR*. Etimologías de Chile - Diccionario Que Explica el Origen de las Palabras. https://etimologias.dechile.net/?servomotor#:~:text=La%20palabra%20viene%20del%20lat%C3%ADn,amplifica%20la%20potencia%20de%20regulaci%C3%B3n.&text=La%20palabra%20latina%20servus%20(esclavo,%2C%20estar%20al%20servicio%20de).

- *Arduino Nano v3.0 (compatible) – Robótica Fácil*. (s. f.). https://roboticafacil.es/?product=arduino-nano3-compatible-suelto
- Arias-García, A. (2021, 3 abril). *Ángulos, grados y radianes*. Totumat. https://totumat.com/2021/02/05/angulos-grados-y-radianes/
- Aula. (2023, 12 diciembre). Servomotores: héroes silenciosos de la tecnología moderna. *aula21 | Formación para la Industria*. https://www.cursosaula21.com/que-es-un-servomotor/
- *Complete 3D printer materials*. (s. f.). 3dsourced. https://www.3dsourced.com/guides/3d-printing-materials-cost/
- *Controlador PID - Control automático - Picuino*. (s. f.). https://www.picuino.com/es/control-pid.html
- *DC 2.5mm ID x 5.5mm OD Power Adaptor*. (s. f.). ServoCity®. https://www.servocity.com/dc-2-5mm-id-x-5-5mm-od-power-adaptor/
- De AdvancedMotionControls, P. (2024, 12 junio). *What is Servomechanism: Servo System Definition, History, Components & Applications*. ADVANCED Controles de Movimiento. https://www.a-m-c.com/es/servomecanismo/
- Dwamena, M. (2023, 13 noviembre). *How Much Does 3D Printing Cost? Cost to 3D Print Objects*. 3D Printerly. https://3dprinterly.com/how-much-does-3d-printing-cost/
- General Driver Motor. (2022, 26 abril). *HISTORIA y EVOLUCIÓN DEL SERVOMOTOR.* https://www.generaldrivermotor.com/noticias/historia-y-evolucion-del-servomotor/
- *I/O Extension Shield para Arduino Nano – Robótica Fácil*. (s. f.). https://roboticafacil.es/?product=arduino-nano-io-extension-shield
- Kang, C. (2016). Origin of Stability Analysis: \"On Governors\" by J.C. Maxwell [Historical Perspectives]. *IEEE Control Systems*, *36*(5), 77-88. https://doi.org/10.1109/mcs.2016.2584358
- Leopoldo Armesto. (2022a, noviembre 29). *Control de servos Dynamixel con la librería Dynamixel2Arduino* [Vídeo]. YouTube. https://www.youtube.com/watch?v=SBKjWDU0j6k

- Leopoldo Armesto. (2022b, diciembre 2). *Modelado y control de una articulación* [Vídeo]. YouTube. https://www.youtube.com/watch?v=dbw9dpQR2vg
- *Maxon 500 error page*. (s. f.). https://www.maxongroup.com/medias/sys_master/8803450421278.pdf
- *Micro Servo MG90S 2.5Kg*. (s. f.). Naylamp Mechatronics - Perú. https://naylampmechatronics.com/servomotores/246-micro-servo-mg90s.html
- Documentación placa U2D2 Power Hub (s. f.-a). *Introduction*. ROBOTIS e-Manual. https://emanual.robotis.com/docs/en/parts/interface/u2d2_power_hub/
- Documentación OpenRB-150 (s. f.-b). *Overview*. ROBOTIS e-Manual. https://emanual.robotis.com/docs/en/parts/controller/openrb-150/
- Documentación Servomotor XL330-M288-T (s. f.-c). *ROBOTIS e-Manual*. ROBOTIS e-Manual. https://emanual.robotis.com/docs/en/dxl/x/xl330-m288/
- *OpenRB-150*. (s. f.). ROBOTIS. https://en.robotis.com/shop_en/item.php?it_id=902-0183-000
- *OpenRB-150 Starter Kit*. (s. f.). ROBOTIS. https://en.robotis.com/shop_en/item.php?it_id=902-0184-000
- *Orígenes del PID*. (2009, 19 noviembre). Polo Estable. https://poloestable.wordpress.com/2009/11/02/origenes-del-pid/
- *Placa de unión de 4,9x2x10 mm*. (s. f.). Leroy Merlín. https://www.leroymerlin.es/productos/ferreteria-y-seguridad/tornillos-tacos-clavos-y-complementos/pletinas-escuadras-y-tornillos-de-ensamblaje/placa-de-union-de-4-9x2x10-mm-12632060.html
- *Que es pwm y como funciona*. (s. f.). https://www.shoptronica.com/curiosidades-tutoriales-y-gadgets/4517-que-es-pwm-y-como-funciona-0689593953254.html
- *Revista ElectroIndustria - Introducción histórica del Control Automático*. (s. f.). https://www.emb.cl/electroindustria/articulo.mvc?xid=474&ni=introduccion-historica-del-control-automatico

- *Servo MG90S – robótica fácil*. (s. f.). https://roboticafacil.es/?product=servo-mg90s
- *Servo Mini SG90 9G Micro 360 motor paso a paso giro continuo*. (s. f.). ElectroComponentes.es. https://www.electrocomponentes.es/motores-y-servos/931-servo-mini-sg90-9g-micro-360-motor-paso-a-paso-giro-continuo.html
- *Servomotores: un breve repaso por su historia | Secoin*. (s. f.). https://www.secoin.com.uy/blog/servomotores-un-breve-repaso-por-su-historia
- *Tornillos M2.6*12 mm*. (s. f.). Amazon. https://www.amazon.es/dp/B07VPC2SYJ?psc=1&ref=ppx_yo2ov_dt_b_product_details
- *U2D2 PHB set*. (s. f.). ROBOTIS. https://en.robotis.com/shop_en/item.php?it_id=902-0145-001
- *Dynamixel2Arduino - Arduino reference*. (s. f.). https://www.arduino.cc/reference/en/libraries/dynamixel2arduino/

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.

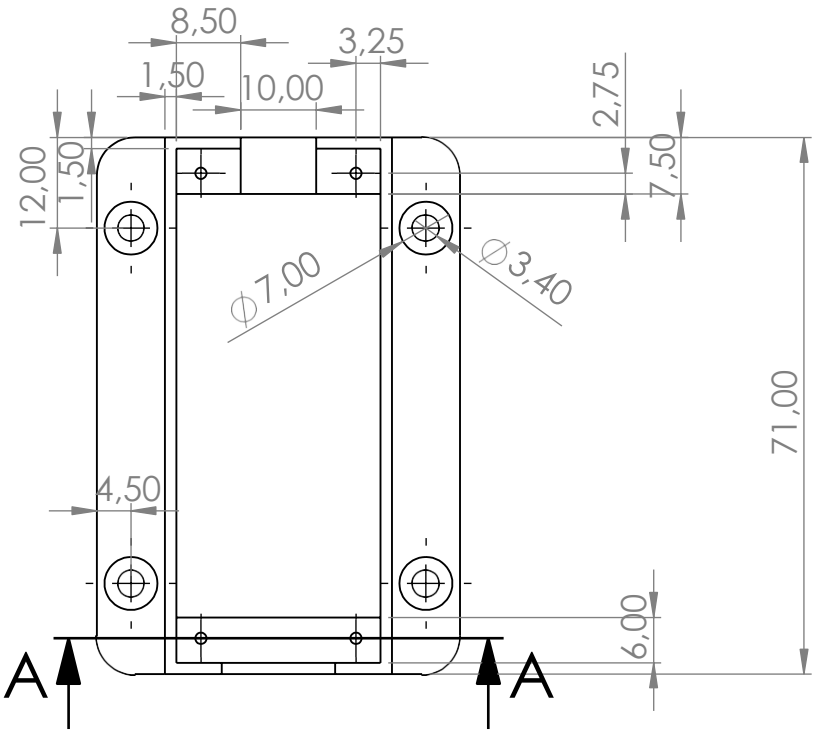
Trabajo Fin de Grado
Grado en Ingeniería Electrónica Industrial y Automática

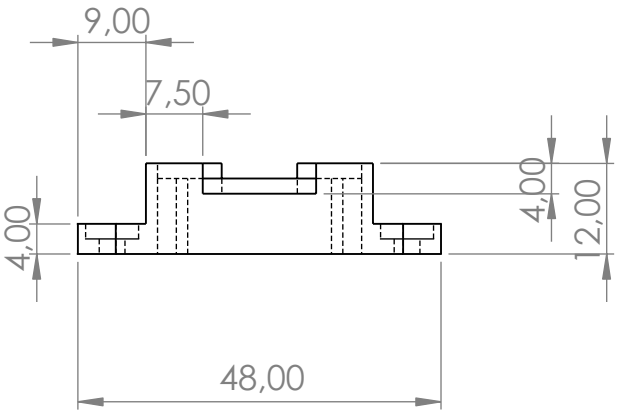
Documento 2:

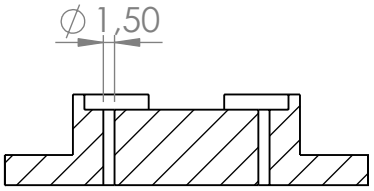
Planos

Autor: José Gutiérrez Jiménez
Tutor: Ranko Zotovic Stanisic

Curso académico 2023-2024



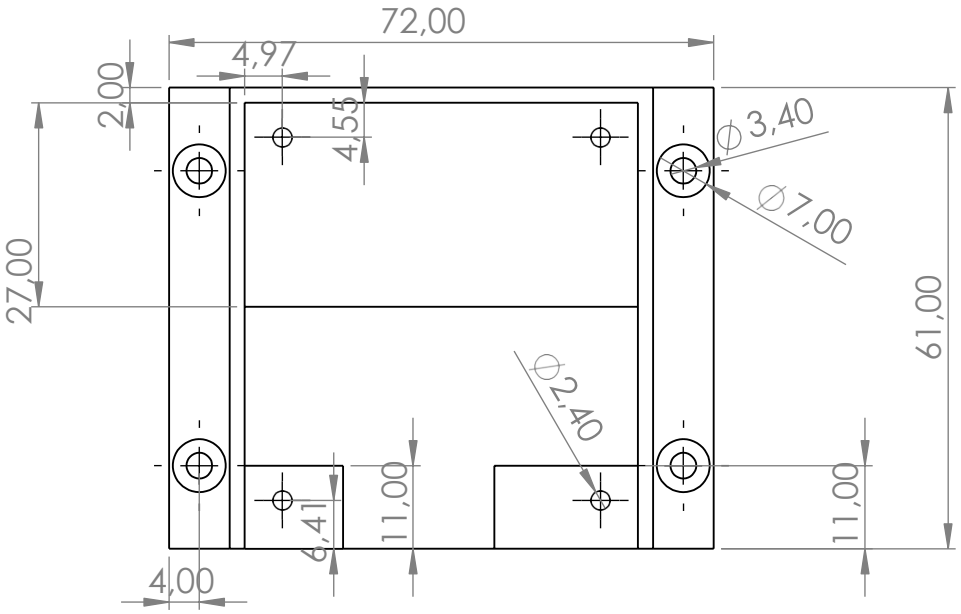


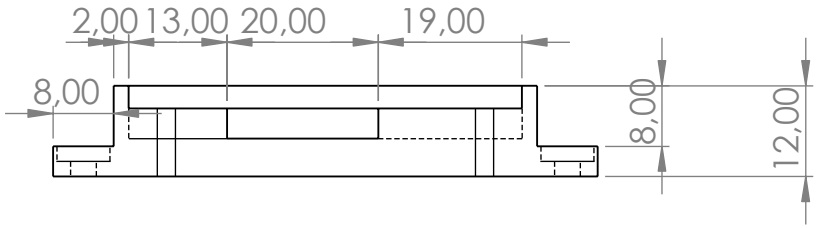


SECCIÓN A-A

ESCALA 1 : 1

FECHA: **10/06/2024**	DIBUJADO POR: **José Gutiérrez Jiménez**	**UNIVERSIDAD POLITÉCNICA DE VALENCIA**	
UNIDADES: **mm**	TÍTULO DEL PROYECTO: **Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.**	**Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial**	
ESCALA: **1:1**	DENOMINACIÓN DEL PLANO: **Base OPENRB-150**	N.º DE PLANO **1**	MATERIAL: **HIPS** / A4 / HOJA 1 DE 1

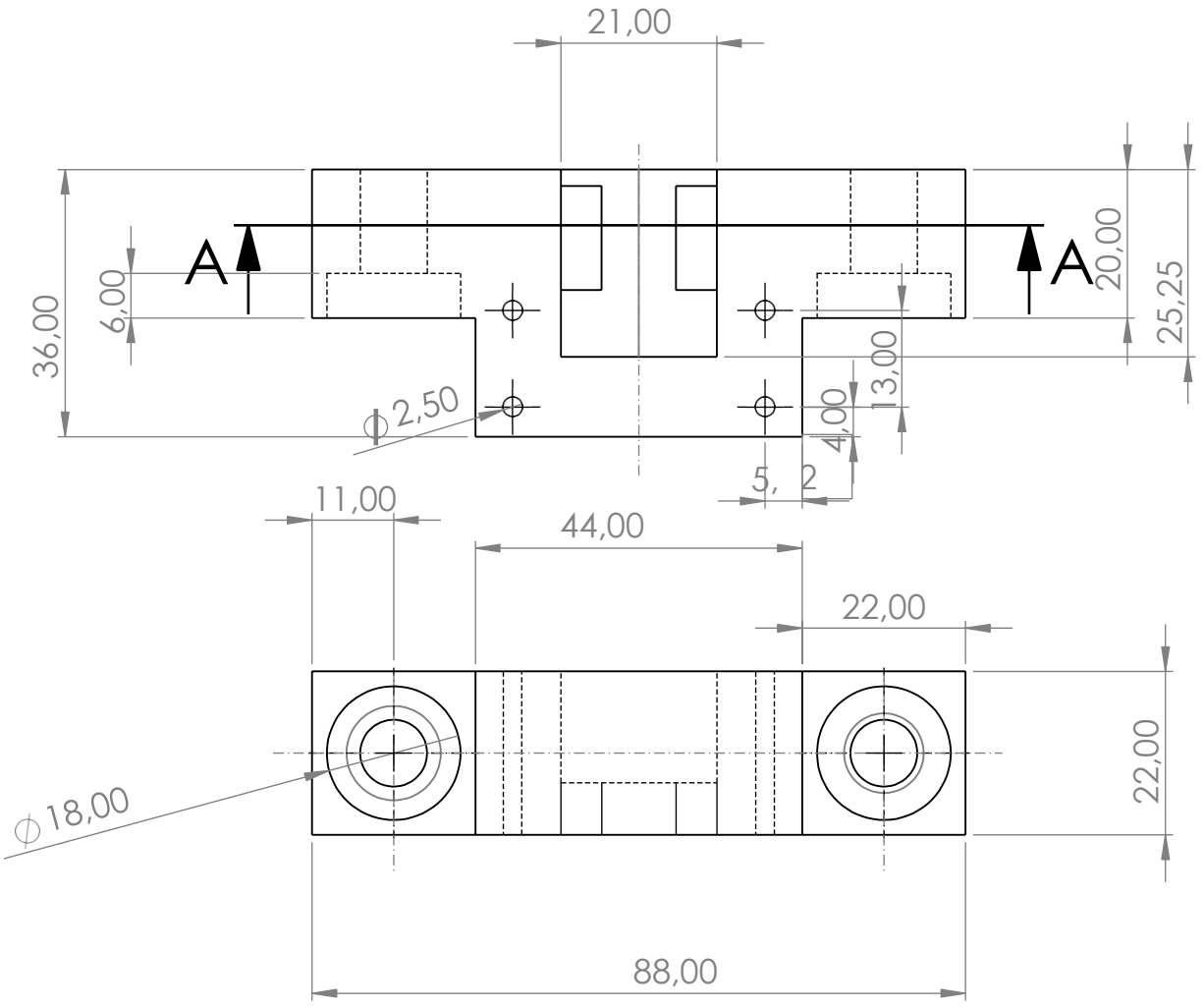




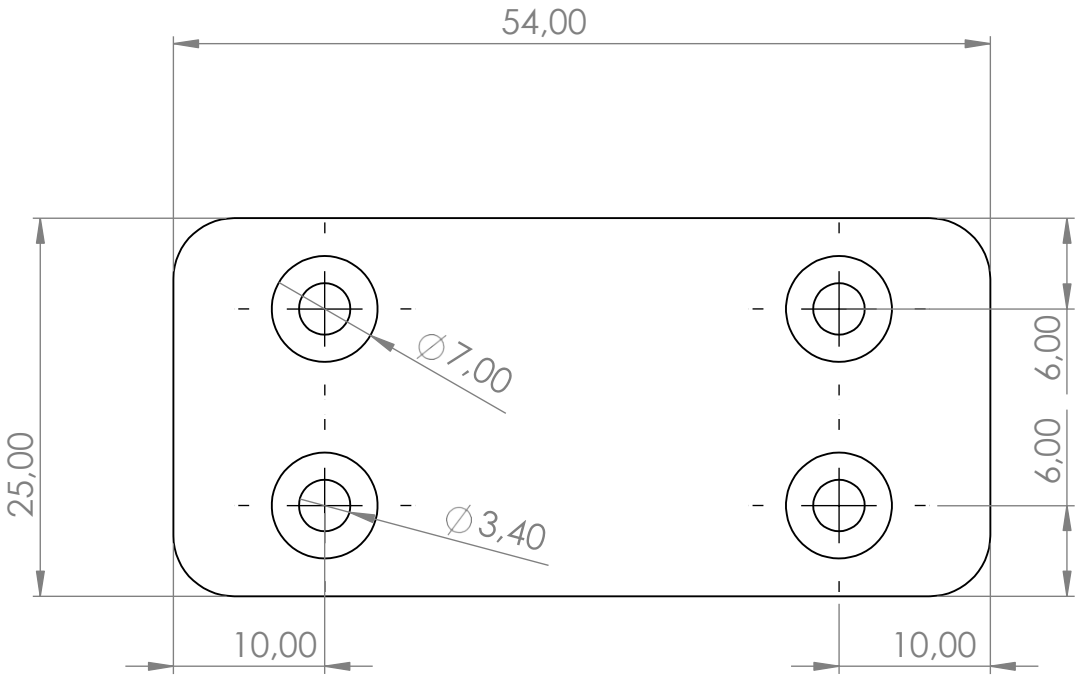
FECHA: **10/06/2024**	DIBUJADO POR: **José Gutiérrez Jiménez**	**UNIVERSIDAD POLITÉCNICA DE VALENCIA**	
UNIDADES: **mm**	TÍTULO DEL PROYECTO: **Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.**	**Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial**	
ESCALA: **1:1**	DENOMINACIÓN DEL PLANO: **Base de U2D2 Power Hub**	N.º DE PLANO **2**	MATERIAL: **HIPS** A4 HOJA 1 DE 1

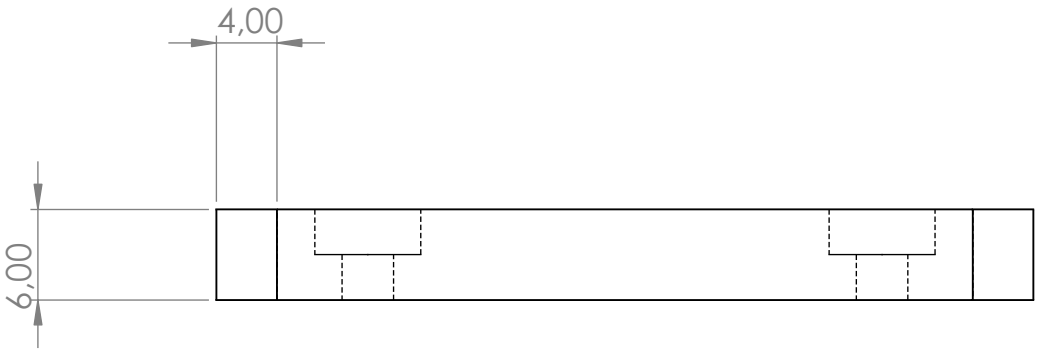
SECCIÓN A-A

ESCALA 1 : 1

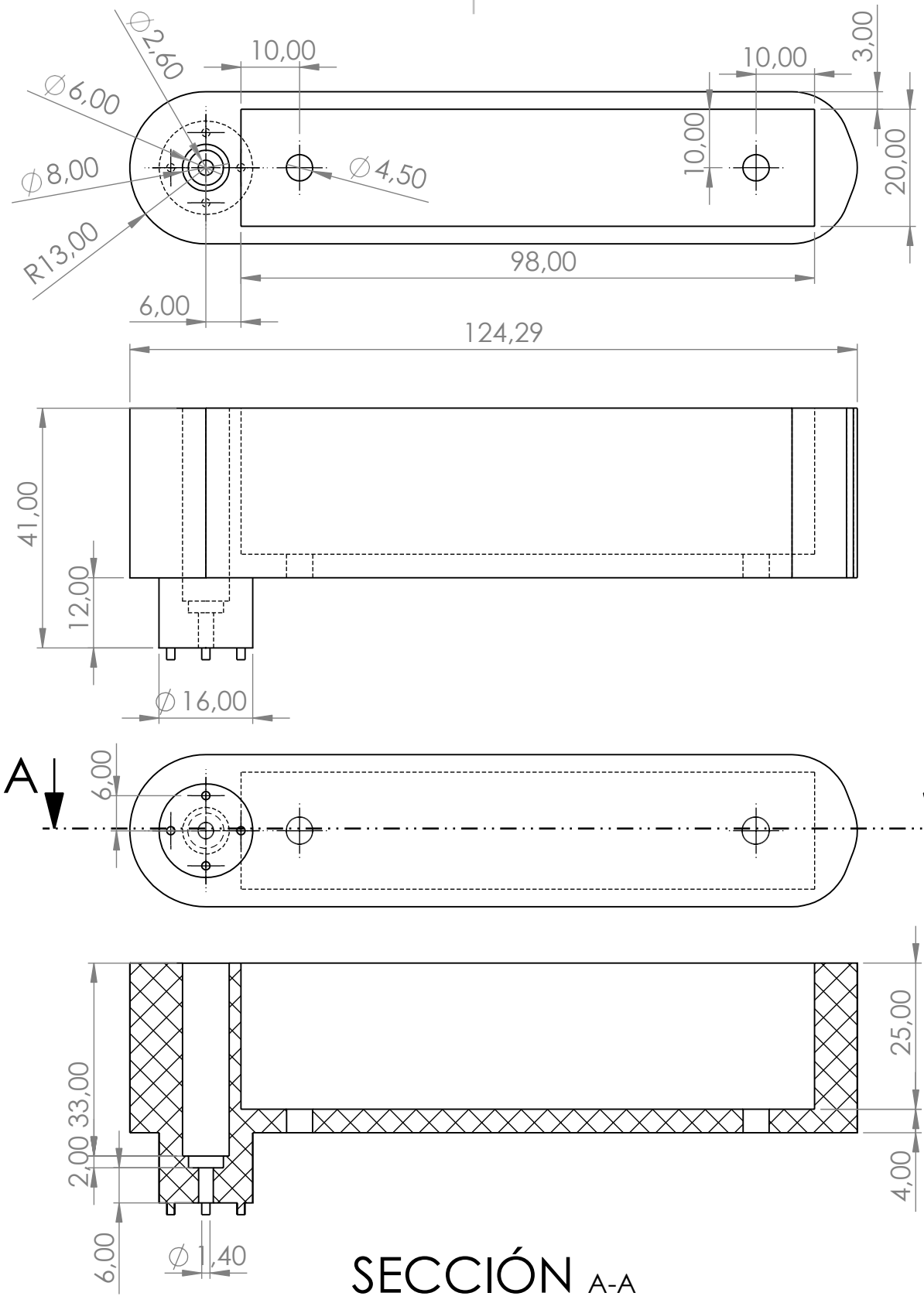


FECHA: **10/06/2024**	DIBUJADO POR: **José Gutiérrez Jiménez**	**UNIVERSIDAD POLITÉCNICA DE VALENCIA**	
UNIDADES: **mm**	TÍTULO DEL PROYECTO: **Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.**	**Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial**	
ESCALA: **1:1**	DENOMINACIÓN DEL PLANO: **Base del Servo**	N.º DE PLANO **3**	MATERIAL: **HIPS** / A4 / HOJA 1 DE 1





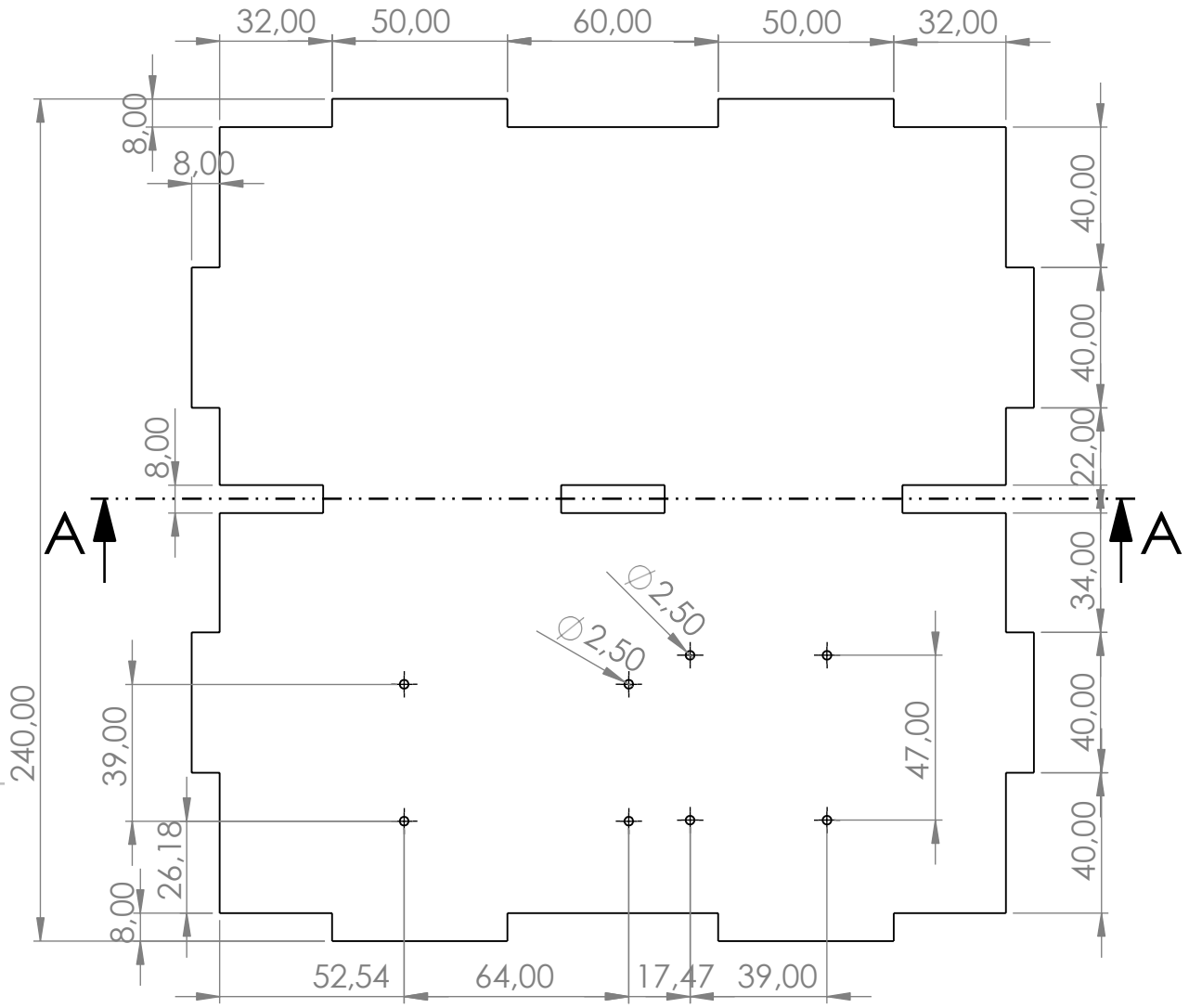
FECHA: **10/06/2024**	DIBUJADO POR: **José Gutiérrez Jiménez**	**UNIVERSIDAD POLITÉCNICA DE VALENCIA**	
UNIDADES: **mm**	TÍTULO DEL PROYECTO: **Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.**	**Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial**	
ESCALA: **2:1**	DENOMINACIÓN DEL PLANO: **Asegurador del Servo**	N.º DE PLANO **4**	MATERIAL: **ABS** A4 HOJA 1 DE 1

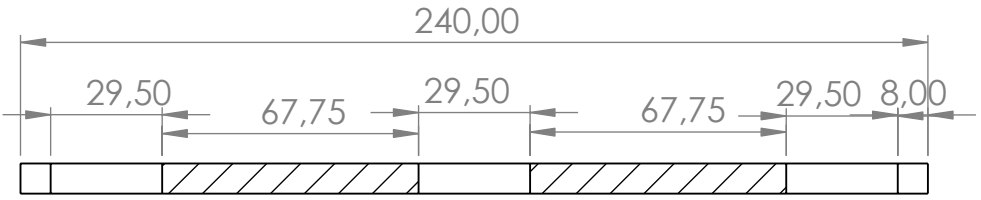


SECCIÓN A-A

ESCALA 1 : 1

FECHA: **10/06/2024**	DIBUJADO POR: **José Gutiérrez Jiménez**	**UNIVERSIDAD POLITÉCNICA DE VALENCIA**		
UNIDADES: **mm**	TÍTULO DEL PROYECTO: **Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.**	**Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial**		
ESCALA: **1:1**	DENOMINACIÓN DEL PLANO: **Eslabón/Articulación**	N.º DE PLANO **5**	MATERIAL: **POLVO DE NILON**	A4 / HOJA 1 DE 1

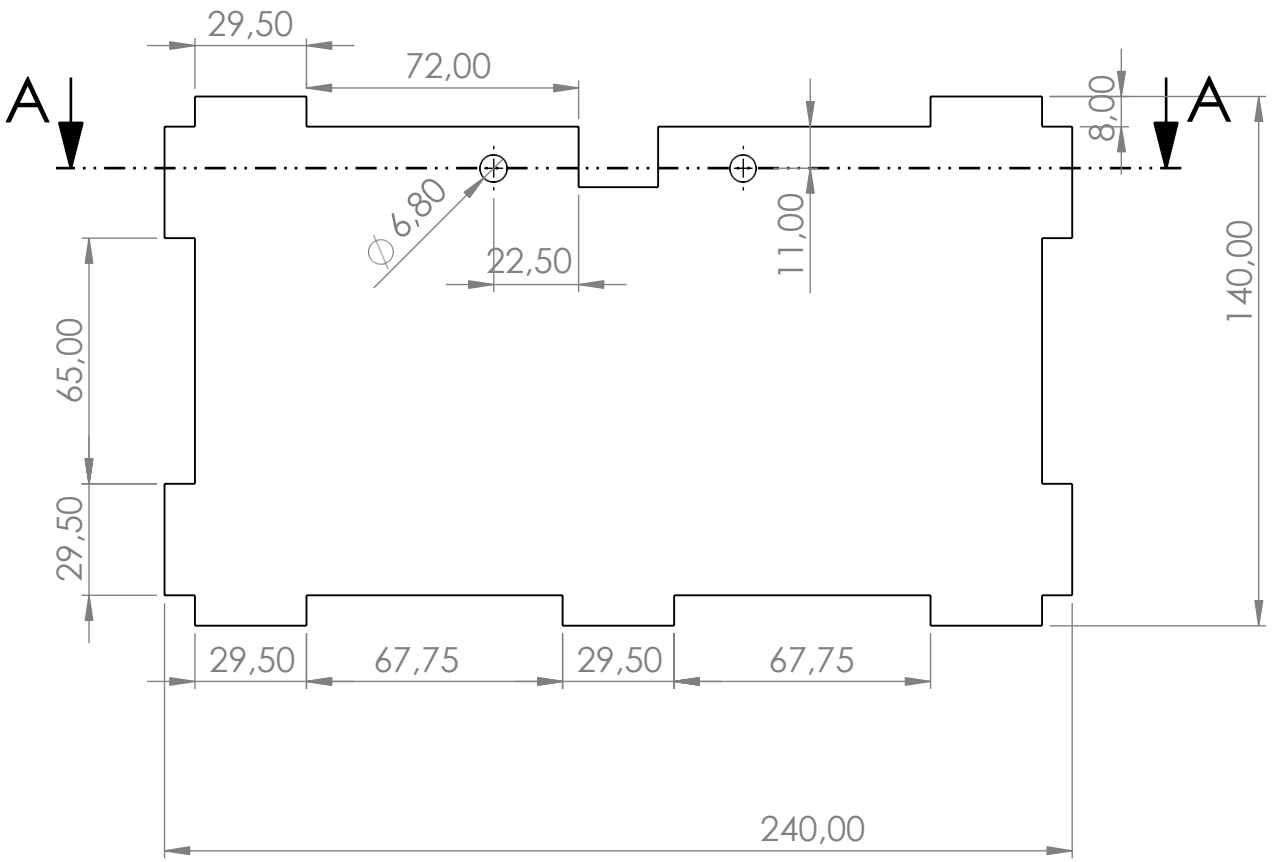


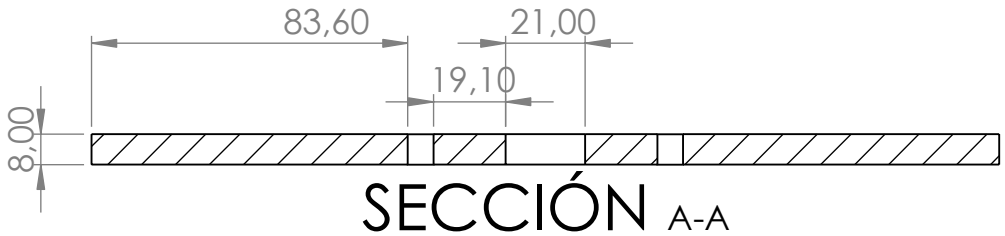


SECCIÓN A-A

ESCALA 1 : 2

FECHA: **10/06/2024**	DIBUJADO POR: **José Gutiérrez Jiménez**	**UNIVERSIDAD POLITÉCNICA DE VALENCIA**	
UNIDADES: **mm**	TÍTULO DEL PROYECTO: **Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.**	**Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial**	
ESCALA: **1:2**	DENOMINACIÓN DEL PLANO: **Base delprototipo**	N.º DE PLANO **6**	MATERIAL: **METACRILATO** A4 HOJA 1 DE 1

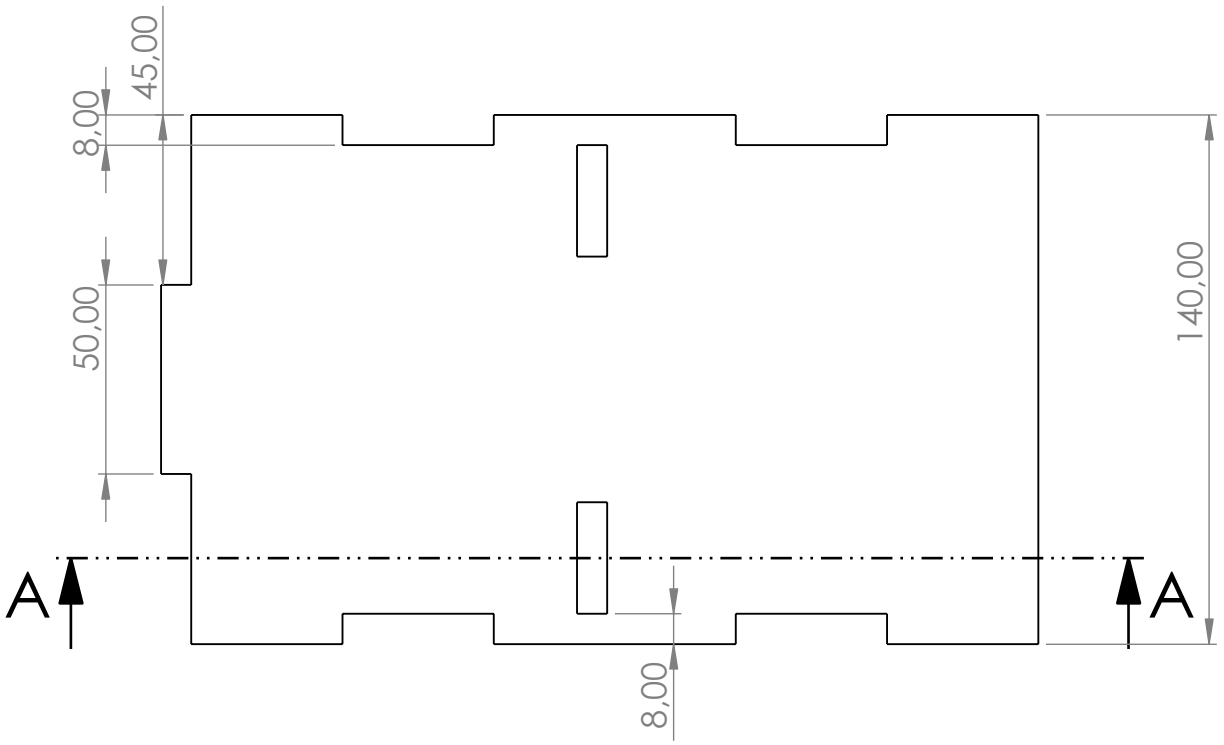


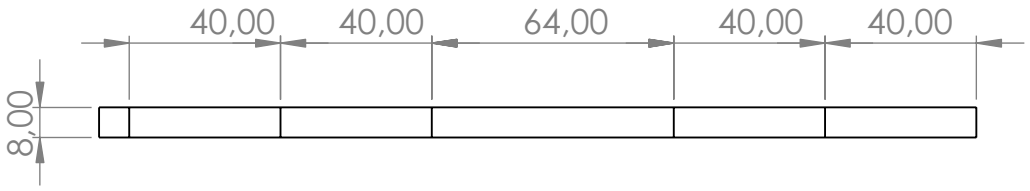


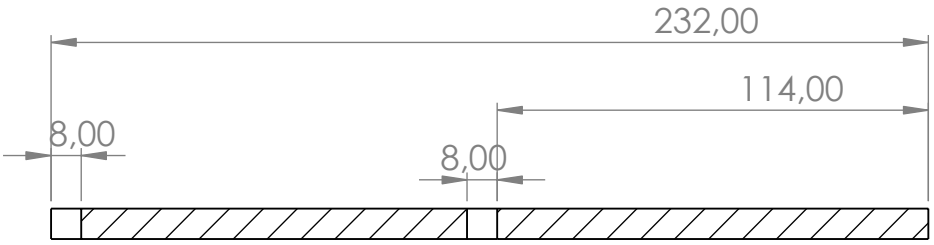
SECCIÓN A-A

ESCALA 1 : 2

FECHA: **10/06/2024**	DIBUJADO POR: **José Gutiérrez Jiménez**	**UNIVERSIDAD POLITÉCNICA DE VALENCIA**	
UNIDADES: **mm**	TÍTULO DEL PROYECTO: **Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.**	**Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial**	
ESCALA: **1:2**	DENOMINACIÓN DEL PLANO: **Pared de Carga**	N.º DE PLANO **7**	MATERIAL: **METACRILATO** A4 HOJA 1 DE 1



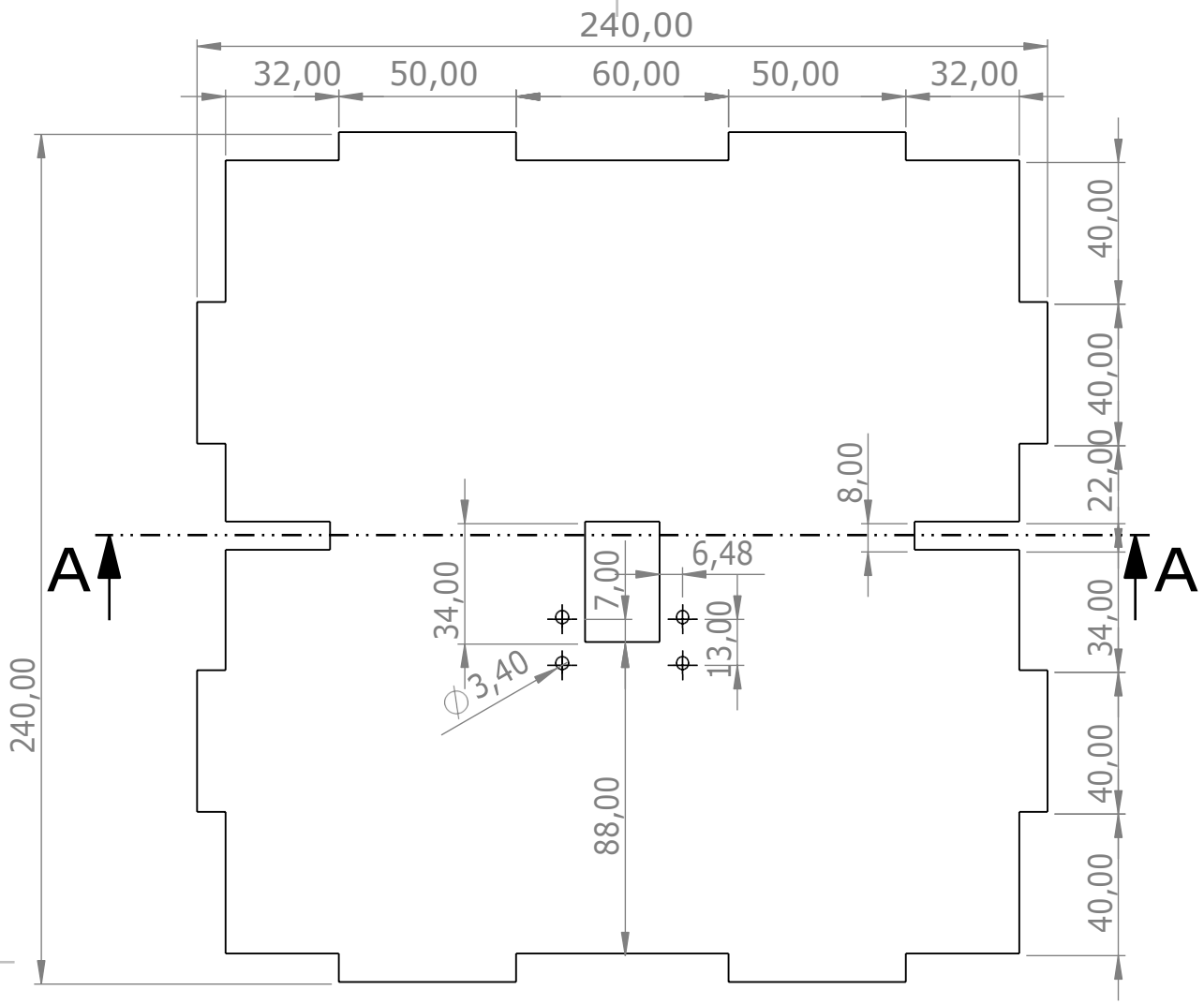


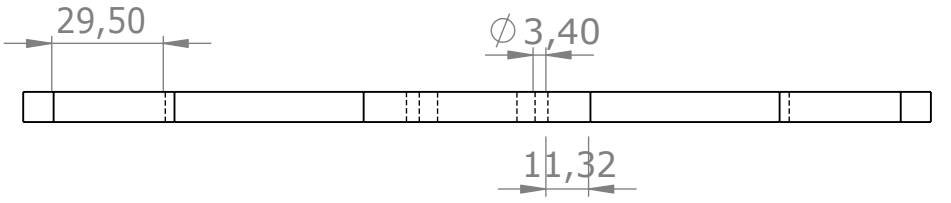


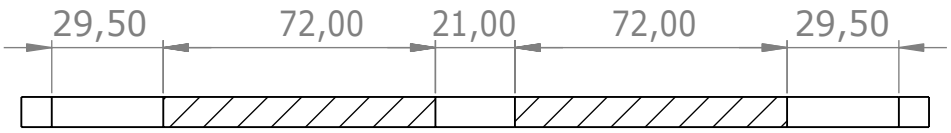
SECCIÓN A-A

ESCALA 1 : 2

FECHA: **10/06/2024**	DIBUJADO POR: **José Gutiérrez Jiménez**	**UNIVERSIDAD POLITÉCNICA DE VALENCIA**	
UNIDADES: **mm**	TÍTULO DEL PROYECTO: **Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.**	**Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial**	
ESCALA: **1:2**	DENOMINACIÓN DEL PLANO: **Pared Izquierda/Derecha**	N.º DE PLANO **8**	MATERIAL: **METACRILATO** / A4 / HOJA 1 DE 1



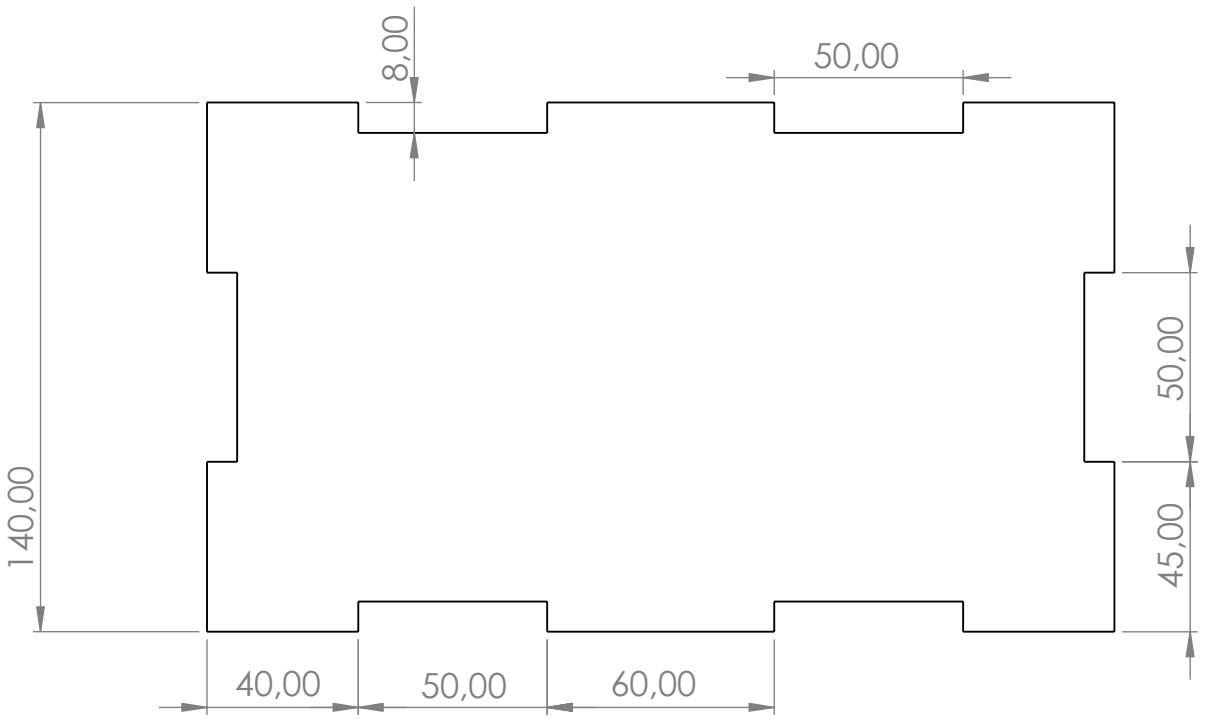


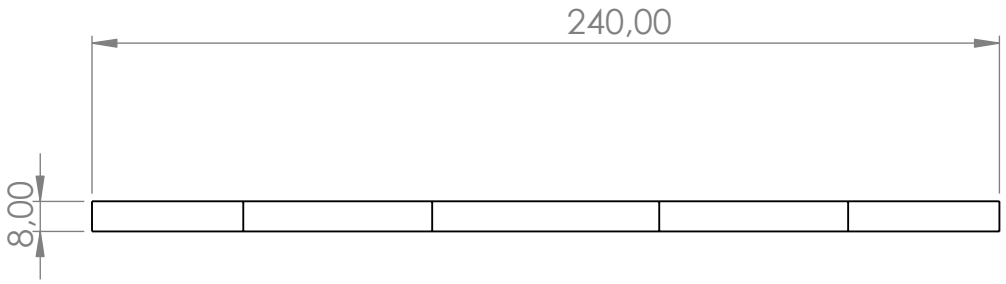


SECCIÓN A-A

ESCALA 1 : 2

FECHA: **10/06/2024**	DIBUJADO POR: **José Gutiérrez Jiménez**	**UNIVERSIDAD POLITÉCNICA DE VALENCIA**	
UNIDADES: **mm**	TITULO DEL PROYECTO: **Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.**	**Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial**	
ESCALA: **1:2**	DENOMINACIÓN DEL PLANO: **Tapa**	N.º DE PLANO **9**	MATERIAL: **METACRILATO** / A4 / HOJA 1 DE 1





FECHA: **10/06/2024**	DIBUJADO POR: **José Gutiérrez Jiménez**	**UNIVERSIDAD POLITÉCNICA DE VALENCIA**	
UNIDADES: **mm**	TÍTULO DEL PROYECTO: **Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.**	**Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial**	
ESCALA: **1:2**	DENOMINACIÓN DEL PLANO: **Pared Fondo**	N.º DE PLANO **10**	MATERIAL: **METACRILATO** A4 HOJA 1 DE 1

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.

Trabajo Fin de Grado
Grado en Ingeniería Electrónica Industrial y Automática

Documento 3:

Pliego de condiciones

Autor: José Gutiérrez Jiménez
Tutor: Ranko Zotovic Stanisic

Curso académico 2023-2024

ÍNDICE

1 OBJETO

El actual pliego de condiciones técnicas reúne el conjunto de especificaciones, condiciones y criterios que establecen todos los requisitos técnicos del proyecto “Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico”.

Asimismo, el pliego incluye las instrucciones para el montaje de la herramienta en el apartado 3. Condiciones de la ejecución.

2 CONDICIONES DE LOS MATERIALES

2.1 Descripción

Los materiales utilizados para realizar este proyecto son:

Material	Cantidad
Servomotor XL-330-M88-T	1
Placa OPENRB-150	1
Placa U2D2 Power Hub	1
Fuente de alimentación 5 V/2A	1
Cable de alimentación	1
Adaptador del cable de alimentación	1
Cable de comunicación USB Tipo C	1
Cables de comunicación TTL	2
Pieza "Base del Servomotor"	1
Pieza "Base del U2D2 Power Hub"	1
Pieza "Base de OPENRB-150"	1
Pieza "Asegurador de Servomotor"	1
Pieza "Base"	1
Pieza "Pared Izquierda/derecha"	2
Pieza "Pared Carga"	1
Pieza "Tapa"	1
Pieza "Pared Fondo"	1
Placas metálicas Leroy Merlín	5
Tornillos punta M2 x 6 mm	4
Tornillos punta M2,6 x 12 mm	1
Tornillos punta M3 x 6 mm	4
Tornillos rosca M3 x 10 mm	8
Tornillos punta M3 x 18 mm	4
Tornillos rosca M3 x 20 mm	2
Tornillos rosca M6 x 18 mm	2
Tuercas M3	4
Tuercas M6	2

Tabla 1. Materiales del prototipo

En caso de necesitar un sustituto o reemplazo para alguno de los materiales hay que primero cerciorarse de cuál es el objetivo y función de ese material, y una vez entendido buscar un sustituto que cumpla esas mismas funciones. Un ejemplo puede ser la fuente de alimentación

que mientras que su salida sea de 5 V y 2 A el sistema seguirá funcionando sin importar que marca sea. Otro ejemplo puede ser la placa U2D2 Power Hub que puede ser sustituida por otra etapa de potencia que tenga las mismas características y puertos que la misma.

2.2 Control de calidad

El control de calidad generalizado de los materiales es una inspección visual y cerciorarse si algún elemento tiene indicaciones de haber sido manipulado indebidamente o un defecto de fábrica, en su caso buscar un reemplazo de ese material o buscar el mismo elemento sin ese error.

Respecto a los componentes electrónicos se ha de comprobar que todas sus entradas y salidas funcionan correctamente y no hay ningún corte en sus conexiones.

En el caso del cableado es necesario primero comprobar su funcionamiento y en caso afirmativo manipular el cable para que no tenga el defecto común que según la posición del cable deja de estar conectado.

Por último, hay que comprobar que las piezas impresas están bien lijadas y no hay ninguna suciedad de la impresión. Siguiendo, hay que comparar los modelos diseñados con las piezas reales para comprobar si ha habido algún error durante el proceso de impresión, en caso de que haya un error se tiene que valorar si ese error impide el funcionamiento normal de la pieza. Como último paso, se ha de proceder a un previo montaje en el que se pruebe que cada pieza encaja en su sitio predeterminado, un caso sería comprobar que en las piezas de metacrilato encajan los agujeros y salientes unos con otras, otro caso sería comprobar que los salientes del eslabón encajan en la articulación.

3 CONDICIONES DE LA EJECUCIÓN

3.1 Descripción

Para seguir adecuadamente el montaje del dispositivo es necesario tener todos los materiales y piezas listados en el apartado 2.1 de este documento de pliego de condiciones. Además, es necesario tener:

- Destornillador de precisión, en el caso del proyecto de punta de estrella de 3 mm.
- Ordenador con Arduino IDE 1.8.19 o superior instalado.

3.1.1 Montaje del dispositivo

Se recomienda un espacio amplio y libre de objetos que pueden causar dificultades durante el montaje de las piezas, además de un lugar separado para acceder a las otras piezas o tornillería. En el caso del prototipo se ha usado una mesa vacía con un trozo de tela para realizar el montaje. Los pasos en orden sería:

[1] Localizar la placa OPENRB-150.
[2] Cambiar el puente al modo de alimentación externa (VIN(DXL)), es decir, que el puente de alimentación esté colocado como se muestra en la imagen:

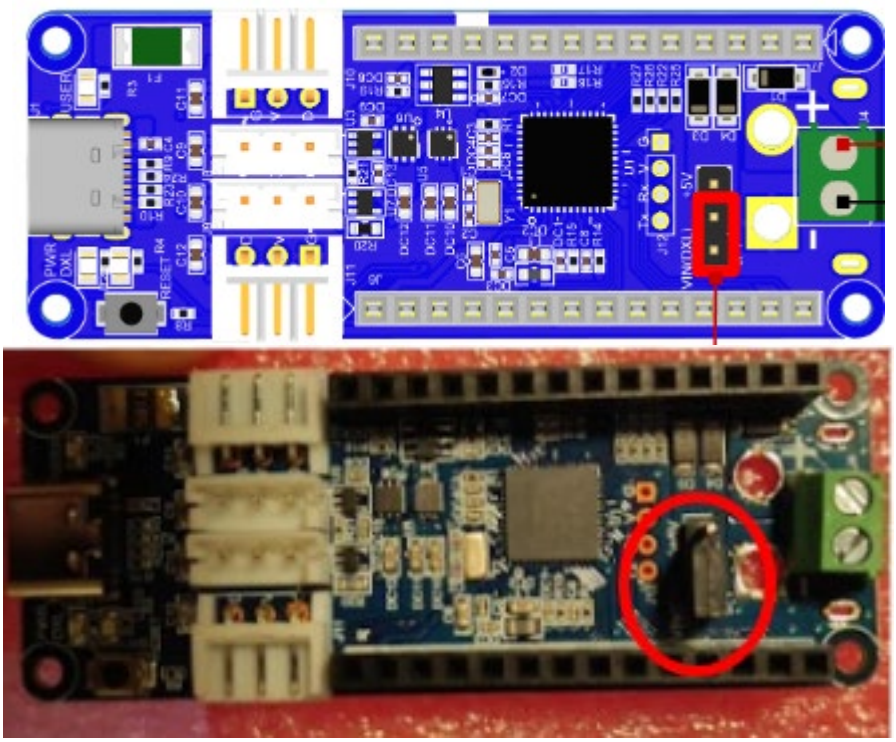


Imagen 1. Puente de alimentación externa placa OPENRB-150

[3] Localizar la pieza “Base del OPENRB-150”.

Imagen 2. Pieza “Base del OPENRB-150”.

[4] Depositar la placa OPENRB-150 en la pieza observando que encaja con algo de soltura, además de observar que el puerto Tipo C queda en la dirección con la apertura más amplia horizontalmente, como en la imagen.

Imagen 3. Resultado paso [4]

[5] Atornillar la placa con 4 tornillos acabados en punta de M2 x 6 mm. Estos han de introducirse en los cuatro orificios en las esquinas de la placa y se ha alinear con los agujeros de la pieza 3D.

[6] Localizar la pieza “Base del U2D2 Power Hub”.

Imagen 4. Pieza "Base del U2D2 Power hub"

[7] Depositar la placa U2D2 Power Hub observando que la entrada del cable de alimentación queda en la única dirección que no tiene ninguna pared/saliente.

Imagen 5. Resultado paso [8]

[8] Atornillas la placa con 4 tornillos en punta M3 * 6 mm usando los agujeros en las 4 esquinas de la placa y alienándolos con los agujeros de la pieza.

[9] Localizar la pieza “Base”.

Imagen 6. Pieza "Base"

[10] Alinear las bases de las placas ya montadas con la pieza de metacrilato observando que coincidan los agujeros de la pieza con los de las bases y asegurándose que los puertos de la placa están orientados hacia el exterior de la

pieza base de metacrilato.

[11] Una vez alineados, atornillar todas las piezas utilizando 8 tornillos de rosca de M3 *10 mm.

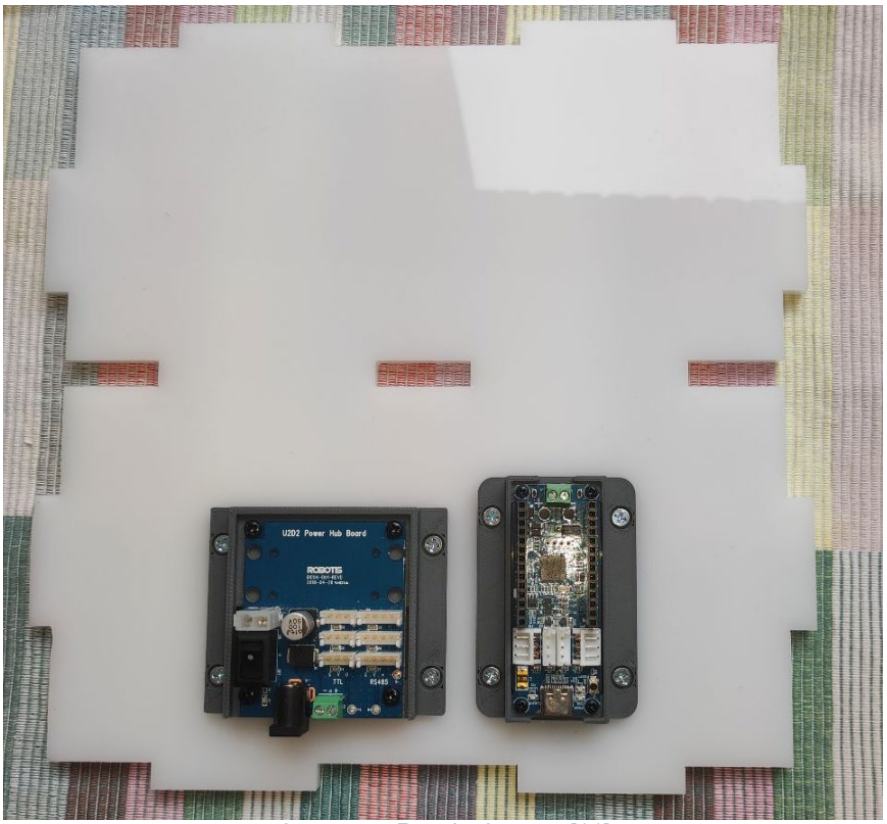


Imagen 7. Resultado paso [11]

[12] Localizar el cable más corto TTL que se tenga y conectar el puerto más alejado a la izquierda de la placa OPENRB-150 con el puerto en la esquina izquierda arriba de la placa Power Hub.

Imagen 8. Resultado paso [12]

[13] Localizar la pieza “Pared de carga”

Imagen 9. Pieza "Pared de carga"

[14] Localizar las dos piezas de “Pared izquierda/derecha”

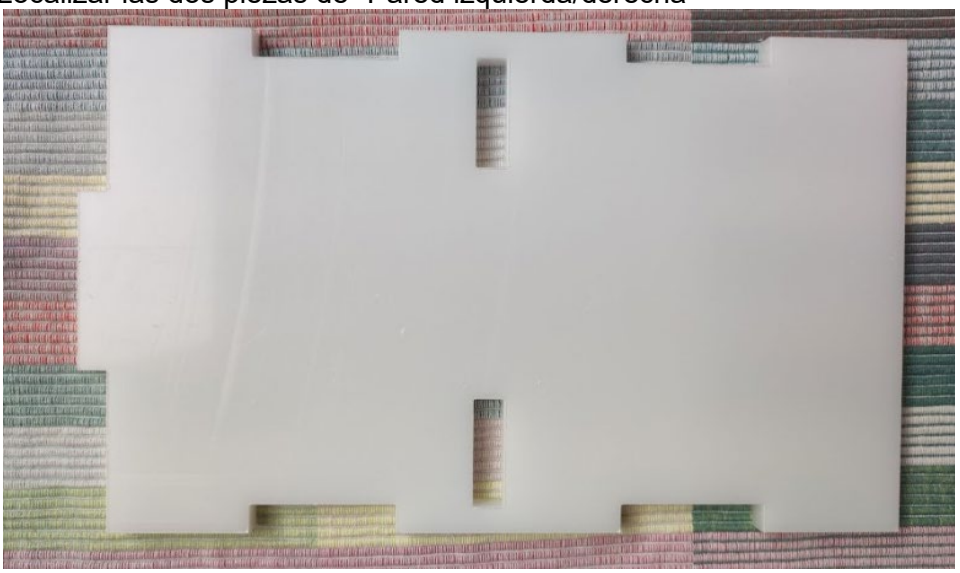


Imagen 10. Piezas "Pared izquierda/derecha"

[15] Levantar la pared de carga y encajar las dos piezas idénticas una a cada lado de la pared de carga, teniendo en cuenta que el lado de las piezas que no tiene ningún saliente ha de quedar orientados la misma dirección. Los salientes de la pared de carga han de encajar en los agujeros que se encuentran en medio de las otras piezas.

Imagen 11. Resultado paso [15]

[16] Juntar las tres piezas encajadas con la base con las placas. Hay que tener precaución de no dañar ninguna placa en el proceso y asegurar que los salientes están en dirección contraria a las placas.

[17] Localizar la pieza “Base del Servomotor”

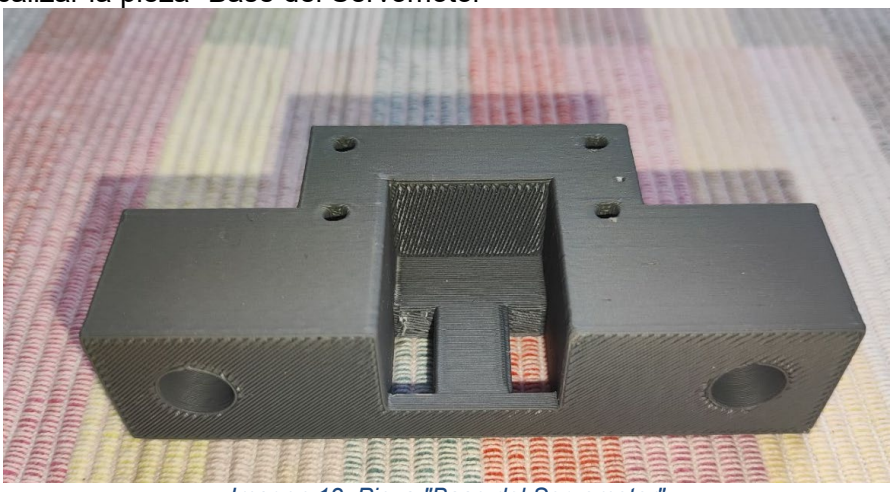


Imagen 12. Pieza "Base del Servomotor"

[18] Introducir dos tornillos con rosca de M6 x 30 mm e introducirlos en la pieza

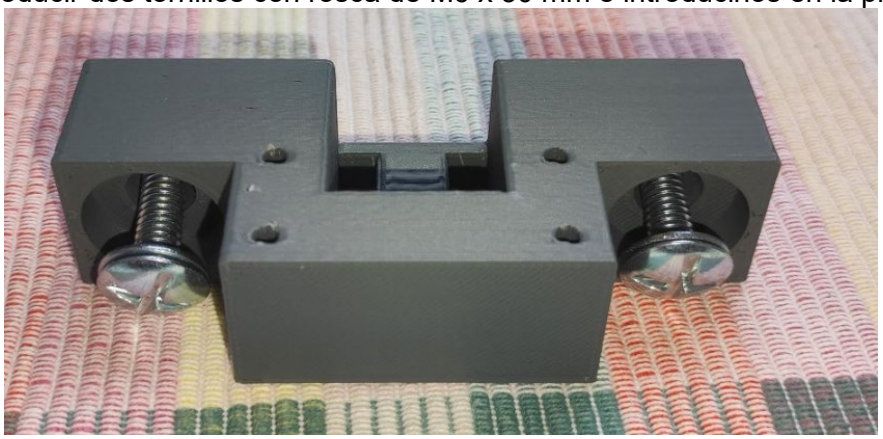


Imagen 13. Resultado paso [18]

[19] Alinear la pieza con la pared de carga. Seguir la referencia del rectángulo en medio de la pieza. Una vez pasado los tornillos a través de la pared de carga introducir una tuerca en cada para mantener la pieza pero dejándola algo suelta.

Imagen 14. Resultado paso [19]

[20] Localizar la pieza “Tapa”.

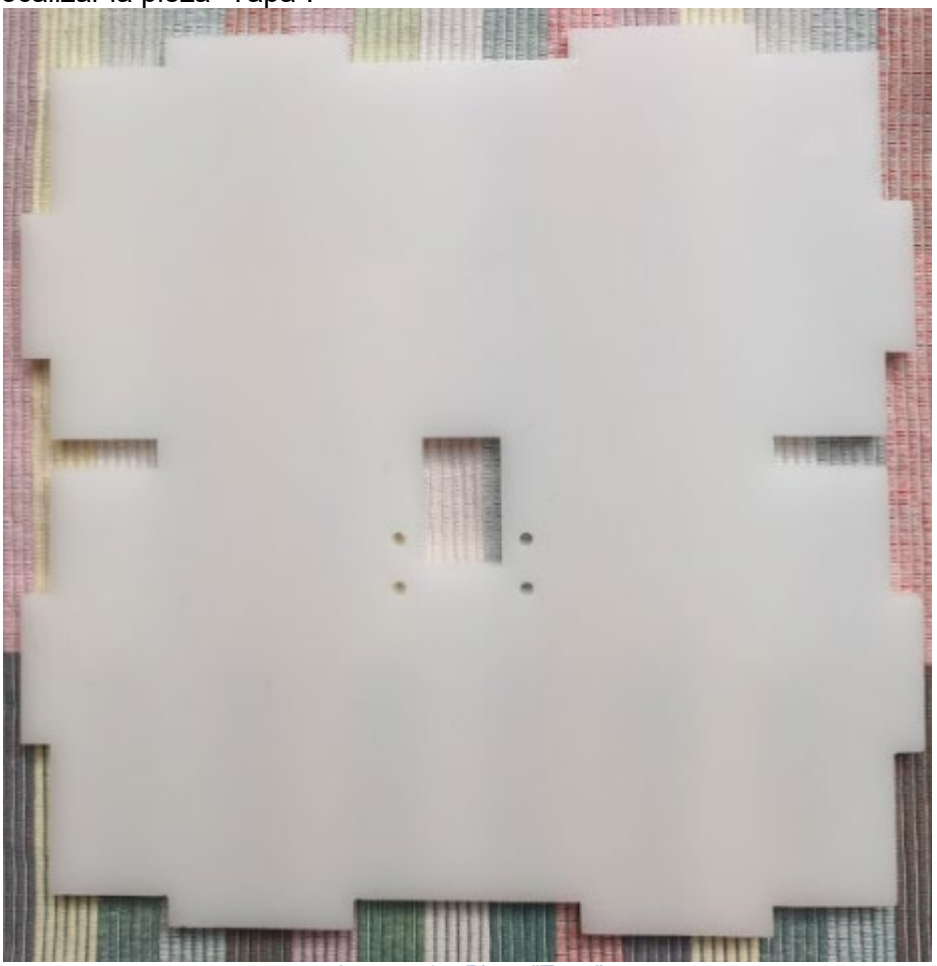


Imagen 15. Pieza "Tapa"

[21] Introducir la tapa sobre el montaje realizado hasta ahora. Los salientes de los lados deberían encajar perfectamente con los agujeros de las otras piezas. Especial atención en que los agujeros de la tapa queden orientados en dirección de las placas y la base del servomotor.

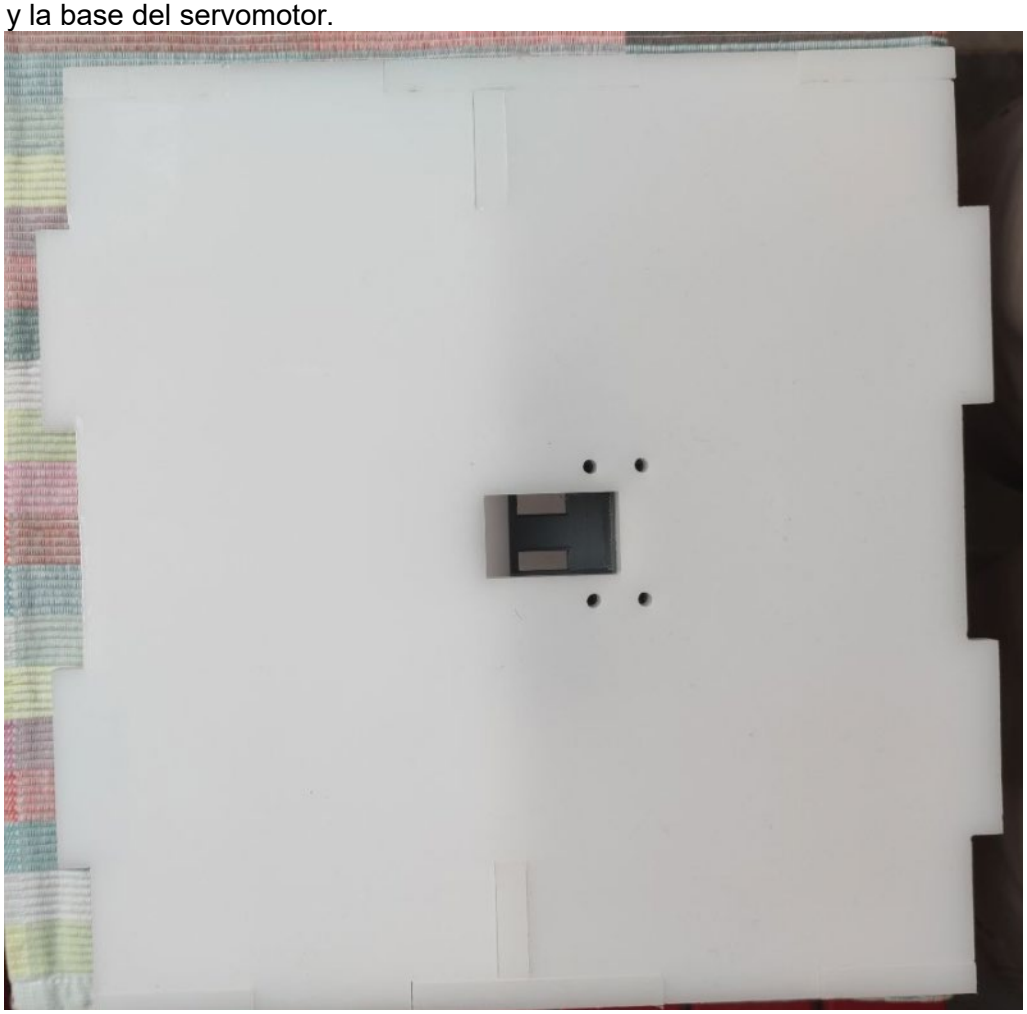


Imagen 16. Resultado paso [21]

[22] Conectar el cable TTL más largo que se tenga al servomotor e introducir la parte desconectada por uno de los agujeros cuadrados de la pieza “base del servomotor”. Seguidamente introducir el servo por el agujero rectangular de la tapa y terminar de ajustar las tuercas de la base. Si hay mucha resistencia es posible que la base esté demasiado apretada. La rosca del servomotor debería quedar por encima de la tapa.

Imagen 17. Resultado paso [22]

[23] Conectar el lado desconectado del cable TTL al puerto inferior ya conectado de la placa Power Hub.

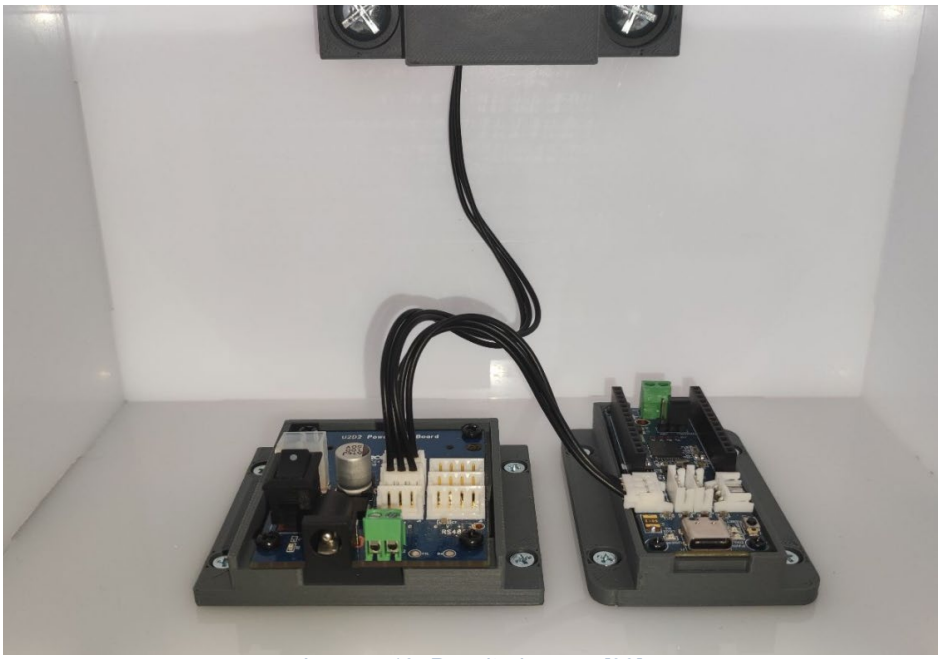


Imagen 18. Resultado paso [23]

[24] Localizar la pieza “Asegurador del Servomotor”.

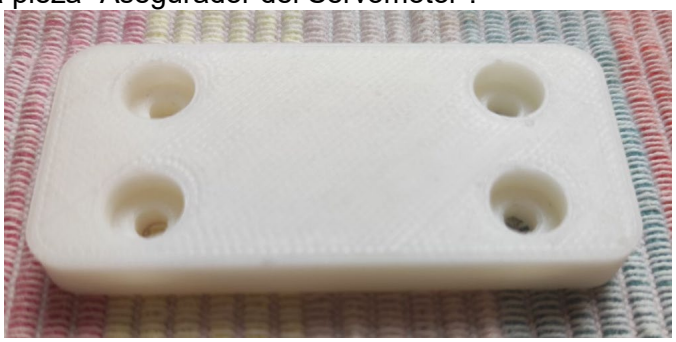


Imagen 19. Pieza "Asegurador Servo"

[25] Alinearlo con los agujeros de la tapa del montaje e introducir 4 tornillos M3 x 18 mm. Estos tornillos deberían introducirse en los agujeros de la pieza “Base del Servomotor”

Imagen 20. Resultado paso [25]

[26] Localizar la pieza “Eslabón/Articulación”

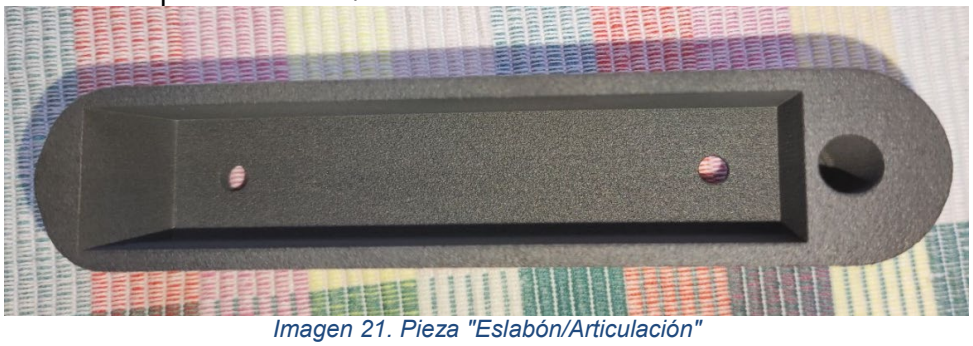


Imagen 21. Pieza "Eslabón/Articulación"

[27] Introducir con un destornillador el tornillo de 2.6 x 12 mm por el agujero que se encuentra más a la derecha de la imagen anterior (Imagen 21). Es posible tener que rotar el tornillo como si se estuviese atornillando para que sobresalir.

Imagen 22. Resultado paso [27]

[28] Atornillar el eslabón al servomotor observando que se introduzcan todos los salientes en los agujeros de la rueda del servo. Ha de quedar alineado en sentido contrario a las placas y al asegurador del servomotor.

Imagen 23. Resultado paso [28]

[29] En el caso del prototipo se ha usado celo al final del montaje para asegurar todas las piezas y ha resultado bastante efectivo.

[30] Los siguientes pasos son opcionales, ya que al introducir la pieza pared del fondo se pierde el acceso al led del servomotor, así que se puede colocar o no.

[31] Opcional: Localizar la pieza “Pared Fondo”.

Imagen 24. Pieza "Pared Fondo"

[32] Opcional: Colocarla en el lado contrario de las placas electrónicas para cerrar la estructura

Imagen 25. Resultado paso [32]

3.1.2 Programación de la placa

Una vez acabada la parte física del montaje, pasamos a la programación de la placa. Para introducir la programación en la placa OPENRB-150 primero es necesario estar en posesión del archivo final con el código de este proyecto (“CodigoTFG_Final.ino”). Conseguido ese código, es necesario abrirlo en la aplicación de Arduino IDE. Si no se tiene instalada la librería de “Dynamixel2Arduino” hará falta instalarla siguiendo la ruta de Programa -> Incluir Librería -> Administrar Bibliotecas, buscando en la barra “dynamixel2arduino” y pulsando instalar.

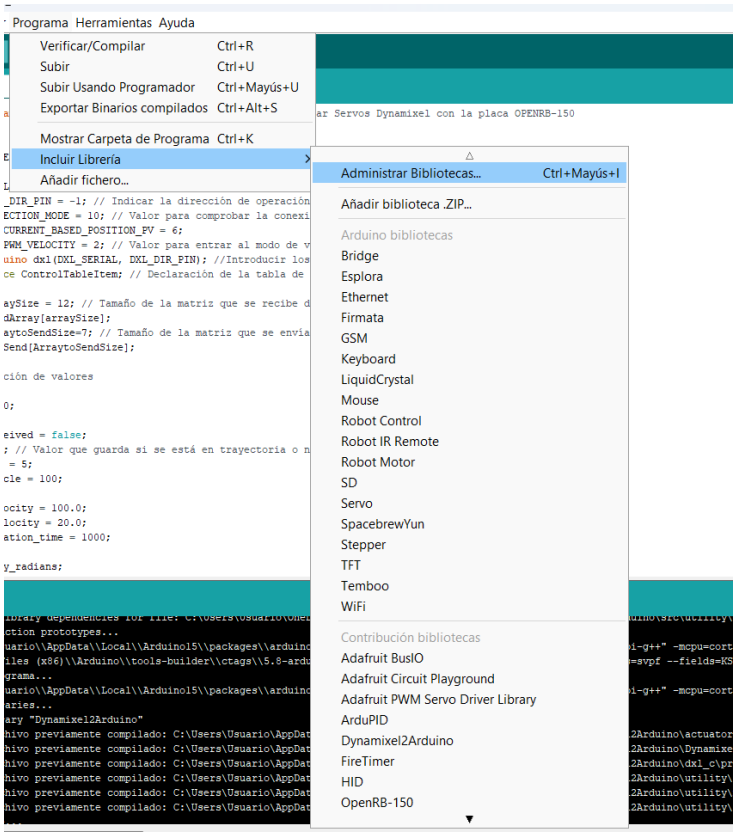


Imagen 26. Pasos para abrir el administrador de Bibliotecas de Arduino

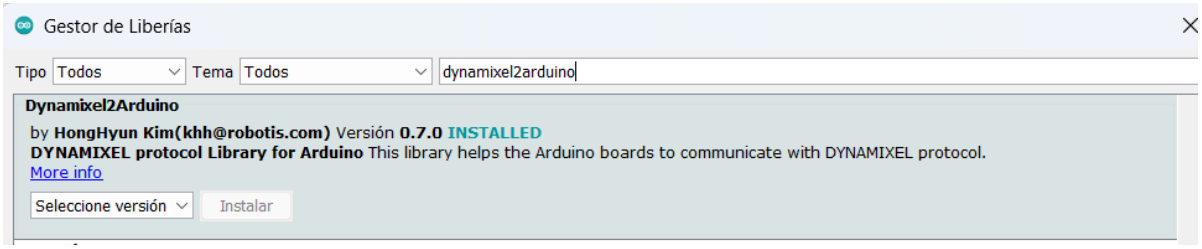


Imagen 27. Librería Dynamixel2Arduino en el gestor de librerías

Antes de continuar, hay que recordar que primero se tiene que conectar la fuente de alimentación a la placa U2D2 Power Hub y pulsar el interruptor. Si se encienden las luces de la placa OPENRB-150 todo es correcto ya se puede conectar la placa al ordenador. Entonces se compila el código y si no hay ningún error se programa la placa utilizando el botón de "Subir" de Arduino IDE. Al acabar aparecerá el mensaje de "Subido".

3.1.3 Instalación de la aplicación control del servo

Para la instalación de esta aplicación se necesita tener en primer instante el archivo de instalación creado por Matlab App designer:

Imagen 28. Instalador de la aplicación del control del Servo

Una vez se tenga ese ejecutable y se pulse abrir solo es necesario seguir las instrucciones del instalador. Una vez acabada la instalación se creará un programa que contiene la aplicación de la herramienta.

3.2 Control de calidad

Con el objetivo de comprobar si se ha realizado el montaje e instalación de forma efectiva, se han de comprobar los siguientes apartados:

- La herramienta en modo horizontal no se comienza a desmontar si se deja sin supervisión.
- La herramienta es capaz de cambiar a su modo vertical (sobre una de sus paredes) y todos los dispositivos se mantienen en su posición sin ningún riesgo de caída o golpe.
- El servomotor es capaz de girar 360 º sin ninguna interferencia, ya sea en vertical u horizontal.
- Al conectar la fuente de alimentación y pulsar el interruptor de placa de alimentación, se encienden las luces leds de las placas.
- Si se conecta el cable de comunicación con la placa OPENRB-150 pero la alimentación no está conectada, el sistema permanece desactivado. El sistema solo ha de encenderse a través de la placa de alimentación, no a través de ningún otro sistema.
- Una vez realizada la instalación del programa en la placa OPENRB-150 se puede observar como parpadea la luz del servomotor estando a la espera de instrucciones.

Si falla cualquiera de los casos anteriores hay un problema que necesita solucionarse.

4 PRUEBAS Y AJUSTES FINALES O DE SERVICIO

En el caso de que todos los materiales hayan sido considerados correctos y las pruebas básicas del montaje, se puede proceder a pruebas avanzadas como comprobar el correcto funcionamiento de la conexión ordenador-servomotor y hacer pruebas básicas con todos los modos de la aplicación.

Como ajustes finales es necesario comprobar que todos los valores por defecto de la aplicación ofrecen respuestas estables ya que al usarse servomotores, masas o materiales que difieran del prototipo los valores calculados y comprobados en este proyecto pueden resultar en fallos.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.

Trabajo Fin de Grado
Grado en Ingeniería Electrónica Industrial y Automática

Documento 4:

Presupuesto

Autor: José Gutiérrez Jiménez
Tutor: Ranko Zotovic Stanisic

Curso académico 2023-2024

En este apartado se procede a mostrar el presupuesto final del proyecto, que tiene dos categorías diferenciadas: presupuesto de fabricación de la herramienta, que puede ser usado como referencia en el caso se quiera montar varios prototipos; y el presupuesto del diseño y desarrollo de todo el trabajo.

1. Cuadro de precios de fabricación del prototipo				
Ud.	**Descripción**	**Cantidad**	**Precio(€/ud)**	**Total(€)**
ud.	OpenRB-150 Starter Kit (Placa + servo)	1	42,75	42,75
ud.	Placa U2D2 Power Hub	1	17,7	17,7
ud.	Coste transporte piezas Robotis	1	45	45
ud.	2 Fuentes de alimentación 5 V / 2 A	1	7,99	7,99
ud.	Cable de alimentación	1	2,94	2,94
ud.	Adaptador cable alimentación	1	1,89	1,89
g	Piezas impresión 3D	46	0,2	9,2
ud.	Piezas corte láser metacrilato	6	14	84
ud.	Placas metálicas Mod.2- 100	5	0,47	2,35
ud.	Caja tornillos Fandway (M3*6)	1	8,99	8,99
ud.	Caja tornillos (M2*6, M2.6*12)	1	11,69	11,69
ud.	Bolsa tornillos rosca M3x10 (30 ud)	1	2,39	2,39
g	4 tornillos M3x18	4	0,01	0,04
ud.	Caja tornillos rosca M3x20 (30 ud)	1	1,2	1,2
ud.	Bolsa tornillos M6 x 18 (8 ud)	1	2,79	2,79
ud.	Arduino IDE	1	0	0
h	Técnico encargado del montaje e instalación	0,5	12,5	6,25
			Total	**247,17**

Tabla 1. Tabla de Cuadro de precios de fabricación del prototipo

En el caso de hacer varios prototipos muchas de las bolsas de tornillería no serían necesarias volver a comprarlas dado que para el montaje de un solo prototipo vienen tronillos de sobra en las bolsas seleccionadas, por lo que hacer una planificación de varios prototipos puede llegar a ahorrar en el presupuesto con vistas a futuro, sobretodo si se consigue ahorrar en el coste de transporte de las placas y servomotores. Dicho lo cual, el presupuesto final para construir la herramienta son **247.17 €.** Cabe aclarar que los costes de impresión son orientativos ya que estos se han hecho a través de la facultad y no han supuesto un coste real en el proyecto.

2. Cuadro de precios del diseño y desarrollo del prototipo				
Ud.	**Descripción**	**Cantidad**	**Precio(€/ud)**	**Total(€)**
h.	Horas dedicadas al proyecto por parte del alumno	275	13	3575
h.	Horas dedicadas al proyecto por parte del tutor	20	40	800
h.	Horas dedicadas al proyecto por parte del técnico de laboratorio	10	15	150
ud.	Licencia Matlab	1	250	250
ud.	Licencia Solidworks durante 3 meses	1	1044	1044
ud.	Arduino IDE	1	0	3575
			Total	**5819**

Tabla 2. Tabla del cuadro de precios del diseño y desarrollo del prototipo

El coste total de la suma de licencias más las horas trabajadas en el proyecto asciende a **5819 €.** Cabe destacar que gracias a las licencias de estudiantes no se ha comprado ninguna de ellas, pero en el precio es necesario que estén presentes porque si este proyecto fuese llevado a cabo en la realidad el ingeniero responsable tendría que asumir los costes de las licencias.

Como último cuadro de precios tenemos el total de coste del desarrollo y fabricación.

3. Cuadro de precios completo				
Ud.	**Descripción**	**Cantidad**	**Precio(€/ud)**	**Total(€)**
ud.	Coste de frabricación	1	247,17	247,17
ud.	Coste de desarrollo	1	5819	5819
			Total	**6066,17**

Tabla 3. Tabla con la suma total de los cuadro de precios anteriores

El presupuesto final y completo del proyecto asciende a **6066.97 €.**

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.

Trabajo Fin de Grado
Grado en Ingeniería Electrónica Industrial y Automática

Anexo:

Anexo A: Objetivos de Desarrollo Sostenible

Autor: José Gutiérrez Jiménez
Tutor: Ranko Zotovic Stanisic

Curso académico 2023-2024

En este anexo se presenta y detalla la conexión del TFG con los Objetivos de Desarrollo Sostenible (ODS) de la agenda 2030.

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS)

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

Tabla 1. Relación del trabajo con los Objetivos de Desarrollo Sostenible

Descripción de la alineación del TFG/TFM con los ODS con un grado de relación más alto.

Este proyecto se relaciona altamente con el Objetivo de Desarrollo Sostenible número 4 ya que la meta principal de este proyecto es complementar una formación sobre uno de los elementos más básicos de la industria tecnológica como es en el servomotor. Gracias a esta herramienta, los estudiantes podrán aplicar los conceptos teóricos aprendidos en clase y ver con sus propios ojos los efectos del cambio en las variables, resultando en un entendimiento mejor de un controlador PID y su importancia en controles automáticos.

Además, fomenta la interactividad y participación entre profesorado y alumnado ya que tener un prototipo real y de fácil acceso añade un dinamismo a las clases y prácticas simuladas que

sin lugar a dudas mejora la retención de conocimiento y el interés por la materia.

Por último, el proyecto deja lugar a espacio para posibles modificaciones que mejorarán las capacidades de análisis y resolución de problemas de futuros alumnos como ciertamente ha conseguido en mi caso.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Control de un servomotor Dynamixel mediante una placa compatible con Arduino para uso académico.

Trabajo Fin de Grado
Grado en Ingeniería Electrónica Industrial y Automática

Anexo:

Anexo B: Código de los programas

Autor: José Gutiérrez Jiménez
Tutor: Ranko Zotovic Stanisic

Curso académico 2023-2024

ÍNDICE

1. Código Arduino IDE

(1) CurrentvsVelocity.ino

Código utilizado para encontrar la relación entre la corriente y la velocidad.

```
#include <Dynamixel2Arduino.h>
const int DXL_DIR_PIN = -1; // DYNAMIXEL Shield DIR PIN
const uint8_t DXL_ID = 1;
int count=0;
int a=0;
float showcurrent[21];
float showvelocity[21];
Dynamixel2Arduino dxl(Serial1, DXL_DIR_PIN);
using namespace ControlTableItem;
void setup() {
  Serial.begin(115200);
  dxl.begin(57600);
  dxl.setPortProtocolVersion(2.0);
  dxl.ping(DXL_ID);
  dxl.torqueOff(DXL_ID);
  dxl.setOperatingMode(DXL_ID, OP_CURRENT);
  dxl.writeControlTableItem(CURRENT_LIMIT, DXL_ID, 1000);
  dxl.torqueOn(DXL_ID);
  delay(5000);
  Serial.println("Empieza ");

}

void loop() {
  if(count==0){

  for (int current = 0; current <= 500; current = current+25) {
    dxl.setGoalCurrent(DXL_ID, current,UNIT_MILLI_AMPERE);
    delay(1000); // Tiempo para asegurarse que el valor de corriente se ha alcanzado
    float velocity = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
    Serial.print("Corriente: ");
    Serial.print(current);
    Serial.print(" -> Velocidad: ");
    Serial.println(velocity);
    showcurrent[a]=current;
    showvelocity[a]=velocity;
    a++;
  }
  delay(1000);
  //Se reduce la velocidad a 0 de forma incremental
  for (float current = 500; current >= 0; current= current-50){
  dxl.setGoalCurrent(DXL_ID, current,UNIT_MILLI_AMPERE);
  delay(150);
  }
    Serial.print("current_values = [");
    for (int i=0; i <a; i++){
      Serial.print(showcurrent[i]);
      Serial.print(", ");
      }
      Serial.println("];");
```

```
    Serial.print("velocity_values_current = [");
    for (int i=0; i <a; i++){
      Serial.print(showvelocity[i]);
      Serial.print(", ");
      }
      Serial.println("];");
  count=1;//  Valor que para el bucle

}
}
```

(2) PWMvsVelocity.ino

Código utilizado para encontrar la relación entre la tensión (PWM) y la velocidad.

```
#include <Dynamixel2Arduino.h>
const int DXL_DIR_PIN = -1; // DYNAMIXEL Shield DIR PIN
const uint8_t DXL_ID = 1;
int count=0;
int a=0;
float showpwm[11];
float showvelocity[11];
Dynamixel2Arduino dxl(Serial1, DXL_DIR_PIN);
double duty_cycle=100.0;
using namespace ControlTableItem;
void setup() {
  Serial.begin(115200);
  dxl.begin(57600);
  dxl.setPortProtocolVersion(2.0);
  dxl.ping(DXL_ID);
  dxl.torqueOff(DXL_ID);
  dxl.setOperatingMode(DXL_ID, OP_PWM);

dxl.writeControlTableItem(PWM_LIMIT,DXL_ID,constrain((uint32_t)(8.85*duty_cycle),0,885))
;
  dxl.torqueOn(DXL_ID);
  delay(3000);
  Serial.println("Empieza ");
}

void loop() {
  if(count==0){
  for (float pwm = 0; pwm <= 100; pwm= pwm+10) {
    dxl.setGoalPWM(DXL_ID, pwm,UNIT_PERCENT);
    delay(1000); // Tiempo para asegurarse que el valor de PWM se ha alcanzado
    float velocity = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
    Serial.print("PWM: ");
    Serial.print(pwm);
    Serial.print(" -> Velocidad: ");
    Serial.println(velocity);
    Serial.print(" -> Corriente: ");
    Serial.println(dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE));
    showpwm[a]=pwm;
    showvelocity[a]=velocity;
    a++;
  }
  delay(1000);
```

```
  //Se reduce la velocidad a 0 de forma gradual
  for (float pwm = 100; pwm >= 0; pwm= pwm-10){
  dxl.setGoalPWM(DXL_ID, pwm,UNIT_PERCENT);
  delay(150);
  }
    Serial.print("pwm_values = [");
    for (int i=0; i <a; i++){
      Serial.print(showpwm[i]);
      Serial.print(", ");
      }
      Serial.println("];");
    Serial.print("velocity_values_pwm = [");
    for (int i=0; i <a; i++){
      Serial.print(showvelocity[i]);
      Serial.print(", ");
      }
      Serial.println("];");
  count=1;// Valor que para el bucle

  }

}
```

(3) CodigoTFG_Final.ino

Código final de placa OPENRB-150 para comunicarse con el servomotor y la aplicación:

```
#include <Dynamixel2Arduino.h> //Librería de Arduino para controlar Servos Dynamixel con la placa OPENRB-150
const uint8_t DXL_ID = 1; // ID del servo, de fábrica es 1

#define DXL_SERIAL Serial1

const long DXL_BAUDRATE = 57600;
const int DXL_DIR_PIN = -1; // Indicar la dirección de operación del servo, en este caso de lectura y escritura
const int CONECTION_MODE = 10; // Valor para comprobar la conexión del servo
const int OP_CURRENT_BASED_POSITION_PV = 6;
const int OP_PWM_VELOCITY = 2; // Valor para entrar al modo de velocidad por tensión
Dynamixel2Arduino dxl(DXL_SERIAL, DXL_DIR_PIN); //Introducir los valores al servo
using namespace ControlTableItem; // Declaración de la tabla de control del Servomtor

const int arraySize = 12; // Tamaño de la matriz que se recibe del ordenador
float receivedArray[arraySize];
const int ArraytoSendSize = 7; // Tamaño de la matriz que se envía al ordenador durante trayectorias
float ArraytoSend[ArraytoSendSize];

// Inicialización de valores

int indice = 0;
int caso = 0;
bool arrayReceived = false;
int count = 0; // Valor que guarda si se está en trayectoria o no
int delaysend = 5;
float duty_cycle = 100;
```

```
float max_velocity = 100.0;
float goal_velocity = 20.0;
float acceleration_time = 1000;

float velocity_radians;
float aceleration_radians;
float velocity_rpm2;
float acceleration_rpm2;

//Trapezoide
struct TrajectoryParams {
  float q0_deg;
  float qT_deg;
  float vmax_deg;
  float amax_deg;
  float vtraj_deg;
  float T1; // Tiempo aceleración/desaceleración
  float T2; // Tiempo
  float T; //tiempo total
  int dir;
};

struct PositionParam {
  float q_actual;
  float v_actual;
};


const float K_PWM = 0.877; // 171 g
const float K_current = 1.218;


// CustomPIDVelocity parámetros
float previous_output = 0;
float previous_error = 0;
float integral_error = 0;
long previous_Time = 0;


// Inicialización de Parámetros para PID
float max_rate_of_change_ = 1.0;
float initial_velocity = 0;


struct TrajectoryParamsV {
  float vmax_rpm;
  float amax_rpm;
  float vini_rpm;
  float vtraj_rpm;
  float T; //tiempo total
  float t0;
};

//Calcular Aceleracion
unsigned long initial_time = 0;
```

```
unsigned long start_time = 0;
unsigned long end_time = 0;// Static variable to store the last time
float last_velocity = 0;
static float acceleration = 0;
//

//funciones
void stopServo(uint8_t id);
void setAllParameterstoZero(uint8_t id);
void setOperatingModeCustom(uint8_t id, uint8_t mode, float data1, float data2);
void setup() {
  Serial.begin(115200);
  Serial.setTimeout(1000);
  dxl.begin(DXL_BAUDRATE);
  dxl.setPortProtocolVersion(2.0);
  dxl.ping(DXL_ID);
  dxl.torqueOff(DXL_ID);
  dxl.writeControlTableItem(DRIVE_MODE, DXL_ID, 0x00);
  dxl.writeControlTableItem(HOMING_OFFSET, DXL_ID, 0);
  dxl.writeControlTableItem(SHUTDOWN, DXL_ID, 0x35); // Configurar las alertas del servo
para que si hay errores en el hardware el motor pare

  dxl.torqueOn(DXL_ID);

  dxl.ledOn(DXL_ID);


  delay(1000);
  stopServo(DXL_ID);


}

void loop() {
  // Verifica que haya llegado el comando de array completo antes de entrar al selector de
modos

  if (arrayReceived) {
    count = 0;
    // Bucle donde se entra al modo de control recibido
    switch (int(receivedArray[0])) { // Switch solo funciona con valores int
      case CONECTION_MODE:
        if (dxl.ping(DXL_ID)) {
          for (int i = 0; i < arraySize; i++) {
            Serial.println(receivedArray[i]);
          }
        }
        break;
      case OP_EXTENDED_POSITION:
        if (dxl.ping(DXL_ID)) {

          if (receivedArray[11] == 0) {
            for (int i = 0; i < arraySize; i++) {
              Serial.println(receivedArray[i]);
            }
            Serial.flush();
```

```
        setOperatingModeCustom(DXL_ID, OP_EXTENDED_POSITION, receivedArray[2],
receivedArray[3]);
        initial_time = millis() + 1000;
      }


      delay(1000);
      //PID
      dxl.writeControlTableItem(POSITION_P_GAIN, DXL_ID, receivedArray[4]);
      dxl.writeControlTableItem(POSITION_I_GAIN, DXL_ID, receivedArray[5]);
      dxl.writeControlTableItem(POSITION_D_GAIN, DXL_ID, receivedArray[6]);

      dxl.setGoalPosition(DXL_ID, receivedArray[1], UNIT_DEGREE);

      do {
        ArraytoSend[0] = (millis() - initial_time) / 1000.0f;

        ArraytoSend[1] = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
        ArraytoSend[2] = dxl.readControlTableItem(POSITION_TRAJECTORY, DXL_ID) *
360 / 4096;

        //ArraytoSend[3]=dxl.getPresentVelocity(DXL_ID,UNIT_RPM)* ((2 * M_PI) / 60 );
        ArraytoSend[3] = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
        ArraytoSend[4] = dxl.readControlTableItem(VELOCITY_TRAJECTORY, DXL_ID) *
0.229;

        ArraytoSend[5] = dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
        ArraytoSend[6] = dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);

        for (int i = 0; i < ArraytoSendSize; i++) {
          Serial.println(ArraytoSend[i]);
        }
        delay(delaysend);
      } while ((Serial.available() == 0));

      Serial.flush();
      delay(100);

    }
    break;

  case OP_CURRENT: // Velocidad por Corriente
    if (dxl.ping(DXL_ID)) {
      if (receivedArray[11] == 0) {
        for (int i = 0; i < arraySize; i++) {
          Serial.println(receivedArray[i]);
        }
        Serial.flush();
        setOperatingModeCustom(DXL_ID, OP_CURRENT, 0, 0);
        initial_velocity = 0;
        initial_time = millis() + 1000;
      }
      delay(1000);
      float reference_velocity = receivedArray[2];
      float acceleration = receivedArray[3];
```

```
        TrajectoryParamsV trajV = calculateTrajectoryVelocity( reference_velocity,
initial_velocity, acceleration);
        // PID parameters
        float Kp = receivedArray[4]; //1000
        float Ki = receivedArray[5]; //100
        float Kd = receivedArray[6]; //700
        float v = 0;
        max_rate_of_change_ = 1;
        while (Serial.available() <= 0) {
          // Example usage of the PID functions
          v = customPIDVelocity(trajV,  Kp, Ki, Kd );

          dxl.setGoalCurrent(DXL_ID, v * K_current, UNIT_MILLI_AMPERE);

          // Envío de datos al ordenador
          ArraytoSend[0] = (millis() - initial_time) / 1000.0f;

          //ArraytoSend[3]=dxl.getPresentVelocity(DXL_ID,UNIT_RPM)* ((2 * M_PI) / 60 );
          ArraytoSend[3] =  dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
          ArraytoSend[4] =  dxl.readControlTableItem(GOAL_CURRENT, DXL_ID);


          ArraytoSend[5] = dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
          ArraytoSend[6] = dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);


          for (int i = 0; i < ArraytoSendSize; i++) {
            Serial.println(ArraytoSend[i]);
          }
          Serial.flush();
          delay(delaysend);
        }
        initial_velocity = reference_velocity;
      }

      break;
    case OP_PWM_VELOCITY: // Velocidad por Tension (PWM)
      if (dxl.ping(DXL_ID)) {
        if (receivedArray[11] == 0) {
          for (int i = 0; i < arraySize; i++) {
            Serial.println(receivedArray[i]);
          }
          Serial.flush();
          setOperatingModeCustom(DXL_ID, OP_PWM, 0, 0);
          initial_velocity = 0;
          initial_time = millis() + 1000;
        }
        delay(1000);
        float reference_velocity = receivedArray[2];
        float acceleration = receivedArray[3];

        TrajectoryParamsV trajV = calculateTrajectoryVelocity( reference_velocity,
initial_velocity, acceleration);
        // PID parámetros
        float Kp = receivedArray[4];
        float Ki = receivedArray[5];
```

```
      float Kd = receivedArray[6];
      float v = 0;
      max_rate_of_change_ = 1;
      while (Serial.available() <= 0) {
        // Example usage of the PID functions
        v = customPIDVelocity(trajV,  Kp, Ki, Kd );

        dxl.setGoalPWM(DXL_ID, v * K_PWM, UNIT_PERCENT);


        // Envío de datos al ordenador
        ArraytoSend[0] = (millis() - initial_time) / 1000.0f;

        //ArraytoSend[3]=dxl.getPresentVelocity(DXL_ID,UNIT_RPM)* ((2 * M_PI) / 60 );
        ArraytoSend[3] =  dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);
        ArraytoSend[4] =  dxl.readControlTableItem(GOAL_PWM, DXL_ID) * 100 / 885;


        ArraytoSend[5] = dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
        ArraytoSend[6] = dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);


        for (int i = 0; i < ArraytoSendSize; i++) {
          Serial.println(ArraytoSend[i]);
        }
        Serial.flush();
        delay(delaysend);
      }
      initial_velocity = reference_velocity;
    }
    break;
  case OP_PWM: // Tension/PWM
    if (dxl.ping(DXL_ID)) {


      if (receivedArray[11] == 0) {
        for (int i = 0; i < arraySize; i++) {
          Serial.println(receivedArray[i]);
        }
        Serial.flush();
        setOperatingModeCustom(DXL_ID, OP_PWM, 0, 0);
        initial_time = millis() + 1000;
      }

      delay(1000);
      //Actualizar parámetros
      if (receivedArray[2] == 0) {
        dxl.setGoalPWM(DXL_ID, receivedArray[1], UNIT_PERCENT);
        do {

          // Envío de datos al ordenador
          ArraytoSend[0] = (millis() - initial_time) / 1000.0f;

          ArraytoSend[1] = dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);
          ArraytoSend[2] = dxl.readControlTableItem(GOAL_PWM, DXL_ID) * 100 / 885;
```

```
          ArraytoSend[3] = dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
          ArraytoSend[4] = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);


          ArraytoSend[5] = dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE); // Se repite para no hacer una excepción con este array
          ArraytoSend[6] = dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);

          for (int i = 0; i < ArraytoSendSize; i++) {
            Serial.println(ArraytoSend[i]);
          }
          delay(delaysend);
          Serial.flush();
        } while ((Serial.available() == 0));
      } else if (receivedArray[2] == 1) {
        dxl.setGoalPWM(DXL_ID, - receivedArray[1], UNIT_PERCENT);
        do {

          //Nuevo
          ArraytoSend[0] = (millis() - initial_time) / 1000.0f;

          ArraytoSend[1] = -dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);
          ArraytoSend[2] = -dxl.readControlTableItem(GOAL_PWM, DXL_ID) * 100 / 885;


          ArraytoSend[3] = -dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
          ArraytoSend[4] = -dxl.getPresentVelocity(DXL_ID, UNIT_RPM);


          ArraytoSend[5] = -dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
          ArraytoSend[6] = -dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);
          for (int i = 0; i < ArraytoSendSize; i++) {
            Serial.println(ArraytoSend[i]);
          }
          delay(delaysend);
          Serial.flush();
        } while ((Serial.available() == 0));
      }
      Serial.flush();
      delay(100);
    }
    break;
  case OP_VELOCITY: // Modo velocidad
    if (dxl.ping(DXL_ID)) {
      if (receivedArray[11] == 0) {
        for (int i = 0; i < arraySize; i++) {
          Serial.println(receivedArray[i]);
        }
        Serial.flush();
        setOperatingModeCustom(DXL_ID, OP_VELOCITY, 0, receivedArray[3]);
        initial_time = millis() + 1000;
      }

      delay(1000);
      //Actualizar parámetros
```

```
        dxl.writeControlTableItem(VELOCITY_P_GAIN, DXL_ID, receivedArray[4]);
        dxl.writeControlTableItem(VELOCITY_I_GAIN, DXL_ID, receivedArray[5]);
        float velocity_rpm = receivedArray[2] * (60 / (2 * M_PI));
        dxl.setGoalVelocity(DXL_ID, velocity_rpm , UNIT_RPM);
        do {

          // Envío de datos al ordenador
          ArraytoSend[0] = (millis() - initial_time) / 1000.0f;

          ArraytoSend[1] = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
          ArraytoSend[2]  =  dxl.readControlTableItem(VELOCITY_TRAJECTORY,  DXL_ID)  *
0.229;

          //ArraytoSend[3]=dxl.getPresentVelocity(DXL_ID,UNIT_RPM)* ((2 * M_PI) / 60 );
          ArraytoSend[3] = dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
          ArraytoSend[4] = dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);

          ArraytoSend[5] = dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
          ArraytoSend[6] = dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);

          for (int i = 0; i < ArraytoSendSize; i++) {
            Serial.println(ArraytoSend[i]);
          }
          delay(delaysend);
        } while ((Serial.available() == 0));
        Serial.flush();
        delay(100);

      }

      break;
    case OP_CURRENT_BASED_POSITION: // Modo de posición por corriente
      if (dxl.ping(DXL_ID)) {

        if (receivedArray[11] == 0) {
          for (int i = 0; i < arraySize; i++) {
            Serial.println(receivedArray[i]);
          }
          Serial.flush();
          setOperatingModeCustom(DXL_ID, OP_CURRENT, 0, 0);
          initial_time = millis() + 1000;
        }
        delay(1000);

        // Recoge los valores mandados por el ordenador

        float q0_deg = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
        float qT_deg = receivedArray[1];
        float vmax_rad = receivedArray[2];  // Velocidad mandada en radianes
        float amax_rad = receivedArray[3];  // Aceleración mandada en radianes
        float Kp = receivedArray[4];
        float Ki = receivedArray[5];
        float Kv = receivedArray[6];
        float Friction = receivedArray[8];  // Fricción recibida
        float J = receivedArray[9];  // Inercia recibida
        float mass = receivedArray[10];  // Masa recibida
```

```
    TrajectoryParams traj = calculateTrajectory(q0_deg, qT_deg, vmax_rad, amax_rad);

    // Ejecutar el bucle de control según los parámetros recibidos
    controlLoop(DXL_ID, traj, Kp, Ki, Kv, J, mass, Friction);


    do {


      // Envío de datos al ordenador
      ArraytoSend[0] = (millis() - initial_time) / 1000.0f;

      ArraytoSend[1] = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
      ArraytoSend[2] = qT_deg;

      //ArraytoSend[3]=dxl.getPresentVelocity(DXL_ID,UNIT_RPM)* ((2 * M_PI) / 60 );
      ArraytoSend[3] = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
      ArraytoSend[4] = 0;

      ArraytoSend[5] = dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
      ArraytoSend[6] = dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);


      for (int i = 0; i < ArraytoSendSize; i++) {
        Serial.println(ArraytoSend[i]);
      }
      delay(delaysend);
    } while ((Serial.available() == 0));

    Serial.flush();
    delay(100);

  }
  break;
 case OP_CURRENT_BASED_POSITION_PV:
  if (dxl.ping(DXL_ID)) {

    if (receivedArray[11] == 0) {
      for (int i = 0; i < arraySize; i++) {
        Serial.println(receivedArray[i]);
      }
      Serial.flush();
      setOperatingModeCustom(DXL_ID, OP_CURRENT, 0, 0);
      initial_time = millis() + 1000;
    }
    delay(1000);
    float q0_deg = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
    // Valores recibidos del ordenador
    float qT_deg = receivedArray[1];
    float vmax_rad = receivedArray[2];
    float amax_rad = receivedArray[3];
    float Kp = receivedArray[4];
    float Ki = 0;
    float Kv = receivedArray[6];
    float J = receivedArray[9];  // Inercia
```

```
        float mass = receivedArray[10];  // Masa
        float Friction = receivedArray[8];  // Coficiente de fricción
        TrajectoryParams traj = calculateTrajectory(q0_deg, qT_deg, vmax_rad, amax_rad);

        // Bucle control
        controlLoopV(DXL_ID, traj, Kp, Ki, Kv, J, mass, Friction);

        do {



          // Envío de datos al ordenador
          ArraytoSend[0] = (millis() - initial_time) / 1000.0f;

          ArraytoSend[1] = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
          ArraytoSend[2] = qT_deg;


          ArraytoSend[3] = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
          ArraytoSend[4] = 0;

          ArraytoSend[5] = dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
          ArraytoSend[6] = dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);

          for (int i = 0; i < ArraytoSendSize; i++) {
            Serial.println(ArraytoSend[i]);
          }
          delay(delaysend);
        } while ((Serial.available() == 0));

        Serial.flush();

        delay(1000);
      }

      break;
    case 9: // Modo de cambio de parámetros
      break;
    default:
      stopServo(DXL_ID);
      dxl.torqueOff(DXL_ID);
      setAllParameterstoZero(DXL_ID);
      dxl.torqueOn(DXL_ID);
      break;
  }


  arrayReceived = false; // Se reincia el valor para la próxima operación
 }

 if (Serial.available() > 0) {

  arrayReceived = false;
  String receivedData = Serial.readStringUntil('\n'); // Lee los datos hasta el terminador, en
este caso '\n'
  float receivedNumber = receivedData.toFloat(); // Convierte los datos recibidos en numeros
```

```
    receivedArray[indice] = receivedNumber;
    indice++;
    if (receivedArray[0] == 9 && count == 0) {
      stopServo(DXL_ID);
      count = 1;
    }
    // Entra si se ha recibido toda la matriz
    if (indice >= arraySize) {
      arrayReceived = true;
      indice = 0; // Reinicio el índice para el siguiente array
    }


  } else {
    dxl.ledOn(DXL_ID);
    delay(250);
    // Turn off the LED on DYNAMIXEL
    dxl.ledOff(DXL_ID);
    delay(250);
  }

}
// Función personalizada para cambiar el modo de operación del servomotor
void setOperatingModeCustom(uint8_t id, uint8_t mode, float data1, float data2) {
  // Deshabilitar el par para poder cambiar parámetros de la tabla EEPROM
  dxl.torqueOff(id);
  setAllParameterstoZero(id);
  // Mandar la instrucción de cambiar al modo deseado
  dxl.setOperatingMode( id, mode);


  // Inicializar los parámetros convenientes de cada modo
  switch (mode) {
    case OP_CURRENT:  // Modo por corriente
      // Initialize CURRENT-specific parameters
      dxl.writeControlTableItem(DRIVE_MODE, DXL_ID, 0x00); // Drive mode gira en sentido
contrario de las agujas del reloj
      dxl.writeControlTableItem(CURRENT_LIMIT, DXL_ID, 1500);

      break;
    case OP_EXTENDED_POSITION:  // Modo de Posición Extendida (+360º)
      // Inicializar todos los parámetros relacionados con la Posición de Control
      velocity_radians = data1;
      aceleration_radians = data2;
      velocity_rpm2 = velocity_radians * (60 / (2 * M_PI));
      acceleration_rpm2 = aceleration_radians * ((60 * 60) / (2 * M_PI));
      dxl.writeControlTableItem(DRIVE_MODE, DXL_ID, 0x00);
      dxl.writeControlTableItem(PROFILE_VELOCITY,                    DXL_ID,
constrain((uint32_t)(velocity_rpm2 / 0.229), 0, 32737)); // Velocidad deseada durante la
trayectoria trapezoidal
      dxl.writeControlTableItem(PROFILE_ACCELERATION,                DXL_ID,
constrain((uint32_t)(acceleration_rpm2 / 214.577), 0, 32737)); // Aceleración deseada durante
la trayectoria trapezoidal
      //Cambiarlo a modo de tiempo e inicializar parámetros
      //
dxl.writeControlTableItem(DRIVE_MODE,DXL_ID,dxl.readControlTableItem(DRIVE_MODE,
```

```
DXL_ID)|0x04); //Lee el valor actual del drive MODE y cambia el bit 3 a modo de Tiempo
    //            dxl.writeControlTableItem(PROFILE_VELOCITY,DXL_ID,3000); //Tiempo de
trayectoria en ms
    //      dxl.writeControlTableItem(PROFILE_ACCELERATION,DXL_ID,200); // Tiempo de
aceleración en ms
     break;
   case OP_PWM:  // Modo de control PWM
     dxl.writeControlTableItem(PWM_LIMIT, DXL_ID, constrain((uint32_t)(8.85 * duty_cycle),
0, 885)); // PWM máximo que puede poner el usuario
     break;
   case OP_VELOCITY:  // Modo de control por velocidad
     acceleration_rpm2 = data2;
     dxl.writeControlTableItem(VELOCITY_LIMIT, DXL_ID, constrain((uint32_t)(max_velocity /
0.229), 0, 2047)); // Velocidad máxima que puede poner el usuario
     dxl.writeControlTableItem(DRIVE_MODE,                                    DXL_ID,
dxl.readControlTableItem(DRIVE_MODE, DXL_ID) & 0x0B);
     dxl.writeControlTableItem(PROFILE_ACCELERATION,                          DXL_ID,
constrain((uint32_t)(acceleration_rpm2 / 214.577), 0, 32737)); // Aceleración deseada durante
la trayectoria


     break;

   default:

     break;
  }

  // Habilitar el par después de cambiar los valores y el modo de operación
  dxl.torqueOn(id);
}

//Función que reinicia/limpia cualquier parámetro que pueda afectar a otros modos
void setAllParameterstoZero(uint8_t id) {
  dxl.writeControlTableItem(DRIVE_MODE, DXL_ID, 0x00); // Drive mode gira en sentido
contrario de las agujas del reloj
  dxl.writeControlTableItem(PROFILE_VELOCITY, DXL_ID, 0);
  dxl.writeControlTableItem(PROFILE_ACCELERATION, DXL_ID, 0);
  dxl.writeControlTableItem(POSITION_P_GAIN, DXL_ID, 0);
  dxl.writeControlTableItem(POSITION_I_GAIN, DXL_ID, 0);
  dxl.writeControlTableItem(POSITION_D_GAIN, DXL_ID, 0);
  dxl.writeControlTableItem(VELOCITY_P_GAIN, DXL_ID, 0);
  dxl.writeControlTableItem(VELOCITY_I_GAIN, DXL_ID, 0);
  dxl.writeControlTableItem(FEEDFORWARD_1ST_GAIN, DXL_ID, 0);
  dxl.writeControlTableItem(FEEDFORWARD_2ND_GAIN, DXL_ID, 0);
  dxl.writeControlTableItem(VELOCITY_LIMIT, DXL_ID, 0);
  dxl.writeControlTableItem(PROFILE_ACCELERATION, DXL_ID, 0);

}

void stopServo(uint8_t id) {
  int actual_position;
  double velocity_radians = 2;
  double aceleration_radians = 1;
  double velocity_rpm2 = velocity_radians * (60 / (2 * M_PI));
  double acceleration_rpm2 = aceleration_radians * ((60 * 60) / (2 * M_PI));
```

```
  actual_position = dxl.getPresentPosition(id, UNIT_DEGREE);
  // Al ser la función de parada no se usa la función Custom para ganar velocidad de procedimiento
  dxl.torqueOff(id);

  dxl.setOperatingMode(id, OP_POSITION);
  dxl.writeControlTableItem(DRIVE_MODE, id, 0x00); // El servo gira en sentido de las agujas del reloj
  if (dxl.ping(id)) {

    int rev = actual_position / 360;
    actual_position = actual_position - rev * 360;
    if (actual_position <= 180) {
      dxl.writeControlTableItem(DRIVE_MODE, id, 0x00);
    } else {
      dxl.writeControlTableItem(DRIVE_MODE, id, 0x01);
    }
    dxl.writeControlTableItem(PROFILE_VELOCITY, id, constrain((uint32_t)(velocity_rpm2 / 0.229), 0, 32737));
    dxl.writeControlTableItem(PROFILE_ACCELERATION, id, constrain((acceleration_rpm2 / 214.577), 0, 32737));

    dxl.torqueOn(id);
    dxl.writeControlTableItem(POSITION_P_GAIN, id, 900);
    dxl.writeControlTableItem(POSITION_I_GAIN, id, 100);
    dxl.writeControlTableItem(POSITION_D_GAIN, id, 1000);
    if (actual_position != 0) {
      dxl.setGoalPosition(id, 1);
      //while (dxl.readControlTableItem(MOVING, id) == 0) {};
      do {
        delay(20);
      } while (dxl.readControlTableItem(MOVING, id) != 0);
    }
  }
}


// Cálculo de la trayectoria trapezoidal
TrajectoryParams calculateTrajectory(float q0_deg, float qT_deg, float vmax_rad, float amax_rad) {
  TrajectoryParams traj;

  traj.q0_deg = q0_deg;
  traj.qT_deg = qT_deg;
  traj.vmax_deg = vmax_rad * (180.0 / M_PI);
  traj.amax_deg = amax_rad * (180.0 / M_PI);

  float q_diff = traj.qT_deg - traj.q0_deg;
  traj.dir = (q_diff > 0) ? 1 : (q_diff < 0 ? -1 : 0);
  q_diff = fabs(q_diff);

  traj.T1 = traj.vmax_deg / traj.amax_deg;
  float q_accel = 0.5 * traj.amax_deg * traj.T1 * traj.T1;

  if (q_accel * 2 > q_diff) {
    traj.T1 = sqrt(q_diff / traj.amax_deg);
```

```
    traj.T2 = traj.T1;
    traj.vmax_deg = traj.amax_deg * traj.T1;
  } else {
    traj.T2 = (q_diff - 2 * q_accel) / traj.vmax_deg + traj.T1;
  }

  traj.T = 2 * traj.T1 + (traj.T2 - traj.T1);

  return traj;
}

// Función para calcular la posición en el tiempo t según la trayectoria
PositionParam calculatePosition(const TrajectoryParams& traj, float t) {
  PositionParam pos;
  if (t < traj.T1) { // Etapa de aceleración
    pos.q_actual = traj.q0_deg + 0.5f * traj.dir * traj.amax_deg * t * t;
    pos.v_actual = traj.dir * traj.amax_deg * t;
  } else if (t < traj.T2) { // Etapa de velocidad constante
    pos.q_actual = traj.q0_deg + 0.5f * traj.dir * traj.amax_deg * traj.T1 * traj.T1 + traj.dir *
traj.vmax_deg * (t - traj.T1);
    pos.v_actual = traj.dir * traj.vmax_deg;
  } else if (t < traj.T) { //Etapa de desaceleración
    float t_dec = t - traj.T2;
    pos.q_actual = traj.q0_deg + 0.5f * traj.dir * traj.amax_deg * traj.T1 * traj.T1 + traj.dir *
traj.vmax_deg * (traj.T2 - traj.T1) + traj.dir * traj.vmax_deg * t_dec - 0.5f * traj.dir *
traj.amax_deg * t_dec * t_dec;
    pos.v_actual = traj.dir * traj.vmax_deg - traj.dir * traj.amax_deg * t_dec;
  } else if (t > traj.T) { //Etapa donde se debería haber llegado ya a la posición objetivo
    pos.q_actual = traj.qT_deg;
    pos.v_actual = 0;
  }
  return pos;
}

// Función que realiza el bucle de control
void controlLoop(const uint8_t DXL_ID, const TrajectoryParams& traj, float Kp, float Ki, float
Kv, float J, float mass, float Friction) {
  float t0 = millis() / 1000.0f;
  float t = 0;

  long previousTime = millis();
  float ePrevious = 0;
  float eIntegral = 0;
  float e_velocity = 0;
  const float G = 9.81f; // Gravedad
  const float Xc = 0.112f; //112 mm (longitud del eslabón)
  const float Kt = 0.354f; // Constante del par

  float max_rate_of_change = 4;
  float previous_output = 0;
  PositionParam pos;
  int a = 0;


  while (Serial.available() <= 0 && t < (traj.T * 2)) {
    t = (millis() / 1000.0f) - t0;
```

```
  PositionParam pos = calculatePosition(traj, t);

  // Calcular cuanto tiempo ha pasado en segundos del anterior ciclo
  float currentTime = millis();
  float deltaT = (currentTime - previousTime) / 1000.0f;

  //Lecturas del servo
  float q_real = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
  float v_real = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);

  // Calculo del error
  float e = pos.q_actual - q_real ;

  eIntegral = eIntegral + e * deltaT;
  e_velocity = (pos.v_actual * 60 / 360) - v_real ;

  // Calculo del par necesario para mover el servo

  float gravity_compensation = 0.5 * mass * G * Xc * cos(q_real * (M_PI / 180.0));
  float acceleration = J * (traj.amax_deg * (M_PI / 180.0) );
  float Friction_compensation = - Friction * sign(v_real); // Da igual unidades dado que la funcion signo solo devuelve el valor
  float control_output = (Kp * e) + (Kv * e_velocity) + (Ki * eIntegral) + Friction_compensation + gravity_compensation + acceleration ; //torque


  // Limita el valor de salida de control para que sea cercano al anterior

  float delta_output = control_output - previous_output;
  if (fabs(delta_output) > max_rate_of_change) {
    delta_output = max_rate_of_change * (delta_output / fabs(delta_output));
  }

  control_output = previous_output + delta_output;


  // Actualización de los valores
  previous_output = control_output;
  previousTime = currentTime;
  ePrevious = e;

  // Instrucciones al servo

  float current_goal = control_output / Kt; // Se convierte el par necesario a la corriente necesaria
  dxl.setGoalCurrent(DXL_ID, current_goal, UNIT_MILLI_AMPERE);


  // Mandar los valores al Ordenador
  ArraytoSend[0] = (millis() - initial_time) / 1000.0f;

  ArraytoSend[1] = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
  ArraytoSend[2] = pos.q_actual;
```

```
    ArraytoSend[3] = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
    ArraytoSend[4] = pos.v_actual;

    ArraytoSend[5] = dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
    ArraytoSend[6] = dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);

    for (int i = 0; i < ArraytoSendSize; i++) {
      Serial.println(ArraytoSend[i]);
    }
    Serial.flush();

    delay(5);
  }

}


void controlLoopV(const uint8_t DXL_ID, const TrajectoryParams& traj, float Kp, float Ki, float
Kv, float J, float mass, float Friction) {
  float t0 = millis() / 1000.0f;
  float t = 0;

  long previousTime = millis();
  float ePrevious = 0;
  float eIntegral = 0;
  float e_velocity = 0;
  const float G = 9.81f;
  const float Xc = 0.112f;
  const float Kt = 0.354f;

  float max_rate_of_change = 5;
  float previous_output = 0;
  PositionParam pos;
  int a = 0;
  while (Serial.available() <= 0 && t < (traj.T * 2)) {
    t = (millis() / 1000.0f) - t0;

    PositionParam pos = calculatePosition(traj, t);

    // Calcular cuanto tiempo ha pasado en segundos del anterior ciclo
    float currentTime = millis();
    float deltaT = (currentTime - previousTime) / 1000.0f;

    //Lecturas del servo
    float q_real = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
    float v_real = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);

    // Calculo del error
    float e = pos.q_actual - q_real ;

    //eIntegral = eIntegral + e * deltaT;

    e_velocity = (v_real) * 0.01;

    // Calculo del par necesaria para mover el servo
```

```
    float gravity_compensation = 0.5 * mass * G * Xc * cos(q_real * (M_PI / 180.0));
    float acceleration = J * (traj.amax_deg * (M_PI / 180.0));
    float Friction_compensation = - Friction * sign(v_real); // da igual unidades dado que la funcion signo solo devuelve el valor
    float control_output = (Kp * e) + (Kv * e_velocity) + Friction_compensation + gravity_compensation + acceleration ; //torque


    // Limita el valor de salida de control para que sea cercano al anterior

    float delta_output = control_output - previous_output;
    if (fabs(delta_output) > max_rate_of_change) {
      delta_output = max_rate_of_change * (delta_output / fabs(delta_output));
    }
    control_output = previous_output + delta_output;


    // Actualización de los valores
    previous_output = control_output;
    previousTime = currentTime;
    ePrevious = e;

    // Instrucciones al servo

    float current_goal = control_output / Kt; // Se pasa de par a corriente
    dxl.setGoalCurrent(DXL_ID, current_goal, UNIT_MILLI_AMPERE);


    // Enviar valores a Matlab
    ArraytoSend[0] = (millis() - initial_time) / 1000.0f;

    ArraytoSend[1] = dxl.getPresentPosition(DXL_ID, UNIT_DEGREE);
    ArraytoSend[2] = pos.q_actual;

    //ArraytoSend[3]=dxl.getPresentVelocity(DXL_ID,UNIT_RPM)* ((2 * M_PI) / 60 );
    ArraytoSend[3] = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
    ArraytoSend[4] = pos.v_actual;

    ArraytoSend[5] = dxl.getPresentCurrent(DXL_ID, UNIT_MILLI_AMPERE);
    ArraytoSend[6] = dxl.getPresentPWM(DXL_ID, UNIT_PERCENT);

    for (int i = 0; i < ArraytoSendSize; i++) {
      Serial.println(ArraytoSend[i]);
    }
    Serial.flush();

    delay(10);
  }
}

// Función que devuelve el signo de un valor
float sign(float value) {
  float x;
  if (value > 0) {
    x = 1;
  } else if ( value < 0 ) {
```

```
    x = -1;
  } else {
    x = 0;
  }
  return x;
}

//Cálculo de la trayectoria para los modos de velocidad
TrajectoryParamsV calculateTrajectoryVelocity( float vmax_rad, float vini_rad, float
amax_rad) {
  TrajectoryParamsV traj;
  previous_Time = millis();

  previous_error = 0;
  integral_error = 0;

  traj.vmax_rpm = vmax_rad * (60 / (2 * M_PI)); // Velocidad en RPM
  traj.amax_rpm = amax_rad * (60 / (2 * M_PI));
  traj.vini_rpm = vini_rad * (60 / (2 * M_PI));
  previous_output = traj.vini_rpm;

  float v_diff = traj.vmax_rpm - traj.vini_rpm;
  float dir = (v_diff > 0) ? 1 : (v_diff < 0 ? -1 : 0);

  v_diff = fabs(v_diff);

  traj.T = v_diff / traj.amax_rpm;
  traj.amax_rpm =  traj.amax_rpm * dir;
  float t0 = millis() / 1000.0f;
  traj.t0 = t0;
  return traj;
}

//Función que devuelve la velocidad v según el tiempo t basado en la trayectoria
float calculateVelocity(const TrajectoryParamsV &traj, float t) {
  float velocity;
  if (t < traj.T) {
    velocity = traj.vini_rpm + traj.amax_rpm * t;
  }  else {
    velocity = traj.vmax_rpm;
  }
  return velocity;

}



// Función de control del modo velocidad
float customPIDVelocity(const TrajectoryParamsV &traj,  float kp, float ki, float kd) {

  float t = (millis() / 1000.0f) - traj.t0;

  float velocity = calculateVelocity(traj, t);

  float current_velocity = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
```

```
  // Cáclulo de la diferencia de tiempo
  float current_Time = millis();
  float deltaT = (current_Time - previous_Time) / 1000.0f;  // Diferencia en segundos

  // Calcular errores y salida del PID
  float error = velocity - current_velocity;
  integral_error = integral_error + error * deltaT;
  float derivative_error = (error - previous_error) / deltaT;
  float control_output = kp * error + ki * integral_error + kd * derivative_error;

  // Limita el valor de salida de control para que sea cercano al anterior
  float delta_output = control_output - previous_output;
  if (fabs(delta_output) > max_rate_of_change_) {
    delta_output = max_rate_of_change_ * (delta_output / fabs(delta_output));
  }
  control_output = previous_output + delta_output;

  // Se actualizan los valores anteriores
  previous_output = control_output;
  previous_error = error;
  previous_Time = current_Time;

  ArraytoSend[1] = dxl.getPresentVelocity(DXL_ID, UNIT_RPM);
  ArraytoSend[2] = velocity;

  // Se devuelve la acción de control calculada
  return control_output;

}
```

2. Código Matlab

(1) Calculo_constantes.m

Código desarrollado para calcular las constantes de corriente, tensión y rozamiento. Se añaden también los valores mostrados del servo para una comprensión más clara:

```
%% Calculo de constantes
% Empieza
% PWM: 0.00 -> Velocidad: 0.00
%  -> Corriente: 0.00
% PWM: 10.00 -> Velocidad: 8.24
%  -> Corriente: 17.00
% PWM: 20.00 -> Velocidad: 21.76
%  -> Corriente: 24.00
% PWM: 30.00 -> Velocidad: 30.23
%  -> Corriente: 41.00
% PWM: 40.00 -> Velocidad: 43.28
%  -> Corriente: 48.00
% PWM: 50.00 -> Velocidad: 54.50
%  -> Corriente: 59.00
% PWM: 60.00 -> Velocidad: 65.04
%  -> Corriente: 67.00
% PWM: 70.00 -> Velocidad: 74.20
%  -> Corriente: 86.00
% PWM: 80.00 -> Velocidad: 87.02
%  -> Corriente: 88.00
% PWM: 90.00 -> Velocidad: 94.12
%  -> Corriente: 125.00
```

```
% PWM: 100.00 -> Velocidad: 106.03
%  -> Corriente: 119.00
% pwm_values = [0.00, 10.00, 20.00, 30.00, 40.00, 50.00, 60.00, 70.00, 80.00, 90.00, 100.00,
];
% velocity_values_pwm = [0.00, 8.24, 21.76, 30.23, 43.28, 54.50, 65.04, 74.20, 87.02, 94.12,
106.03, ];


%%
% Empieza
% Empieza
% Corriente: 0 -> Velocidad: 0.00
% Corriente: 25 -> Velocidad: 20.15
% Corriente: 50 -> Velocidad: 39.16
% Corriente: 75 -> Velocidad: 64.58
% Corriente: 100 -> Velocidad: 85.42
% Corriente: 125 -> Velocidad: 103.51
% Corriente: 150 -> Velocidad: 106.03
% Corriente: 175 -> Velocidad: 107.63
% Corriente: 200 -> Velocidad: 107.17
% Corriente: 225 -> Velocidad: 104.88
% Corriente: 250 -> Velocidad: 104.65
% Corriente: 275 -> Velocidad: 106.71
% Corriente: 300 -> Velocidad: 107.17
% Corriente: 325 -> Velocidad: 106.03
% Corriente: 350 -> Velocidad: 103.74
% Corriente: 375 -> Velocidad: 105.80
% Corriente: 400 -> Velocidad: 107.17
% Corriente: 425 -> Velocidad: 106.71
% Corriente: 450 -> Velocidad: 104.88
% Corriente: 475 -> Velocidad: 103.74
% Corriente: 500 -> Velocidad: 105.80
% current_values = [0.00, 25.00, 50.00, 75.00, 100.00, 125.00, 150.00, 175.00, 200.00, 225.00,
250.00, 275.00, 300.00, 325.00, 350.00, 375.00, 400.00, 425.00, 450.00, 475.00, 500.00, ];
% velocity_values_current = [0.00, 20.15, 39.16, 64.58, 85.42, 103.51, 106.03, 107.63, 107.17,
104.88, 104.65, 106.71, 107.17, 106.03, 103.74, 105.80, 107.17, 106.71, 104.88, 103.74,
105.80, ];



%%
% Datos
pwm_values = [0.00, 10.00, 20.00, 30.00, 40.00, 50.00, 60.00, 70.00, 80.00, 90.00, 100.00];
velocity_values_pwm = [0.00, 8.24, 21.76, 30.23, 43.28, 54.50, 65.04, 74.20, 87.02, 94.12,
106.03];

current_values = [0.00, 25.00, 50.00, 75.00, 100.00, 125.00, 150.00, 175.00, 200.00];
velocity_values_current = [0.00, 20.15, 39.16, 64.58, 85.42, 103.51, 106.03, 107.63, 107.17];

%Calculo de K_current
sum=0;

for i=2:1:length(pwm_values) %El primer valor es 0 por lo que no es necesario contabilizarlo,
además crea una número NaN
    relation=pwm_values(i)/velocity_values_pwm (i);
    sum= sum+ relation;
end

K_PWM= sum/length(pwm_values);
```

```
%Calculo de K_current
sum=0;

for i=2:1:length(current_values)
    relation=current_values(i)/velocity_values_current(i);
    sum= sum+ relation;
end

K_current= sum/length(current_values);

% Resultados de constantes
fprintf('K_PWM= %.3f\n', K_PWM);
fprintf('K_current= %.3f\n', K_current);
%%
% Empieza
% PWM: 0.00 -> Velocidad: 0.00
% -> Corriente: 0.00
% PWM: 1.00 -> Velocidad: 0.00
%  -> Corriente: 12.00
% PWM: 2.00 -> Velocidad: 0.00
%  -> Corriente: 13.00
% PWM: 3.00 -> Velocidad: 0.00
%  -> Corriente: 15.00
% PWM: 4.00 -> Velocidad: 2.75
%  -> Corriente: 16.00
% PWM: 5.00 -> Velocidad: 4.81
%  -> Corriente: 14.00
% PWM: 6.00 -> Velocidad: 5.27
%  -> Corriente: 17.00
% PWM: 7.00 -> Velocidad: 5.72
%  -> Corriente: 17.00
% PWM: 8.00 -> Velocidad: 5.95
%  -> Corriente: 20.00
% PWM: 9.00 -> Velocidad: 8.47
%  -> Corriente: 19.00
% PWM: 10.00 -> Velocidad: 10.08
%  -> Corriente: 20.00
% pwm_values = [0.00, 1.00, 2.00, 3.00, 4.00, 5.00, 6.00, 7.00, 8.00, 9.00, 10.00, ];
% velocity_values_pwm = [0.00, 0.00, 0.00, 0.00, 2.75, 4.81, 5.27, 5.72, 5.95, 8.47, 10.08, ];

Kt = 0.354; %Constante calculada
r=0.112; %en metros
% PWM: 4.00 -> Velocidad: 2.75
%  -> Corriente: 16.00

Current_change_A=16/1000;


Friction_Force=  Kt * Current_change_A;
fprintf('Friction_Force= %.5f\n', Friction_Force);
```

(2) ControlServo.mlapp

Aplicación desarrollada para la herramienta. Las instrcciones en inglés son comentarios puestos por el mismo programa:

```
classdef ControlServo < matlab.apps.AppBase

    % Properties that correspond to app components
```

```
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        LeyendaPanel                    matlab.ui.container.Panel
        Valor4Label                     matlab.ui.control.Label
        Valor3Label                     matlab.ui.control.Label
        ValorObjetivoLabel              matlab.ui.control.Label
        ValorRealLabel                  matlab.ui.control.Label
        KgLabel                         matlab.ui.control.Label
        Kgm2Label                       matlab.ui.control.Label
        NmLabel                         matlab.ui.control.Label
        radsLabel_2                     matlab.ui.control.Label
        Label                           matlab.ui.control.Label
        rads2Label                      matlab.ui.control.Label
        radsLabel                       matlab.ui.control.Label
        GradosLabel                     matlab.ui.control.Label
        MasaEditField                   matlab.ui.control.NumericEditField
        MasaEditFieldLabel              matlab.ui.control.Label
        MomentodeInerciaJEditField      matlab.ui.control.NumericEditField
        MomentodeInerciaJEditFieldLabel  matlab.ui.control.Label
        RozamientoEditField             matlab.ui.control.NumericEditField
        RozamientoEditFieldLabel        matlab.ui.control.Label
        RefrescarlistapuertosButton     matlab.ui.control.Button
        CambiarParmetrosButton          matlab.ui.control.Button
        CicloPWMSpinner                 matlab.ui.control.Spinner
        CicloPWMSpinnerLabel            matlab.ui.control.Label
        DireccionPWMSwitch              matlab.ui.control.Switch
        DireccionPWMSwitch_2Label       matlab.ui.control.Label
        EmpezarButton                   matlab.ui.control.Button
        VelocidaddeseadaSpinner         matlab.ui.control.Spinner
        VelocidaddeseadaSpinnerLabel    matlab.ui.control.Label
        AceleracinTrapezoideEditField   matlab.ui.control.NumericEditField
        AceleracinTrapezoideEditFieldLabel  matlab.ui.control.Label
        VelocidadTrapezoideEditField    matlab.ui.control.NumericEditField
        VelocidadTrapezoideEditFieldLabel  matlab.ui.control.Label
        DKdEditField                    matlab.ui.control.NumericEditField
        DKdEditFieldLabel               matlab.ui.control.Label
        IKiEditField                    matlab.ui.control.NumericEditField
        IKiEditFieldLabel               matlab.ui.control.Label
        PKpEditField                    matlab.ui.control.NumericEditField
        PKpEditFieldLabel               matlab.ui.control.Label
        ModosDropDown                   matlab.ui.control.DropDown
        ModosDropDownLabel              matlab.ui.control.Label
        PosiciondeseadaEditField        matlab.ui.control.NumericEditField
        PosiciondeseadaEditFieldLabel   matlab.ui.control.Label
        PararButton                     matlab.ui.control.Button
        PuertoSerialDropDownLabel       matlab.ui.control.Label
        PuertoSerialDropDown            matlab.ui.control.DropDown
        ConectarServoButton             matlab.ui.control.Button
        ServoConectadoLamp              matlab.ui.control.Lamp
        ServoConectadoLampLabel         matlab.ui.control.Label
        DesconectarButton               matlab.ui.control.Button
        UIAxes                          matlab.ui.control.UIAxes
    end


    properties (Access = private)
        com % Datos Puerto Serial
        MsgHandler % Administra cuantas Ventanas de Mensajes hay
        ConnectFunction = 10 % Valor para conectarse con el servo

        %Modos del servomotor
```

```
        OP_CURRENT = 0 % Valor modo corriente
        OP_VELOCITY = 1 % Valor modo velocidad
        OP_PWM_VELOCITY= 2 %Valor modo velocidad por pwm
        OP_POSITION = 3 % Valor modo posición
        OP_EXTENDED_POSITION = 4 % Valor modo posición extendida
        OP_CURRENT_BASED_POSITION = 5  % Valor modo posición por corriente
        OP_CURRENT_BASED_POSITION_PV = 6  % Valor modo posición por corriente versión PV
        OP_PWM = 16 % Valor modo pwm

        STOP = false % Valor que guarda si ha de parar la trayectoria

        % Matrices
        TimeData = [] % Matriz que guarda los datos de tiempo
        RealValue = [] % Matriz que guarda los valores reales del servo
        GoalValue = [] % Matriz que guarda los valores objetivos reales del servo
        ThirdData = [] % Matriz suplementaria que guarda valores importantes
        FourthData = [] % Matriz suplementaria que guarda valores importantes
        FifthData = [] % Matriz suplementaria que guarda valores importantes
        SixthData = [] % Matriz suplementaria que guarda valores importantes
        BufferSize = 100 % Valor que administra cuantos valores pueden quedarse esperando

        Line1 % Linea 1 de la gráfica
        Line2 % Linea 2 de la gráfica
        Line3 % Linea 3 de la gráfica
        Line4 % Linea 4 de la gráfica
        Mode % Valor que guarda en que modo se está
        arraySize = 12 % Valor que guarda el tamaño de la matriz que se envia al servo
        Direction = 0

    end

    methods (Access = private)
        %Función que lee los valores del microcontrolador
        function result = readArrayFromSerial(~,serialObj, arraySize)
            result = zeros(1, arraySize);% Leer los elementos de la raiz del puerto serial
            for i = 1:arraySize
                data = fscanf(serialObj, '%f');
                result(i) =  data;
            end

        end


        %Función que envia un mensaje y guarda que se ha hecho
        function  SendMsgBox(app, string)
            a = msgbox(string);
            app.MsgHandler = [app.MsgHandler, a];
        end



        %Función que representa en tiempo real los datos del servo y luego
        %genera las gráficas al final del recorrido
        function acquireData(app)
            app.SendMsgBox("Empieza la trayectoria");
            clear app.UIAxes
            %Incicialización de las lineas y se les asigna colores
            app.Line1= animatedline(app.UIAxes,"Color","g");
            app.Line2= animatedline(app.UIAxes,"Color","r");
            app.Line3= animatedline(app.UIAxes,"Color","y");
            app.Line4= animatedline(app.UIAxes,"Color","m");
```

```
app.UIAxes.YGrid='on';
app.UIAxes.XLim = [0 50];
app.STOP=false;
contador=1;

%Incialización de las matrices
app.TimeData= [];
app.RealValue = [];
app.GoalValue = [];
app.ThirdData= [];
app.FourthData = [];
app.FifthData = [];
app.SixthData = [];
while (app.STOP==false)



    if app.com.NumBytesAvailable > 0
        flush(app.com,"output");
        % Se reciben 7 valores del servo
        valor = app.readArrayFromSerial(app.com, 7);



        app.TimeData(contador) = valor(1); % Tiempo
        app.RealValue(contador) = valor(2); % Valor real
        app.GoalValue (contador) = valor(3); % Valor objetivo
        app.ThirdData(contador) = valor(4); % Dato 3
        app.FourthData(contador) = valor(5); % Dato 4
        app.FifthData(contador) = valor(6); % Dato 5
        app.SixthData(contador) = valor(7); % Dato 6

        addpoints(app.Line1, contador, app.RealValue(contador));
        addpoints(app.Line2, contador, app.GoalValue (contador));
        addpoints(app.Line3, contador, app.ThirdData(contador));
        addpoints(app.Line4, contador, app.FourthData(contador));
        app.UIAxes.XLim = [0 50+contador];

        drawnow
        contador=contador+1;

    end
    pause(0.005);
end

app.STOP=false;
app.SendMsgBox("Preparando Gráficas");
clearpoints(app.Line1);
clearpoints(app.Line2);
clearpoints(app.Line3);
clearpoints(app.Line4);
clear app.Line1
clear app.Line2
clear app.Line3
clear app.Line4

figure;
% Según el modo representa de una manera o otra
switch app.Mode
    case "Posicion"
```

```
                    subplot(4,1,1)

                    plot(app.TimeData, app.RealValue,Color='g');
                    hold on
                    plot(app.TimeData, app.GoalValue, Color='r');
                    minim = min(min(app.RealValue, app.GoalValue));
                    maxim = max(max(app.RealValue, app.GoalValue));
                    ylim([minim-10 maxim+10]);
                    error=app.equad(app.GoalValue,app.RealValue);

                    title("Posicion - Error Cuadrático: "+ num2str(error));
                    ylabel("Posicion (º)");
                    xlabel("Tiempo (s)");
                    legend("Posicion Real (º)", "Posicion objetivo (º)");

                    subplot(4,1,2)

                    plot(app.TimeData, app.ThirdData,Color='g');
                    hold on
                    plot(app.TimeData, app.FourthData,Color='r');
                    hold on
                    minim = min(min(app.ThirdData, app.FourthData)); %Doble porque el primer
min devuelve un set de datos
                    maxim = max(max(app.FourthData, app.ThirdData));
                    ylim([minim-10 maxim+10]);
                    error=app.equad(app.ThirdData,app.FourthData);

                    title("Velocidad - Error Cuadrático: "+ num2str(error));
                    ylabel("Velocidad (RPM)");
                    xlabel("Tiempo (s)");
                    legend("Velocidad Real (RPM)", "Velocidad objetivo (RPM)");

                    subplot(4,1,3)

                    accel= app.getAcceleration(app.ThirdData, app.TimeData);
                    plot(app.TimeData, accel,Color='g');
                    hold on
                    accel2 = app.getAcceleration(app.FourthData, app.TimeData);
                    plot(app.TimeData, accel2,Color='r');
                    hold on
                    minim = min(min(accel, accel2));
                    maxim = max(max(accel, accel2));
                    ylim([minim-20 maxim+20]);
                    error=app.equad(accel2,accel);

                    title("Aceleracion - Error Cuadrático: "+ num2str(error));
                    ylabel("Aceleración (RPM)");
                    xlabel("Tiempo (s)");
                    legend("Aceleración Real (RPM^2)", "Aceleración objetivo (RPM^2)");

                    subplot(4,1,4)

                    plot(app.TimeData, app.FifthData,Color='y');
                    hold on
                    plot(app.TimeData, app.SixthData, Color = "b");
                    hold on
                    minim = min(min(app.FifthData, app.SixthData));
                    maxim = max(max(app.FifthData, app.SixthData));
                    ylim([minim-10 maxim+10]);
                    xlabel("Tiempo (s)");
                    title("Corriente vs PWM");
```

```
        legend("Corriente (mA)", "PWM (%)");

    case "Tension/PWM"
        subplot(4,1,1)

        plot(app.TimeData, app.RealValue,Color='g');
        hold on
        plot(app.TimeData, app.GoalValue, Color='r');
        minim = min(min(app.RealValue, app.GoalValue));
        maxim = max(max(app.RealValue, app.GoalValue));
        ylim([minim-10 maxim+10]);
        ylabel("PWM (%)");
        xlabel("Tiempo (s)");
        error=app.equad(app.GoalValue,app.RealValue);
        title("PWM - Error Cuadrático: "+ num2str(error));
        legend("PWM real (%)", "PWM objetivo (%)");



        subplot(4,1,2)
        %plot(app.TimeData, app.ThirdData,Color='g');
        hold on
        plot(app.TimeData, app.FourthData,Color='r');
        minim = min(min(app.FourthData, app.FourthData));
        maxim = max(max(app.FourthData, app.FourthData));
        ylim([minim-10 maxim+10]);
        ylabel("Velocidad (RPM)");
        xlabel("Tiempo (s)");
        title("Velocidad");
        legend("Velocidad Real (RPM)", "Velocidad objetivo (RPM)");


        subplot(4,1,3)
        %accel= app.getAcceleration(app.ThirdData, app.TimeData);
        %plot(app.TimeData, accel,Color='g');
        hold on
        accel2 = app.getAcceleration(app.FourthData, app.TimeData);
        plot(app.TimeData, accel2,Color='r');
        minim = min(min(accel2, accel2));
        maxim = max(max(accel2, accel2));
        ylim([minim-10 maxim+10]);
        ylabel("Aceleración (RPM^2)");
        xlabel("Tiempo (s)");
        title("Aceleracion");
        legend("Aceleración Real (RPM^2)", "Aceleración objetivo (RPM^2)");

        subplot(4,1,4)
        plot(app.TimeData, app.FifthData,Color='y');
        hold on
        plot(app.TimeData, app.SixthData, Color = "b");
        minim = min(min(app.FifthData, app.SixthData));
        maxim = max(max(app.FifthData, app.SixthData));
        ylim([minim-10 maxim+10]);
        xlabel("Tiempo (s)");
        title("Corriente vs PWM");
        legend("Corriente (mA)", "PWM (%)");


    case    "Velocidad"
        subplot(3,1,1)
```

```
        plot(app.TimeData, app.RealValue,Color='g');
        hold on
        plot(app.TimeData, app.GoalValue, Color='r');
        hold on
        minim = min(min(app.RealValue, app.GoalValue));
        maxim = max(max(app.RealValue, app.GoalValue));
        ylim([minim-10 maxim+10]);
        error=app.equad(app.GoalValue,app.RealValue);
        title("Velocidad - Error Cuadrático: "+ num2str(error));
        ylabel("Velocidad (RPM)");
        xlabel("Tiempo (s)");
        legend("Velocidad Real (RPM)", "Velocidad objetivo (RPM)");

        subplot(3,1,2)
        accel= app.getAcceleration(app.RealValue, app.TimeData);
        plot(app.TimeData, accel,Color='g');
        hold on
        accel2 = app.getAcceleration(app.GoalValue, app.TimeData);
        plot(app.TimeData, accel2,Color='r');
        minim = min(min(accel, accel2));
        maxim = max(max(accel, accel2));
        ylim([minim-10 maxim+10]);
        error=app.equad(accel2,accel);
        title("Aceleracion - Error Cuadrático: "+ num2str(error));
        %title("Aceleracion");
        ylabel("Aceleración (RPM)");
        xlabel("Tiempo (s)");
        legend("Aceleración Real (RPM^2)", "Aceleración objetivo (RPM^2)");

        subplot(3,1,3)
        plot(app.TimeData, app.FifthData,Color='y');
        hold on
        plot(app.TimeData, app.SixthData, Color = "b");
        minim = min(min(app.FifthData, app.SixthData));
        maxim = max(max(app.FifthData, app.SixthData));
        ylim([minim-10 maxim+10]);
        xlabel("Tiempo (s)");
        title("Corriente vs PWM");
        legend("Corriente (mA)", "PWM (%)");

    case "Velocidad por Tension"
        subplot(4,1,1)


        plot(app.TimeData, app.ThirdData,Color='g');
        hold on
        plot(app.TimeData, app.FourthData,Color='r');
        minim = min(min(app.ThirdData, app.FourthData));
        maxim = max(max(app.ThirdData, app.FourthData));
        ylim([minim-10 maxim+10]);
        ylabel("PWM (%)");
        xlabel("Tiempo (s)");
        error=app.equad(app.FourthData, app.ThirdData);
        title("PWM - Error Cuadrático: "+ num2str(error));
        %title("PWM");
        legend("PWM real (%)", "PWM objetivo (%)");

        subplot(4,1,2)

        plot(app.TimeData, app.RealValue,Color='g');
        hold on
```

```
        plot(app.TimeData, app.GoalValue, Color='r');
        minim = min(min(app.RealValue, app.GoalValue));
        maxim = max(max(app.RealValue, app.GoalValue));
        ylim([minim-10 maxim+10]);
        error=app.equad(app.GoalValue,app.RealValue);
        title("Velocidad - Error Cuadrático: "+ num2str(error));
        %title("Velocidad");
        ylabel("Velocidad (RPM)");
        xlabel("Tiempo (s)");
        legend("Velocidad Real (RPM)", "Velocidad objetivo (RPM)");

        subplot(4,1,3)

        accel= app.getAcceleration(app.RealValue, app.TimeData);
        plot(app.TimeData, accel,Color='g');
        hold on
        accel2 = app.getAcceleration(app.GoalValue, app.TimeData);
        plot(app.TimeData, accel2,Color='r');
        minim = min(min(accel, accel2));
        maxim = max(max(accel, accel2));
        ylim([minim-10 maxim+10]);
        ylabel("Aceleración (RPM^2)");
        xlabel("Tiempo (s)");
        error=app.equad(accel2,accel);
        title("Aceleracion - Error Cuadrático: "+ num2str(error));
        %title("Aceleracion");
        legend("Aceleración Real (RPM^2)", "Aceleración objetivo (RPM^2)");

        subplot(4,1,4)
        plot(app.TimeData, app.FifthData,Color='y');
        hold on
        plot(app.TimeData, app.SixthData, Color = "b");
        minim = min(min(app.FifthData, app.SixthData));
        maxim = max(max(app.FifthData, app.SixthData));
        ylim([minim-10 maxim+10]);
        xlabel("Tiempo (s)");
        title("Corriente vs PWM");
        legend("Corriente (mA)", "PWM (%)");

    case "Velocidad por Corriente"
        subplot(4,1,1)


        plot(app.TimeData, app.ThirdData,Color='g');
        hold on
        plot(app.TimeData, app.FourthData,Color='r');
        minim = min(min(app.ThirdData, app.FourthData));
        maxim = max(max(app.ThirdData, app.FourthData));
        ylim([minim-10 maxim+10]);
        error=app.equad(app.FourthData, app.ThirdData);
        title("Corriente - Error Cuadrático: "+ num2str(error));
        %title("Corriente");
        ylabel("Corriente (mA)");
        xlabel("Tiempo (s)");
        legend("Corriente Real (RPM)", "Corriente objetivo (RPM)");

        subplot(4,1,2)

        plot(app.TimeData, app.RealValue,Color='g');
        hold on
        plot(app.TimeData, app.GoalValue, Color='r');
```

```
            minim = min(min(app.RealValue, app.GoalValue));
            maxim = max(max(app.RealValue, app.GoalValue));
            ylim([minim-10 maxim+10]);
            error=app.equad(app.GoalValue,app.RealValue);
            title("Velocidad - Error Cuadrático: "+ num2str(error));
            %title("Velocidad");
            ylabel("Velocidad (RPM)");
            xlabel("Tiempo (s)");
            legend("Velocidad Real (RPM)", "Velocidad objetivo (RPM)");

            subplot(4,1,3)
            accel= app.getAcceleration(app.RealValue, app.TimeData);
            plot(app.TimeData, accel,Color='g');
            hold on
            accel2 = app.getAcceleration(app.GoalValue, app.TimeData);
            plot(app.TimeData, accel2,Color='r');
            minim = min(min(accel, accel2));
            maxim = max(max(accel, accel2));
            ylim([minim-10 maxim+10]);
            ylabel("Aceleración (RPM^2)");
            xlabel("Tiempo (s)");
            error=app.equad(accel2,accel);
            title("Aceleracion - Error Cuadrático: "+ num2str(error));
            %title("Aceleracion");
            legend("Aceleración Real (RPM^2)", "Aceleración objetivo (RPM^2)");

            subplot(4,1,4)
            plot(app.TimeData, app.FifthData,Color='y');
            hold on
            plot(app.TimeData, app.SixthData, Color = "b");
            minim = min(min(app.FifthData, app.SixthData));
            maxim = max(max(app.FifthData, app.SixthData));
            ylim([minim-10 maxim+10]);
            xlabel("Tiempo (s)");
            title("Corriente vs PWM");
            legend("Corriente (mA)", "PWM (%)");

        case "Posicion por Corriente"
            subplot(4,1,1)

            plot(app.TimeData, app.RealValue,Color='g');
            hold on
            plot(app.TimeData, app.GoalValue, Color='r');
            minim = min(min(app.RealValue, app.GoalValue));
            maxim = max(max(app.RealValue, app.GoalValue));
            ylim([minim-10 maxim+10]);
            error=app.equad(app.GoalValue,app.RealValue);
            title("Posicion - Error Cuadrático: "+ num2str(error));
            %title("Posicion");
            ylabel("Posicion (º)");
            xlabel("Tiempo (s)");
            legend("Posicion Real (º)", "Posicion objetivo (º)");

            subplot(4,1,2)

            plot(app.TimeData, app.ThirdData,Color='g');
            hold on
            plot(app.TimeData, app.FourthData,Color='r');
            minim = min(min(app.ThirdData, app.FourthData));
            maxim = max(max(app.ThirdData, app.FourthData));
            ylim([minim-10 maxim+10]);
```

```
        error=app.equad(app.FourthData,app.ThirdData);

        title("Velocidad - Error Cuadrático: "+ num2str(error));
        %title("Velocidad");
        ylabel("Velocidad (RPM)");
        xlabel("Tiempo (s)");
        legend("Velocidad Real (RPM)", "Velocidad objetivo (RPM)");

        subplot(4,1,3)

        accel= app.getAcceleration(app.ThirdData, app.TimeData);
        plot(app.TimeData, accel,Color='g');
        hold on
        accel2 = app.getAcceleration(app.FourthData, app.TimeData);
        plot(app.TimeData, accel2,Color='r');
        minim = min(min(accel, accel2));
        maxim = max(max(accel, accel2));
        ylim([minim-20 maxim+20]);

        error=app.equad(accel2,accel);
        title("Aceleracion - Error Cuadrático: "+ num2str(error));
        %title("Aceleracion");
        ylabel("Aceleración (RPM^2)");
        xlabel("Tiempo (s)");
        legend("Aceleración Real (RPM^2)", "Aceleración objetivo (RPM^2)");

        subplot(4,1,4)


        plot(app.TimeData, app.FifthData,Color='y');
        hold on
        plot(app.TimeData, app.SixthData, Color = "b");
        minim = min(min(app.FifthData, app.SixthData));
        maxim = max(max(app.FifthData, app.SixthData));
        ylim([minim-10 maxim+10]);
        xlabel("Tiempo (s)");
        title("Corriente vs PWM");
        legend("Corriente (mA)", "PWM (%)");

    case "Posicion por Corriente (PV)"
        subplot(4,1,1)

        plot(app.TimeData, app.RealValue,Color='g');
        hold on
        plot(app.TimeData, app.GoalValue, Color='r');
        minim = min(min(app.RealValue, app.GoalValue));
        maxim = max(max(app.RealValue, app.GoalValue));
        ylim([minim-10 maxim+10]);
        error=app.equad(app.GoalValue,app.RealValue);
        title("Posicion - Error Cuadrático: "+ num2str(error));
        %title("Posicion");
        ylabel("Posicion (º)");
        xlabel("Tiempo (s)");
        legend("Posicion Real (º)", "Posicion objetivo (º)");

        subplot(4,1,2)

        plot(app.TimeData, app.ThirdData,Color='g');
        hold on
        plot(app.TimeData, app.FourthData,Color='r');
        minim = min(min(app.ThirdData, app.FourthData));
```

```
                maxim = max(max(app.ThirdData, app.FourthData));
                ylim([minim-10 maxim+10]);
                error=app.equad(app.FourthData, app.ThirdData);

                title("Velocidad - Error Cuadrático: "+ num2str(error));
                %title("Velocidad");
                ylabel("Velocidad (RPM)");
                xlabel("Tiempo (s)");
                legend("Velocidad Real (RPM)", "Velocidad objetivo (RPM)");

                subplot(4,1,3)

                accel= app.getAcceleration(app.ThirdData, app.TimeData);
                plot(app.TimeData, accel,Color='g');
                hold on
                accel2 = app.getAcceleration(app.FourthData, app.TimeData);
                plot(app.TimeData, accel2,Color='r');
                minim = min(min(accel, accel2));
                maxim = max(max(accel, accel2));
                ylim([minim-20 maxim+20]);
                error=app.equad(accel2,accel);

                title("Aceleracion - Error Cuadrático: "+ num2str(error));
                ylabel("Aceleración (RPM^2)");
                xlabel("Tiempo (s)");
                legend("Aceleración Real (RPM^2)", "Aceleración objetivo (RPM^2)");

                subplot(4,1,4)

                plot(app.TimeData, app.FifthData,Color='y');
                hold on
                plot(app.TimeData, app.SixthData, Color = "b");
                minim = min(min(app.FifthData, app.SixthData));
                maxim = max(max(app.FifthData, app.SixthData));
                ylim([minim-10 maxim+10]);
                xlabel("Tiempo (s)");
                title("Corriente vs PWM");
                legend("Corriente (mA)", "PWM (%)");
        end

    end

    % Funcion que activa los botones necesarios para su modo y desactiva los demás
    function  EnableButtons(app)

        switch app.Mode
            case 'Posicion'
                app.SendMsgBox(app.Mode); %Indica en que modo se encuentra
                % PWM
                app.DireccionPWMSwitch.Enable= "off";
                app.CicloPWMSpinner.Enable="off";

                %Velocidad
                app.VelocidaddeseadaSpinner.Enable= "off";

                % Posicion
                app.AceleracinTrapezoideEditField.Enable= "on";
                app.VelocidadTrapezoideEditField.Enable= "on";
                app.DKdEditField.Enable= "on";
                app.IKiEditField.Enable= "on";
                app.PKpEditField.Enable= "on";
```

```
            app.PosiciondeseadaEditField.Enable= "on";

            % Posición Corriente
            app.MomentodeInerciaJEditField.Enable= "off";
            app.RozamientoEditField.Enable= "off";
            app.MasaEditField.Enable= "off";

        case "Tension/PWM"
            app.SendMsgBox(app.Mode);
            % PWM
            app.DireccionPWMSwitch.Enable= "on";
            app.CicloPWMSpinner.Enable="on";

            % Velocidad
            app.VelocidaddeseadaSpinner.Enable= "off";

            % Posicion
            app.AceleracinTrapezoideEditField.Enable= "off";
            app.VelocidadTrapezoideEditField.Enable= "off";
            app.DKdEditField.Enable= "off";
            app.IKiEditField.Enable= "off";
            app.PKpEditField.Enable= "off";
            app.PosiciondeseadaEditField.Enable= "off";

            % Posición Corriente
            app.MomentodeInerciaJEditField.Enable= "off";
            app.RozamientoEditField.Enable= "off";
            app.MasaEditField.Enable= "off";

        case "Velocidad"
            app.SendMsgBox(app.Mode);
            % PWM
            app.DireccionPWMSwitch.Enable= "off";
            app.CicloPWMSpinner.Enable="off";

            % Velocidad
            app.VelocidaddeseadaSpinner.Enable= "on";

            % Posicion
            app.AceleracinTrapezoideEditField.Enable= "on";
            app.VelocidadTrapezoideEditField.Enable= "off";
            app.DKdEditField.Enable= "off";
            app.IKiEditField.Enable= "on";
            app.PKpEditField.Enable= "on";
            app.PosiciondeseadaEditField.Enable= "off";

            % Posición Corriente
            app.MomentodeInerciaJEditField.Enable= "off";
            app.RozamientoEditField.Enable= "off";
            app.MasaEditField.Enable= "off";

        case "Velocidad por Tension"
            app.SendMsgBox(app.Mode);
            %PWM
            app.DireccionPWMSwitch.Enable= "off";
            app.CicloPWMSpinner.Enable="off";

            %Velocidad
            app.VelocidaddeseadaSpinner.Enable= "on";
```

```
            %Posicion
            app.AceleracinTrapezoideEditField.Enable= "on";
            app.VelocidadTrapezoideEditField.Enable= "off";
            app.DKdEditField.Enable= "on";
            app.IKiEditField.Enable= "on";
            app.PKpEditField.Enable= "on";
            app.PosiciondeseadaEditField.Enable= "off";

            %Corriente
            app.MomentodeInerciaJEditField.Enable= "off";
            app.RozamientoEditField.Enable= "off";
            app.MasaEditField.Enable= "off";

        case "Velocidad por Corriente"
            app.SendMsgBox(app.Mode);
            %PWM
            app.DireccionPWMSwitch.Enable= "off";
            app.CicloPWMSpinner.Enable="off";

            %Velocidad
            app.VelocidaddeseadaSpinner.Enable= "on";

            %Posicion
            app.AceleracinTrapezoideEditField.Enable= "on";
            app.VelocidadTrapezoideEditField.Enable= "off";
            app.DKdEditField.Enable= "on";
            app.IKiEditField.Enable= "on";
            app.PKpEditField.Enable= "on";
            app.PosiciondeseadaEditField.Enable= "off";

            %Posición Corriente

            app.MomentodeInerciaJEditField.Enable= "off";
            app.RozamientoEditField.Enable= "off";
            app.MasaEditField.Enable= "off";

        case "Posicion por Corriente"
            app.SendMsgBox(app.Mode);
            %PWM
            app.DireccionPWMSwitch.Enable= "off";
            app.CicloPWMSpinner.Enable="off";

            %Velocidad
            app.VelocidaddeseadaSpinner.Enable= "off";

            %Posicion
            app.AceleracinTrapezoideEditField.Enable= "on";
            app.VelocidadTrapezoideEditField.Enable= "on";
            app.DKdEditField.Enable= "on";
            app.IKiEditField.Enable= "on";
            app.PKpEditField.Enable= "on";
            app.PosiciondeseadaEditField.Enable= "on";

            %Posición Corriente

            app.MomentodeInerciaJEditField.Enable= "on";
            app.RozamientoEditField.Enable= "on";
            app.MasaEditField.Enable= "on";

        case "Posicion por Corriente (PV)"
            app.SendMsgBox(app.Mode);
```

```
            %PWM
            app.DireccionPWMSwitch.Enable= "off";

            app.CicloPWMSpinner.Enable="off";
            %Velocidad
            app.VelocidaddeseadaSpinner.Enable= "off";

            %Posicion
            app.AceleracinTrapezoideEditField.Enable= "on";
            app.VelocidadTrapezoideEditField.Enable= "on";
            app.DKdEditField.Enable= "on";
            app.IKiEditField.Enable= "off";
            app.PKpEditField.Enable= "on";
            app.PosiciondeseadaEditField.Enable= "on";

            %Posición Corriente

            app.MomentodeInerciaJEditField.Enable= "on";
            app.RozamientoEditField.Enable= "on";
            app.MasaEditField.Enable= "on";

        end

    end

    % Función que calcula la aceleración dada la velocidad
    function results = getAcceleration(app, velocity, time)

        a= diff(velocity);
        b= diff(time/60);
        acceleration = zeros(size(velocity));
        for i= 2 :length(velocity)
            acceleration(i)= a(i-1)/b(i-1);
        end

            % Devuelve el vector de aceleraciones
            results= acceleration;

    end

    % Cálculo del error cuadrático
    function e2 = equad(app, vector_ref, vector_real)
        % Verifica que los vectores tengan la misma longitud
        if length(vector_ref) ~= length(vector_real)
            error('Los vectores deben tener la misma longitud');
        end

        % Calcula el error cuadrático medio
        N = length(vector_ref);
        quadratic_errors = diag((vector_ref-vector_real)'*(vector_ref-vector_real));
        e2 = sum(quadratic_errors) / N;
    end

    % Función que deshabilita todos los botones al principio del
    % programa
    function EnableInitial(app)
        app.ServoConectadoLamp.Color='red';

        %PWM
        app.DireccionPWMSwitch.Enable= "off";
        app.CicloPWMSpinner.Enable="off";
```

```
        %Velocidad
        app.VelocidaddeseadaSpinner.Enable= "off";

        %Posicion
        app.AceleracinTrapezoideEditField.Enable= "off";
        app.VelocidadTrapezoideEditField.Enable= "off";
        app.DKdEditField.Enable= "off";
        app.IKiEditField.Enable= "off";
        app.PKpEditField.Enable= "off";
        app.PosiciondeseadaEditField.Enable= "off";

        % Posición Corriente
        app.RozamientoEditField.Enable="off";
        app.MomentodeInerciaJEditField.Enable="off";
        app.MasaEditField.Enable="off";

        % General
        app.EmpezarButton.Enable="off";
        app.ServoConectadoLampLabel.Text="Servo Desconectado";
        app.PararButton.Enable="off";
        app.CambiarParmetrosButton.Enable="off";
        app.ModosDropDown.Enable="off";

    end
end


% Callbacks that handle component events
methods (Access = private)

    % Code that executes after component creation
    function startupFcn(app)
        app.PuertoSerialDropDown.Items = serialportlist("available");
        app.STOP = false;
        app.DesconectarButton.Enable="off";
        app.EnableInitial();


        app.Mode='Posicion';
        app.PosiciondeseadaEditField.Value= 50;
        app.VelocidadTrapezoideEditField.Value= 1.0;
        app.AceleracinTrapezoideEditField.Value= 0.5;
        app.PKpEditField.Value= 900;
        app.IKiEditField.Value= 100;
        app.DKdEditField.Value= 1000;


    end

    % Button pushed function: DesconectarButton
    function DesconectarButtonPushed(app, event)
        if( ~isempty(app.com))
            arrayToSend = [1, 2, 3, 4, 5,6,7,8,9,10,11,12];
            arrayToSend(1)=9;
            for i = 1:length(arrayToSend)
                fprintf(app.com, '%d\n', arrayToSend(i));
                pause(0.1);
            end
        app.STOP=true;
        pause(0.5);
```

```
        fclose ('all');
        delete(app.com);
        clear app.com;
        app.ConectarServoButton.Enable="on";
        app.EnableInitial();
        end
    end

    % Close request function: UIFigure
    function UIFigureCloseRequest(app, event)

        for i = 1:length(app.MsgHandler)
            if isvalid(app.MsgHandler(i))
                delete(app.MsgHandler(i));
            end
        end
        try
        arrayToSend = [1, 2, 3, 4, 5,6,7,8,9,10,11,12];
        arrayToSend(1)=9;
        for i = 1:length(arrayToSend)
            fprintf(app.com, '%d\n', arrayToSend(i));
            pause(0.1);
        end
        app.STOP=true;
        pause(0.5);
        fclose('all');
        delete(app.com);
        delete(app);
        catch
            delete(app.com);
            delete(app);
        end


    end

    % Button pushed function: ConectarServoButton
    function ConectarServoButtonPushed(app, event)
        if  ~isempty(app.com)
            clear app.com;
        end
        port=app.PuertoSerialDropDown.Value;

        try

        app.ServoConectadoLampLabel.Text="Conectando";
        app.ServoConectadoLamp.Color='yellow';
        app.com = serialport(port, 115200);
            if ~isempty(app.com)

                configureTerminator(app.com,"LF","LF");
                fopen(app.com);
                arrayToSend = [1, 2, 3, 4, 5,6,7,8,9,10,11,12];
                arrayToSend(1)=app.ConnectFunction;


                for i = 1:length(arrayToSend)
                    fprintf(app.com, '%f\n', arrayToSend(i));
                    pause(0.1);
                end
```

```
            response = app.readArrayFromSerial(app.com, app.arraySize);

            app.SendMsgBox(num2str(response));



                if (response(1)==app.ConnectFunction)
                    app.ServoConectadoLamp.Color='green';
                    app.ServoConectadoLampLabel.Text="Servo Conectado";
                    app.EnableButtons();
                    app.EmpezarButton.Enable='on';
                    app.DesconectarButton.Enable="on";
                    app.ConectarServoButton.Enable="off";
                    app.PararButton.Enable="on";

                    app.ModosDropDown.Enable="on";
                end


        end
    catch
        app.SendMsgBox("Error en comunicarse con el serial: "+port);
        clear app.com;
        clear port;

        app.ServoConectadoLampLabel.Text="Error en la conexión";
        app.ServoConectadoLamp.Color='red';
    end


end

% Drop down opening function: PuertoSerialDropDown
function PuertoSerialDropDownOpening(app, event)
    %app.PuertoSerialDropDown.Items = serialportlist("available");
end

% Button pushed function: PararButton
function PararButtonPushed(app, event)
    app.EmpezarButton.Enable="on";
    app.CambiarParmetrosButton.Enable="off";
    app.ModosDropDown.Enable="on";
    arrayToSend = [1, 2, 3, 4, 5,6,7,8,9,10,11,12];
    arrayToSend(1)=9;
    for i = 1:length(arrayToSend)
        fprintf(app.com, '%d\n', arrayToSend(i));
        pause(0.1);
    end
    app.STOP= true;
    app.SendMsgBox("Parar");
    app.EnableButtons();


end

% Value changed function: ModosDropDown
function ModosDropDownValueChanged(app, event)
    app.Mode= app.ModosDropDown.Value;
    app.EnableButtons();
    switch app.Mode
        case 'Posicion'
```

```
        app.PosiciondeseadaEditField.Value= 50;
        app.VelocidadTrapezoideEditField.Value= 1.0;
        app.AceleracinTrapezoideEditField.Value= 0.5;
        app.AceleracinTrapezoideEditFieldLabel.Text= 'Aceleración Trapezoide';
        app.PKpEditField.Value= 900;
        app.IKiEditField.Value= 100;
        app.DKdEditField.Value= 1000;
        app.DKdEditFieldLabel.Text= ' D (Kd)';


    case "Velocidad"
        app.VelocidaddeseadaSpinner.Value= 1.0;
        app.AceleracinTrapezoideEditField.Value= 0.5;
        app.AceleracinTrapezoideEditFieldLabel.Text= 'Aceleración';
        app.PKpEditField.Value= 10;
        app.IKiEditField.Value= 10;

    case "Tension/PWM"
        app.VelocidaddeseadaSpinner.Value= 5.0;

    case "Velocidad por Tension"
        app.VelocidaddeseadaSpinner.Value= 1.0;
        app.AceleracinTrapezoideEditField.Value= 0.5;
        app.AceleracinTrapezoideEditFieldLabel.Text= 'Aceleración';
        app.DKdEditFieldLabel.Text= ' D (Kd)';
        app.PKpEditField.Value= 10;
        app.IKiEditField.Value= 5;
        app.DKdEditField.Value= 10;

    case "Velocidad por Corriente"
        app.VelocidaddeseadaSpinner.Value= 1.0;
        app.AceleracinTrapezoideEditField.Value= 0.5;
        app.AceleracinTrapezoideEditFieldLabel.Text= 'Aceleración';
        app.DKdEditFieldLabel.Text= ' D (Kd)';
        app.PKpEditField.Value= 10;
        app.IKiEditField.Value= 5;
        app.DKdEditField.Value= 10;

    case "Posicion por Corriente"
        app.PosiciondeseadaEditField.Value= 150;
        app.VelocidadTrapezoideEditField.Value= 1.0;
        app.AceleracinTrapezoideEditField.Value= 0.5;
        app.AceleracinTrapezoideEditFieldLabel.Text= 'Aceleración Trapezoide';
        app.PKpEditField.Value= 8;
        app.IKiEditField.Value= 1;
        app.DKdEditField.Value= 20;
        app.DKdEditFieldLabel.Text= ' V (Kv)';
        app.MasaEditField.Value = 0.171;
        app.MomentodeInerciaJEditField.Value = 0.00063597;
        app.RozamientoEditField.Value = 0.00566;

      case "Posicion por Corriente (PV)"
        app.PosiciondeseadaEditField.Value= 150;
        app.VelocidadTrapezoideEditField.Value= 1.0;
        app.AceleracinTrapezoideEditField.Value= 0.5;
        app.AceleracinTrapezoideEditFieldLabel.Text= 'Aceleración Trapezoide';
        app.PKpEditField.Value= 1.7;
        app.IKiEditField.Value= 0;
        app.DKdEditField.Value= 15;
        app.DKdEditFieldLabel.Text= ' V (Kv)';
        app.MasaEditField.Value = 0.171;
```

```
            app.MomentodeInerciaJEditField.Value = 0.00063597;
            app.RozamientoEditField.Value = 0.00566;

        end
    end

    % Button pushed function: EmpezarButton
    function EmpezarButtonPushed(app, event)
        try
        app.EmpezarButton.Enable="off";
        app.CambiarParmetrosButton.Enable="on";
        app.ModosDropDown.Enable="off";
        switch app.Mode
            case 'Posicion'

            app.ValorRealLabel.Text= 'Posición Real (º)';
            app.ValorObjetivoLabel.Text= 'Posición Objetivo (º)';
            app.Valor3Label.Text= 'Velocidad Real (RPM)';
            app.Valor4Label.Text= 'Velocidad Objetivo (RPM)';


            flush(app.com);
            arrayToSend = [app.OP_EXTENDED_POSITION, 2, 3, 4, 5, 6,7,8,9,10,11,12];
            arrayToSend(2) = app.PosiciondeseadaEditField.Value;
            arrayToSend(3) = app.VelocidadTrapezoideEditField.Value;
            arrayToSend(4) = app.AceleracinTrapezoideEditField.Value;
            arrayToSend(5) = app.PKpEditField.Value;
            arrayToSend(6) = app.IKiEditField.Value;
            arrayToSend(7) = app.DKdEditField.Value;

            arrayToSend(12) = 0;
            app.AceleracinTrapezoideEditField.Enable= "off";
            app.VelocidadTrapezoideEditField.Enable= "off";

            max=400;
            if (arrayToSend(2)>400)
                max=arrayToSend(2)+20;
            end
            app.UIAxes.YLim = [0 max];



            for i = 1:length(arrayToSend)
                fprintf(app.com, '%f\n', arrayToSend(i));
                pause(0.1);
            end

            response = app.readArrayFromSerial(app.com, app.arraySize);
            app.SendMsgBox(num2str(response));

            pause(0.5);

            app.acquireData();
        case "Tension/PWM"
            app.ValorRealLabel.Text= 'PWM Real (%)';
            app.ValorObjetivoLabel.Text= 'PWM Objetivo (%)';
            app.Valor3Label.Text= 'Corriente (mA)';
            app.Valor4Label.Text= 'Velocidad (RPM)';

            flush(app.com);
            arrayToSend = [app.OP_PWM, 2, 3, 4, 5, 6,7,8,9,10,11,12];
```

```
            arrayToSend(2)= app.CicloPWMSpinner.Value;
            arrayToSend(3)=0;
            if(app.DireccionPWMSwitch.Value == "Direccion 2")
                arrayToSend(3)= 1;
            end

            arrayToSend(12) = 0;


            app.UIAxes.YLim = [0 120];

            for i = 1:length(arrayToSend)
                fprintf(app.com, '%f\n', arrayToSend(i));
                pause(0.1);
            end

           response = app.readArrayFromSerial(app.com, app.arraySize);
            app.SendMsgBox(num2str(response));

            pause(0.5);

            app.acquireData();

        case "Velocidad"

            app.ValorRealLabel.Text= 'Velocidad Real (RPM)';
            app.ValorObjetivoLabel.Text= 'Velocidad Objetivo (RPM)';
            app.Valor3Label.Text= 'Corriente (mA)';
            app.Valor4Label.Text= 'PWM  (%)';

            flush(app.com);
            arrayToSend = [app.OP_VELOCITY, 2, 3, 4, 5, 6,7,8,9,10,11,12];
             arrayToSend(2)=0;
            arrayToSend(3)= app.VelocidaddeseadaSpinner.Value;
            arrayToSend(4)=app.AceleracinTrapezoideEditField.Value;
            arrayToSend(5)=app.PKpEditField.Value;
            arrayToSend(6)=app.IKiEditField.Value;
            app.MomentodeInerciaJEditField.Enable= "off";


            arrayToSend(12) = 0;


            app.UIAxes.YLim = [0 200];

            for i = 1:length(arrayToSend)
                fprintf(app.com, '%f\n', arrayToSend(i));
                pause(0.1);
            end

            response = app.readArrayFromSerial(app.com, app.arraySize);
            app.SendMsgBox(num2str(response));

            pause(0.5);

            app.acquireData();
        case "Velocidad por Tension"

            app.ValorRealLabel.Text= 'Velocidad Real (RPM)';
            app.ValorObjetivoLabel.Text= 'Velocidad Objetivo (RPM)';
            app.Valor3Label.Text= 'PWM real (%)';
```

```
            app.Valor4Label.Text= 'PWM Objetivo (%)';

            flush(app.com);
            arrayToSend = [app.OP_PWM_VELOCITY, 2, 3, 4, 5, 6,7,8,9,10,11,12];
            arrayToSend(2)=0;
            arrayToSend(3)= app.VelocidaddeseadaSpinner.Value;
            arrayToSend(4)=app.AceleracinTrapezoideEditField.Value;
            arrayToSend(5)=app.PKpEditField.Value;
            arrayToSend(6)=app.IKiEditField.Value;
            arrayToSend(7) = app.DKdEditField.Value;
            app.MomentodeInerciaJEditField.Enable= "off";


            arrayToSend(12) = 0;


            app.UIAxes.YLim = [0 200];

            for i = 1:length(arrayToSend)
                fprintf(app.com, '%f\n', arrayToSend(i));
                pause(0.1);
            end

           response = app.readArrayFromSerial(app.com, app.arraySize);
            app.SendMsgBox(num2str(response));

            pause(0.5);

            app.acquireData();
        case "Velocidad por Corriente"

            app.ValorRealLabel.Text= 'Velocidad Real (RPM)';
            app.ValorObjetivoLabel.Text= 'Velocidad Objetivo (RPM)';
            app.Valor3Label.Text= 'Corriente real (mA)';
            app.Valor4Label.Text= 'Corriente objetivo (mA)';

            flush(app.com);
            arrayToSend = [app.OP_CURRENT, 2, 3, 4, 5, 6,7,8,9,10,11,12];
            arrayToSend(2)=0;
            arrayToSend(3)= app.VelocidaddeseadaSpinner.Value;
            arrayToSend(4)=app.AceleracinTrapezoideEditField.Value;
            arrayToSend(5)=app.PKpEditField.Value;
            arrayToSend(6)=app.IKiEditField.Value;
            arrayToSend(7) = app.DKdEditField.Value;
            app.MomentodeInerciaJEditField.Enable= "off";


            arrayToSend(12) = 0;


            app.UIAxes.YLim = [0 200];

            for i = 1:length(arrayToSend)
                fprintf(app.com, '%f\n', arrayToSend(i));
                pause(0.1);
            end

            response = app.readArrayFromSerial(app.com, app.arraySize);
            app.SendMsgBox(num2str(response));

            pause(0.5);
```

```
                app.acquireData();
            case "Posicion por Corriente"

                app.ValorRealLabel.Text= 'Posición Real (º)';
                app.ValorObjetivoLabel.Text= 'Posición Objetivo (º)';
                app.Valor3Label.Text= 'Velocidad Real (RPM)';
                app.Valor4Label.Text= 'Velocidad Objetivo (RPM)';

                flush(app.com);
                arrayToSend = [app.OP_CURRENT_BASED_POSITION, 2, 3, 4, 5,
6,7,8,9,10,11,12];
                arrayToSend(2) = app.PosiciondeseadaEditField.Value;
                arrayToSend(3) = app.VelocidadTrapezoideEditField.Value;
                arrayToSend(4) = app.AceleracinTrapezoideEditField.Value;
                arrayToSend(5) = app.PKpEditField.Value;
                arrayToSend(6) = app.IKiEditField.Value;
                arrayToSend(7) = app.DKdEditField.Value;

                %arrayToSend(8) = app.CorrientemximamAEditField.Value;
                %En caso de querer enviar una corriente máxima se puede
                %hablitar este array y crear la herramienta
                arrayToSend(9) = app.RozamientoEditField.Value;
                arrayToSend(10) = app.MomentodeInerciaJEditField.Value;
                arrayToSend(11) = app.MasaEditField.Value;
                arrayToSend(12) = 0;
                app.AceleracinTrapezoideEditField.Enable= "off";
                app.VelocidadTrapezoideEditField.Enable= "off";

                max=400;
                if (arrayToSend(2)>400)
                    max=arrayToSend(2)+20;
                end
                app.UIAxes.YLim = [0 max];

                for i = 1:length(arrayToSend)
                    fprintf(app.com, '%f\n', arrayToSend(i));
                    pause(0.1);
                end

                response = app.readArrayFromSerial(app.com, app.arraySize);
                app.SendMsgBox(num2str(response));

                pause(0.5);

                app.acquireData();
            case "Posicion por Corriente (PV)"

                app.ValorRealLabel.Text= 'Posición Real (º)';
                app.ValorObjetivoLabel.Text= 'Posición Objetivo (º)';
                app.Valor3Label.Text= 'Velocidad Real (RPM)';
                app.Valor4Label.Text= 'Velocidad Objetivo (RPM)';

                flush(app.com);
                arrayToSend = [app.OP_CURRENT_BASED_POSITION_PV, 2, 3, 4, 5,
6,7,8,9,10,11,12];
                arrayToSend(2) = app.PosiciondeseadaEditField.Value;
                arrayToSend(3) = app.VelocidadTrapezoideEditField.Value;
                arrayToSend(4) = app.AceleracinTrapezoideEditField.Value;
                arrayToSend(5) = app.PKpEditField.Value;
                arrayToSend(6) = 0;
```

```
                arrayToSend(7) = app.DKdEditField.Value;

                %arrayToSend(8) = app.CorrientemximamAEditField.Value;
                arrayToSend(9) = app.RozamientoEditField.Value;
                arrayToSend(10) = app.MomentodeInerciaJEditField.Value;
                arrayToSend(11) = app.MasaEditField.Value;
                arrayToSend(12) = 0;
                app.AceleracinTrapezoideEditField.Enable= "off";
                app.VelocidadTrapezoideEditField.Enable= "off";

                max=400;
                if (arrayToSend(2)>400)
                    max=arrayToSend(2)+20;
                end
                app.UIAxes.YLim = [0 max];

                for i = 1:length(arrayToSend)
                    fprintf(app.com, '%f\n', arrayToSend(i));
                    pause(0.1);
                end

               response = app.readArrayFromSerial(app.com, app.arraySize);
                app.SendMsgBox(num2str(response));

                pause(0.5);

                app.acquireData();
        end
        catch
            app.EmpezarButton.Enable="on";
            app.SendMsgBox("Error al iniciar trayectoria");
        end

    end

    % Button pushed function: CambiarParmetrosButton
    function CambiarParmetrosButtonPushed(app, event)


         switch app.Mode
            case 'Posicion'

                pause(0.2);
                arrayToSend = [app.OP_EXTENDED_POSITION, 2, 3, 4, 5, 6,7,8,9,10,11,12];
                arrayToSend(2) = app.PosiciondeseadaEditField.Value;
                arrayToSend(3) = app.VelocidadTrapezoideEditField.Value;
                arrayToSend(4) = app.AceleracinTrapezoideEditField.Value;
                arrayToSend(5) = app.PKpEditField.Value;
                arrayToSend(6) = app.IKiEditField.Value;
                arrayToSend(7) = app.DKdEditField.Value;
                arrayToSend(12)=1;

                max=400;
                if (arrayToSend(2)>400)
                    max=arrayToSend(2)+20;
                end
                app.UIAxes.YLim = [0 max];

             case "Tension/PWM"
                flush(app.com, "input");
                pause(0.2);
```

```
                    arrayToSend = [app.OP_PWM, 2, 3, 4, 5, 6,7,8,9,10,11,12];
                    arrayToSend(2) = app.CicloPWMSpinner.Value;
                    arrayToSend(3)=0;
                    if(app.DireccionPWMSwitch.Value == "Direccion 2")
                        arrayToSend(3)= 1;
                    end

                    arrayToSend(12)=1;

                case "Velocidad"
                    flush(app.com, "input");
                    pause(0.2);
                    arrayToSend = [app.OP_VELOCITY, 2, 3, 4, 5, 6,7,8,9,10,11,12];
                    arrayToSend(3)= app.VelocidaddeseadaSpinner.Value;
                    arrayToSend(4)=app.AceleracinTrapezoideEditField.Value;
                    arrayToSend(5)=app.PKpEditField.Value;
                    arrayToSend(6)=app.IKiEditField.Value;


                    arrayToSend(12)=1;

                 case "Velocidad por Tension"
                    flush(app.com, "input");
                    pause(0.2);
                    arrayToSend = [app.OP_PWM_VELOCITY, 2, 3, 4, 5, 6,7,8,9,10,11,12];
                    arrayToSend(2) = 0;
                    arrayToSend(3)= app.VelocidaddeseadaSpinner.Value;
                    arrayToSend(4)=app.AceleracinTrapezoideEditField.Value;
                    arrayToSend(5)=app.PKpEditField.Value;
                    arrayToSend(6)=app.IKiEditField.Value;
                    arrayToSend(7) = app.DKdEditField.Value;


                    arrayToSend(12)=1;

                case "Velocidad por Corriente"
                    flush(app.com, "input");
                    pause(0.2);
                    arrayToSend = [app.OP_CURRENT, 2, 3, 4, 5, 6,7,8,9,10,11,12];
                    arrayToSend(2) = 0;
                    arrayToSend(3)= app.VelocidaddeseadaSpinner.Value;
                    arrayToSend(4)=app.AceleracinTrapezoideEditField.Value;
                    arrayToSend(5)=app.PKpEditField.Value;
                    arrayToSend(6)=app.IKiEditField.Value;
                    arrayToSend(7) = app.DKdEditField.Value;


                    arrayToSend(12)=1;

                case "Posicion por Corriente"
                    flush(app.com, "input");
                    pause(0.2);
                    arrayToSend = [app.OP_CURRENT_BASED_POSITION, 2, 3, 4, 5,
6,7,8,9,10,11,12];
                    arrayToSend(2) = app.PosiciondeseadaEditField.Value;
                    arrayToSend(3) = app.VelocidadTrapezoideEditField.Value;
                    arrayToSend(4) = app.AceleracinTrapezoideEditField.Value;
                    arrayToSend(5) = app.PKpEditField.Value;
                    arrayToSend(6) = app.IKiEditField.Value;
                    arrayToSend(7) = app.DKdEditField.Value;
                    % En el caso de necesitar limitar la corriente máxima
```

```
                    %arrayToSend(8) = app.CorrientemximamAEditField.Value;
                    arrayToSend(9) = app.RozamientoEditField.Value;
                    arrayToSend(10) = app.MomentodeInerciaJEditField.Value;
                    arrayToSend(11) = app.MasaEditField.Value;
                    arrayToSend(12) = 1;

                    max=400;
                    if (arrayToSend(2)>400)
                        max=arrayToSend(2)+20;
                    end
                    app.UIAxes.YLim = [0 max];

                case "Posicion por Corriente (PV)"
                    flush(app.com, "input");
                    pause(0.2);
                    arrayToSend = [app.OP_CURRENT_BASED_POSITION_PV, 2, 3, 4, 5,
6,7,8,9,10,11,12];
                    arrayToSend(2) = app.PosiciondeseadaEditField.Value;
                    arrayToSend(3) = app.VelocidadTrapezoideEditField.Value;
                    arrayToSend(4) = app.AceleracinTrapezoideEditField.Value;
                    arrayToSend(5) = app.PKpEditField.Value;
                    arrayToSend(6) = 0;
                    arrayToSend(7) = app.DKdEditField.Value;
                    % En el caso de necesitar limitar la corriente máxima
                    %arrayToSend(8) = app.CorrientemximamAEditField.Value;
                    arrayToSend(9) = app.RozamientoEditField.Value;
                    arrayToSend(10) = app.MomentodeInerciaJEditField.Value;
                    arrayToSend(11) = app.MasaEditField.Value;
                    arrayToSend(12) = 1;

                    max=400;
                    if (arrayToSend(2)>400)
                        max=arrayToSend(2)+20;
                    end
                    app.UIAxes.YLim = [0 max];

            end
            flush(app.com, "input");
               for i = 1:length(arrayToSend)
                    fprintf(app.com, '%f\n', arrayToSend(i));
                    pause(0.1);
               end
               flush(app.com, "input");

            app.SendMsgBox('Enviado nuevo parámetro ');
        end

        % Button pushed function: RefrescarlistapuertosButton
        function RefrescarlistapuertosButtonPushed(app, event)
            app.PuertoSerialDropDown.Items = serialportlist("available");
        end

        % Value changed function: DireccionPWMSwitch
        function DireccionPWMSwitchValueChanged(app, event)
            app.Direction = ~ app.Direction;


        end
    end

    % Component initialization
```

```
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 919 699];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest,
true);

            % Create UIAxes
            app.UIAxes = uiaxes(app.UIFigure);
            title(app.UIAxes, 'Grafica Trayectoria')
            xlabel(app.UIAxes, 'Pasos')
            ylabel(app.UIAxes, 'Valor')
            zlabel(app.UIAxes, 'Z')
            app.UIAxes.YGrid = 'on';
            app.UIAxes.Position = [9 158 583 474];

            % Create DesconectarButton
            app.DesconectarButton = uibutton(app.UIFigure, 'push');
            app.DesconectarButton.ButtonPushedFcn = createCallbackFcn(app,
@DesconectarButtonPushed, true);
            app.DesconectarButton.Position = [35 631 100 23];
            app.DesconectarButton.Text = 'Desconectar';

            % Create ServoConectadoLampLabel
            app.ServoConectadoLampLabel = uilabel(app.UIFigure);
            app.ServoConectadoLampLabel.HorizontalAlignment = 'right';
            app.ServoConectadoLampLabel.Position = [382 650 136 22];
            app.ServoConectadoLampLabel.Text = 'Servo Conectado';

            % Create ServoConectadoLamp
            app.ServoConectadoLamp = uilamp(app.UIFigure);
            app.ServoConectadoLamp.Position = [527 649 25 25];

            % Create ConectarServoButton
            app.ConectarServoButton = uibutton(app.UIFigure, 'push');
            app.ConectarServoButton.ButtonPushedFcn = createCallbackFcn(app,
@ConectarServoButtonPushed, true);
            app.ConectarServoButton.Position = [35 663 100 23];
            app.ConectarServoButton.Text = 'Conectar Servo';

            % Create PuertoSerialDropDown
            app.PuertoSerialDropDown = uidropdown(app.UIFigure);
            app.PuertoSerialDropDown.DropDownOpeningFcn = createCallbackFcn(app,
@PuertoSerialDropDownOpening, true);
            app.PuertoSerialDropDown.Position = [246 636 100 22];

            % Create PuertoSerialDropDownLabel
            app.PuertoSerialDropDownLabel = uilabel(app.UIFigure);
            app.PuertoSerialDropDownLabel.HorizontalAlignment = 'right';
            app.PuertoSerialDropDownLabel.Position = [157 636 74 22];
            app.PuertoSerialDropDownLabel.Text = 'Puerto Serial';

            % Create PararButton
            app.PararButton = uibutton(app.UIFigure, 'push');
            app.PararButton.ButtonPushedFcn = createCallbackFcn(app, @PararButtonPushed,
true);
```

```
            app.ParарButton.Position = [820 128 100 23];
            app.ParарButton.Text = 'Parar';

            % Create PosiciondeseadaEditFieldLabel
            app.PosiciondeseadaEditFieldLabel = uilabel(app.UIFigure);
            app.PosiciondeseadaEditFieldLabel.HorizontalAlignment = 'right';
            app.PosiciondeseadaEditFieldLabel.Position = [628 589 100 22];
            app.PosiciondeseadaEditFieldLabel.Text = 'Posicion deseada';

            % Create PosiciondeseadaEditField
            app.PosiciondeseadaEditField = uieditfield(app.UIFigure, 'numeric');
            app.PosiciondeseadaEditField.Position = [734 589 100 22];

            % Create ModosDropDownLabel
            app.ModosDropDownLabel = uilabel(app.UIFigure);
            app.ModosDropDownLabel.HorizontalAlignment = 'right';
            app.ModosDropDownLabel.Position = [674 650 41 22];
            app.ModosDropDownLabel.Text = 'Modos';

            % Create ModosDropDown
            app.ModosDropDown = uidropdown(app.UIFigure);
            app.ModosDropDown.Items = {'Posicion', 'Tension/PWM', 'Velocidad', 'Velocidad por
Tension', 'Velocidad por Corriente', 'Posicion por Corriente', 'Posicion por Corriente (PV)'};
            app.ModosDropDown.ValueChangedFcn = createCallbackFcn(app,
@ModosDropDownValueChanged, true);
            app.ModosDropDown.Position = [730 650 100 22];
            app.ModosDropDown.Value = 'Posicion';

            % Create PKpEditFieldLabel
            app.PKpEditFieldLabel = uilabel(app.UIFigure);
            app.PKpEditFieldLabel.HorizontalAlignment = 'right';
            app.PKpEditFieldLabel.Position = [687 495 42 22];
            app.PKpEditFieldLabel.Text = ' P (Kp)';

            % Create PKpEditField
            app.PKpEditField = uieditfield(app.UIFigure, 'numeric');
            app.PKpEditField.Position = [735 495 100 22];

            % Create IKiEditFieldLabel
            app.IKiEditFieldLabel = uilabel(app.UIFigure);
            app.IKiEditFieldLabel.HorizontalAlignment = 'right';
            app.IKiEditFieldLabel.Position = [695 462 34 22];
            app.IKiEditFieldLabel.Text = ' I (Ki)';

            % Create IKiEditField
            app.IKiEditField = uieditfield(app.UIFigure, 'numeric');
            app.IKiEditField.Position = [735 462 100 22];

            % Create DKdEditFieldLabel
            app.DKdEditFieldLabel = uilabel(app.UIFigure);
            app.DKdEditFieldLabel.HorizontalAlignment = 'right';
            app.DKdEditFieldLabel.Position = [689 430 40 22];
            app.DKdEditFieldLabel.Text = 'D (Kd)';

            % Create DKdEditField
            app.DKdEditField = uieditfield(app.UIFigure, 'numeric');
            app.DKdEditField.Position = [735 431 100 22];

            % Create VelocidadTrapezoideEditFieldLabel
            app.VelocidadTrapezoideEditFieldLabel = uilabel(app.UIFigure);
            app.VelocidadTrapezoideEditFieldLabel.HorizontalAlignment = 'right';
```

```
            app.VelocidadTrapezoideEditFieldLabel.Position = [609 556 120 22];
            app.VelocidadTrapezoideEditFieldLabel.Text = 'Velocidad Trapezoide';

            % Create VelocidadTrapezoideEditField
            app.VelocidadTrapezoideEditField = uieditfield(app.UIFigure, 'numeric');
            app.VelocidadTrapezoideEditField.Position = [735 556 100 22];

            % Create AceleracinTrapezoideEditFieldLabel
            app.AceleracinTrapezoideEditFieldLabel = uilabel(app.UIFigure);
            app.AceleracinTrapezoideEditFieldLabel.HorizontalAlignment = 'right';
            app.AceleracinTrapezoideEditFieldLabel.Position = [599 524 130 22];
            app.AceleracinTrapezoideEditFieldLabel.Text = 'Aceleración Trapezoide';

            % Create AceleracinTrapezoideEditField
            app.AceleracinTrapezoideEditField = uieditfield(app.UIFigure, 'numeric');
            app.AceleracinTrapezoideEditField.Position = [735 524 100 22];

            % Create VelocidaddeseadaSpinnerLabel
            app.VelocidaddeseadaSpinnerLabel = uilabel(app.UIFigure);
            app.VelocidaddeseadaSpinnerLabel.HorizontalAlignment = 'right';
            app.VelocidaddeseadaSpinnerLabel.Position = [614 276 106 22];
            app.VelocidaddeseadaSpinnerLabel.Text = 'Velocidad deseada';

            % Create VelocidaddeseadaSpinner
            app.VelocidaddeseadaSpinner = uispinner(app.UIFigure);
            app.VelocidaddeseadaSpinner.Limits = [1 100];
            app.VelocidaddeseadaSpinner.Position = [735 276 100 22];
            app.VelocidaddeseadaSpinner.Value = 1;

            % Create EmpezarButton
            app.EmpezarButton = uibutton(app.UIFigure, 'push');
            app.EmpezarButton.ButtonPushedFcn = createCallbackFcn(app, @EmpezarButtonPushed,
true);
            app.EmpezarButton.Position = [585 128 100 23];
            app.EmpezarButton.Text = 'Empezar';

            % Create DireccionPWMSwitch_2Label
            app.DireccionPWMSwitch_2Label = uilabel(app.UIFigure);
            app.DireccionPWMSwitch_2Label.HorizontalAlignment = 'center';
            app.DireccionPWMSwitch_2Label.Position = [716 314 88 22];
            app.DireccionPWMSwitch_2Label.Text = 'Direccion PWM';

            % Create DireccionPWMSwitch
            app.DireccionPWMSwitch = uiswitch(app.UIFigure, 'slider');
            app.DireccionPWMSwitch.Items = {'Direccion 1', 'Direccion 2'};
            app.DireccionPWMSwitch.ValueChangedFcn = createCallbackFcn(app,
@DireccionPWMSwitchValueChanged, true);
            app.DireccionPWMSwitch.Position = [740 341 40 17];
            app.DireccionPWMSwitch.Value = 'Direccion 1';

            % Create CicloPWMSpinnerLabel
            app.CicloPWMSpinnerLabel = uilabel(app.UIFigure);
            app.CicloPWMSpinnerLabel.HorizontalAlignment = 'right';
            app.CicloPWMSpinnerLabel.Position = [629 374 90 22];
            app.CicloPWMSpinnerLabel.Text = 'Ciclo PWM  (%)';

            % Create CicloPWMSpinner
            app.CicloPWMSpinner = uispinner(app.UIFigure);
            app.CicloPWMSpinner.Limits = [1 100];
            app.CicloPWMSpinner.Position = [734 374 100 22];
            app.CicloPWMSpinner.Value = 1;
```

```
            % Create CambiarParmetrosButton
            app.CambiarParmetrosButton = uibutton(app.UIFigure, 'push');
            app.CambiarParmetrosButton.ButtonPushedFcn = createCallbackFcn(app,
@CambiarParmetrosButtonPushed, true);
            app.CambiarParmetrosButton.Position = [689 128 126 23];
            app.CambiarParmetrosButton.Text = 'Cambiar Parámetros';

            % Create RefrescarlistapuertosButton
            app.RefrescarlistapuertosButton = uibutton(app.UIFigure, 'push');
            app.RefrescarlistapuertosButton.ButtonPushedFcn = createCallbackFcn(app,
@RefrescarlistapuertosButtonPushed, true);
            app.RefrescarlistapuertosButton.Position = [185 665 135 25];
            app.RefrescarlistapuertosButton.Text = 'Refrescar lista puertos';

            % Create RozamientoEditFieldLabel
            app.RozamientoEditFieldLabel = uilabel(app.UIFigure);
            app.RozamientoEditFieldLabel.HorizontalAlignment = 'right';
            app.RozamientoEditFieldLabel.Position = [653 247 67 22];
            app.RozamientoEditFieldLabel.Text = 'Rozamiento';

            % Create RozamientoEditField
            app.RozamientoEditField = uieditfield(app.UIFigure, 'numeric');
            app.RozamientoEditField.Position = [737 247 100 22];

            % Create MomentodeInerciaJEditFieldLabel
            app.MomentodeInerciaJEditFieldLabel = uilabel(app.UIFigure);
            app.MomentodeInerciaJEditFieldLabel.HorizontalAlignment = 'right';
            app.MomentodeInerciaJEditFieldLabel.Position = [592 218 128 22];
            app.MomentodeInerciaJEditFieldLabel.Text = 'Momento de Inercia (J)';

            % Create MomentodeInerciaJEditField
            app.MomentodeInerciaJEditField = uieditfield(app.UIFigure, 'numeric');
            app.MomentodeInerciaJEditField.Position = [737 218 100 22];

            % Create MasaEditFieldLabel
            app.MasaEditFieldLabel = uilabel(app.UIFigure);
            app.MasaEditFieldLabel.HorizontalAlignment = 'right';
            app.MasaEditFieldLabel.Position = [684 177 34 22];
            app.MasaEditFieldLabel.Text = 'Masa';

            % Create MasaEditField
            app.MasaEditField = uieditfield(app.UIFigure, 'numeric');
            app.MasaEditField.Position = [735 177 100 22];

            % Create GradosLabel
            app.GradosLabel = uilabel(app.UIFigure);
            app.GradosLabel.Interpreter = 'tex';
            app.GradosLabel.Position = [851 589 69 22];
            app.GradosLabel.Text = 'º (Grados)';

            % Create radsLabel
            app.radsLabel = uilabel(app.UIFigure);
            app.radsLabel.Interpreter = 'tex';
            app.radsLabel.Position = [857 555 35 22];
            app.radsLabel.Text = 'rad/s';

            % Create rads2Label
            app.rads2Label = uilabel(app.UIFigure);
            app.rads2Label.Interpreter = 'tex';
            app.rads2Label.Position = [857 524 40 22];
```

```
app.rads2Label.Text = 'rad/s^2';

% Create Label
app.Label = uilabel(app.UIFigure);
app.Label.Interpreter = 'tex';
app.Label.Position = [857 371 35 22];
app.Label.Text = '%';

% Create radsLabel_2
app.radsLabel_2 = uilabel(app.UIFigure);
app.radsLabel_2.Interpreter = 'tex';
app.radsLabel_2.Position = [857 275 35 22];
app.radsLabel_2.Text = 'rad/s';

% Create NmLabel
app.NmLabel = uilabel(app.UIFigure);
app.NmLabel.Interpreter = 'tex';
app.NmLabel.Position = [857 249 33 22];
app.NmLabel.Text = 'N/m';

% Create Kgm2Label
app.Kgm2Label = uilabel(app.UIFigure);
app.Kgm2Label.Interpreter = 'tex';
app.Kgm2Label.Position = [857 218 48 22];
app.Kgm2Label.Text = 'Kg*m^2';

% Create KgLabel
app.KgLabel = uilabel(app.UIFigure);
app.KgLabel.Interpreter = 'tex';
app.KgLabel.Position = [860 177 25 22];
app.KgLabel.Text = 'Kg';

% Create LeyendaPanel
app.LeyendaPanel = uipanel(app.UIFigure);
app.LeyendaPanel.ForegroundColor = [1 1 1];
app.LeyendaPanel.TitlePosition = 'centertop';
app.LeyendaPanel.Title = 'Leyenda';
app.LeyendaPanel.BackgroundColor = [0 0 0];
app.LeyendaPanel.Position = [152 44 288 107];

% Create ValorRealLabel
app.ValorRealLabel = uilabel(app.LeyendaPanel);
app.ValorRealLabel.HorizontalAlignment = 'center';
app.ValorRealLabel.FontWeight = 'bold';
app.ValorRealLabel.FontColor = [0.3922 0.8314 0.0745];
app.ValorRealLabel.Position = [5 61 283 22];
app.ValorRealLabel.Text = 'Valor Real';

% Create ValorObjetivoLabel
app.ValorObjetivoLabel = uilabel(app.LeyendaPanel);
app.ValorObjetivoLabel.HorizontalAlignment = 'center';
app.ValorObjetivoLabel.FontWeight = 'bold';
app.ValorObjetivoLabel.FontColor = [1 0 0];
app.ValorObjetivoLabel.Position = [5 42 283 22];
app.ValorObjetivoLabel.Text = 'Valor Objetivo';

% Create Valor3Label
app.Valor3Label = uilabel(app.LeyendaPanel);
app.Valor3Label.HorizontalAlignment = 'center';
app.Valor3Label.FontWeight = 'bold';
app.Valor3Label.FontColor = [1 1 0];
```

```
            app.Valor3Label.Position = [5 21 283 22];
            app.Valor3Label.Text = 'Valor 3';

            % Create Valor4Label
            app.Valor4Label = uilabel(app.LeyendaPanel);
            app.Valor4Label.HorizontalAlignment = 'center';
            app.Valor4Label.FontWeight = 'bold';
            app.Valor4Label.FontColor = [1 0 1];
            app.Valor4Label.Position = [5 0 283 22];
            app.Valor4Label.Text = 'Valor 4';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = ControlServo

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```