Trabajo Fin de Grado

Grado de Ingeniería de las Tecnologías de Telecomunicación

EVALUACIÓN EXPERIMENTAL DE SERVO MOTORES PARA ROBÓTICA AÉREA

Autor: José Manuel Sánchez Muñoz

Tutores: Guillermo Heredia Benot/Alejandro Suarez Fdez-Miranda

Departamento de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado

Grado de Ingeniería de las Tecnologías de Telecomunicación

EVALUACIÓN EXPERIMENTAL DE SERVO MOTORES PARA ROBÓTICA AÉREA

Autor:
José Manuel Sánchez Muñoz

Tutores:
Guillermo Heredia Benot
Alejandro Suarez Fernández-Miranda

Departamento de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Grado: Evaluación experimental de servo motores para robótica aérea

Autor: **José Manuel Sánchez Muñoz**

Tutores: **Guillermo Heredia Benot/Alejandro Suarez Fernández-Miranda**

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, a ____ de _____ de 2017

El Secretario del Tribunal

A mi familia

A mis profesores

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mis tutores en este Trabajo Guillermo Heredia Benot y Alejandro Suarez, por su guía, consejo y constante ayuda a la hora de afrontar este proyecto. Me ha permitido conocer y profundizar en un área de la ingeniería de gran importancia como es la robótica, y sus estudios y artículos me han resultado de gran ayuda para comprender un tema tan interesante como es el de los servomotores y su aplicación en los sistemas aéreos no tripulados.

Agradecer a mi familia, en especial a mis padres, por su apoyo y cariño durante todos mis años de formación pre-universitaria y universitaria.

Agradecer a todos y cada uno de los profesores que he tenido en mis años en la Escuela Superior de Ingenieros de la Universidad de Sevilla todos los conocimientos adquiridos y su esfuerzo para hacer de esta carrera una ingeniería interesante y bonita.

Agradecer, por último, a todos mis amigos y en especial a mi mujer, el haberme soportado durante tantos años, tanto en los buenos como en los malos momentos.

RESUMEN

Este proyecto tiene como finalidad el estudio, análisis y evaluación experimental de servomotores para su posterior uso en brazos robóticos integrados en Vehículos Aéreos No Tripulados, o UAS (*Unmmaned Aereial System)* de tipo multirotor, tratando de determinar cuáles son las características de interés para la manipulación aérea.

Tras realizar una comparativa entre dos de los principales servomotores que se pueden encontrar en el mercado (Herkulex y Dynamixel), este proyecto se centra en los servomotores Herkulex del fabricante coreano Dongbu Robot, debido a su precio más competitivo y a sus mejores prestaciones en lo que a protocolo de comunicaciones se refiere. A la hora de la evaluación experimental se ha utilizado principalmente el software 'Herkulex Manager' proporcionado por el fabricante, aunque también se han desarrollado diferentes códigos en distintos lenguajes de programación, como C++, en cuyo lenguaje se han desarrollado diferentes funciones que permiten la interactuación con el servo, o Arduino, donde se ha desarrollado un breve código apoyado por librerías ya existentes. El software Herkulex Manager permite obtener una gran cantidad de datos de utilidad a la hora de evaluar las prestaciones de estos servos.

Se explicará con la mayor claridad posible el modo de empleo de las herramientas y aplicaciones que ofrece el Software Herkulex Manager, así como las distintas funciones desarrolladas, con el fin de que pueda servir de apoyo a otros futuros proyecto que utilicen estos actuadores. Se diseñaran diferentes experimentos donde se pueda observar la gran variedad de prestaciones que ofrecen estos servomotores.

ABSTRACT

This project aims to study, analyze and perform an experimental evaluation of servo motors for their late use in robotic arms integrated on Unmanned Aerial Vehicles. It is a research in the field of smart servos to see which possibilities and advantages they could offer.

Following a market analysis between two of the main servo motors that can be found on the market (Herkulex and Dynamixel), this project focuses on Herkulex's servomotors distributed by the Korean manufacturer Dongbu Robot, due to its more competitive price and because of the benefits in terms of communication protocols. The software Herkulex Manager has been mainly used for the experimental evaluation. Furthermore, different programming languages, like C++, have been used in order to develop several functions to interact with the servo. Arduino development platform has also been used, where a brief code, supported by existing libraries, have been developed. The manufacture's software allows users to obtain a large amount of useful data, which is very helpful to evaluate the performance of these servos.

It will be explained as clearly as possible how to use the tools and applications offered by the Herkulex Manager, as well as the different programming codes developed, in order to provide support for other future project that plan to use these servos.

Difference experiments have been designed to get able to see a big variety of benefits that these servos offer.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

La Aeronáutica es una industria de importancia estratégica. Los Sistemas Aéreos No Tripulados, más conocidos por sus siglas en inglés como UAV`s (*Unmanned Aerial Vehicles*) representan un nuevo paso en la evolución aeronáutica, dando lugar a una amplia variedad de aplicaciones tales como la seguridad y vigilancia [1], fotografía y filmación aérea [2], transportes de objetos ligeros, control de cultivos[3]. A su vez esta tecnología está impulsando la aparición de nuevas oportunidades de explotación de tecnologías dispersas en otros campos (robótica), como es el caso de la manipulación aérea [4].

1.1. Manipulación Aérea

Este trabajo está centrado precisamente en el estudio de servomotores para su aplicación en manipulación aérea. Dentro de este campo, se consideran principalmente dos tipos de plataformas aéreas para la integración de brazos robóticos: en multirotores [4], cuya capacidad de carga es más limitada (de unos 2 a 3 kg), o bien sobre helicópteros [5], los cuales ofrecen una mayor capacidad de carga (en torno a unos 15 kg), aunque el control de este tipo de vehículos es más complejo. Son varios los proyectos que han sido o están siendo llevados a cabo en éste área. A continuación se mencionan algunos de los más relevantes:

- ARCAS

Proyecto del séptimo programa marco financiado por la Unión Europea, que se centra en la manipulación aérea de manera coordinada entre uno o varios sistemas UAV (tanto helicópteros cómo multirotores) con brazos robóticos integrados. Este proyecto se centra principalmente en la manipulación aérea de objetos que se encuentran ubicados en zonas peligrosas o de difícil acceso para los humanos.

Figura 1: Robot aéreo del proyecto ARCAS: quadrotor en configuración co-axial con brazo robótico de seis grados de libertad.

- AEROARMS

Proyecto financiado por el programa Horizonte 2020 de la Unión Europea, cuyo objetivo es el desarrollo del primer sistema UAV con múltiples brazos robóticos con capacidades avanzadas de manipulación aérea para poder realizar tareas de inspección y mantenimiento de plantas industriales.

Figura 2: Multirotor equipado con dos brazos robóticos del proyecto AEROARMS.

- PRODRONE

Empresa Japonesa que ha desarrollado sistemas UAVs para la manipulación aérea. Estos multirotores poseen dos brazos robóticos que permiten desplazar una gran variedad de objetos, como se puede ver en la siguiente figura.

Figura 3: Manipulador aéreo del fabricante PRODRONE

- Yale Aerial Manipulator

Proyecto de la Universidad de YALE, que se centra en la manipulación aérea a través de helicópteros UAV. Tanto este proyecto como el siguiente son muy reseñables ya que es más difícil ver a helicópteros interactuar con objetos debido a que el control de este tipo de plataformas es más complejo que en el caso de los multirotores.

Figura 4: Helicóptero UAV manipulador aéreo de la Universidad de Yale.

- Helicoptero DLR

El Grupo de Robótica Aérea del Centro Aeroespacial Alemán (DLR) ha desarrollado un manipulador aéreo compuesto de un helicóptero autónomo de elevada capacidad de carga (~20 kg), equipado con un brazo robótico industrial instalado en la base.

Figura 5: Helicóptero con brazo integrado de DLR

1.2. Smart Servos

Este proyecto propone el uso de servo motores Herkulex para el desarrollo de brazos robóticos de bajo peso integrados en un vehículo aéreo no tripulado, típicamente de tipo multirotor (quadrotor o hexarotor). Dicha investigación permitirá a estos sistemas aéreos acceder a nuevos sectores impensables hace apenas unos años. Este desarrollo abre nuevas puertas al uso de los UAV`s, como pueden ser: la realización de tareas en zonas de difícil acceso, las tareas de mantenimiento de naves industriales, el transporte de objetos,….

Los servomotores Herkulex, denominados 'smart servos', son sistemas diseñados para aplicaciones robóticas de relativo bajo coste, ya que cada servo tiene la capacidad de controlar su velocidad y posición, entre otras. El algoritmo de control de posición de cada servo puede ajustarse individualmente, permitiendo cambiar los diferentes parámetros de control de posición así como las variables del controlador. Cada servomotor integra un micro controlador que interpreta los comandos enviados por un controlador de más alto nivel, la mayoría de los cuales fijan o leen parámetros que definen el comportamiento del servomotor. A diferencia de los servos de radio-control, en los que sólo se puede controlar el ángulo de giro entre 0-180° por medio de

una señal de PWM, estos servomotores admiten una gran cantidad de configuraciones, lo que los hace más interesantes para aplicaciones que requieran mayor grado de control sobre el actuador; además el ángulo de giro se puede controlar entre +/-160º aprox. cuando se trabaje en modo de control de posición, y a su vez tiene la posibilidad de funcionar como un motor de rotación continua cuando se trabaje en el modo de control de velocidad. Una de las principales características de estos servomotores es la comunicación serial full-dúplex, que aumenta la tasa de lectura/escritura del servo. Además, varios servos pueden ser conectados en cascada compartiendo el mismo bus TTL, distinguiéndose por medio de un identificador unívoco.

Actualmente, estos servos son lo más avanzado en mecatrónica para aplicaciones de robótica de bajo peso y es por eso que ya están siendo utilizados en diferentes proyectos en el área de la robótica [6], [7], [8], [9]. Estos actuadores integran el motor, reductora, electrónica de control y comunicaciones en un dispositivo compacto que se puede ensamblar fácilmente para construir un robot manipulador. Además de permitir controlar con precisión la velocidad y la posición como ya hemos indicado anteriormente, incorporan otras prestaciones de gran utilidad como son los controladores PID (proportional-integral-derivative), distintos mecanismos de protección (sobre-carga manual, control de alimentación y temperatura) corrección automática de posición, curva de aceleración y control de velocidad sobre la marcha. Su construcción es de plástico de muy alta resistencia (heavy duty), funcionan con corriente directa (se alimentan a un determinado rango de voltios, en función de la versión del servo. Para los DRS-0101 el rango es de 7.4 a 9v, mientras que para las versiones más avanzadas DRS-0402 y 0602 es de 11.2 a 14.7V) y tienen engranes de plástico súper reforzados.

Figura 6: Herkulex Servo model DRS-0101.

1.3. Objetivo del Trabajo

El objetivo de este trabajo es realizar un estudio completo de las posibilidades que ofrecen los servomotores Herkulex con la idea de aplicarlos en un brazo robótico integrado en una plataforma aérea de tipo multirotor, así como realizar una comparativa con otras alternativas que se encuentran actualmente en el mercado y cuyas prestaciones sean similares. Se intentara demostrar por qué estos servos pueden ser más beneficiosos para su posterior uso en los sistemas aéreos no tripulados y resaltar sus ventajas y desventajas.

Además de la utilización del software del fabricante 'Herkulex Manager', se va a trabajar con distintos lenguajes de programación (Arduino y C++) viendo las ventajas que ofrece cada uno de ellos y buscando cual puede ser el más oportuno. Se generaran diferentes códigos en diferentes plataformas (Ubuntu, Arduino), y se llevaran a cabo las pruebas oportunas para comprobar las principales funcionalidades que estos servos pueden ofrecer.

Se realizaran algunas pruebas que se consideren de utilidad para su posterior uso en un brazo robótico, como el control del VOR (Velocity Over Ride), adquisición de la señal PWM, análisis del control PID del servo,…

2. SERVOMOTOR

Un **servomotor** (también conocido como **servo**) es un tipo de actuador eléctrico que integra un motor, normalmente DC, una caja reductora, y la electrónica de control y comunicaciones, permitiendo desplazarse a cualquier posición dentro de su rango de operación, y mantenerse estable en dicha posición. Estos servomotores suelen ser controlados en posición, aunque en algunos modelos es posible controlar la velocidad.

El sistema de control de los servos de radiocontrol se limita a indicar a qué posición se debe desplazar. Esto se realiza mediante una señal PWM (*Pulse Width Modulation*) de tal manera que la duración del pulso indica el ángulo de giro del motor. Cada servo tiene sus márgenes de operación, que se corresponden con el ancho del pulso máximo y mínimo que el servo puede interpretar. Si se sobrepasan los límites de movimiento del servo, éste comenzará a generar vibraciones en la señal (inestabilidad) indicando que se debe cambiar la longitud del pulso. El factor limitante es el tope del potenciómetro y los límites mecánicos constructivos.

La principal diferencia que se puede encontrar entre los servos y el resto de motores es que un servomotor tiene integrado al menos un sensor de posición (encoder o potenciómetro) que permite conocer su posicionamiento y/o velocidad en tiempo real. En otras palabras, un servomotor es un motor eléctrico al que se ha añadido un sistema de control (típicamente implementado en un microcontrolador), un potenciómetro y un conjunto de engranajes para la reductora. Con anterioridad los servomotores no permitían que el motor girara cerca de +/-180 grados (360º), solo aproximadamente +/-90 grados (180º); sin embargo, hoy en día existen servomotores en los que puede ser controlada su posición en unos +/-160 grados (320º) y su velocidad en rotación continua (Véase el manual de los servos Herkulex). En la práctica, se suelen usar estos servos para posicionar superficies de control como el movimiento de palancas, pequeños ascensores y timones. También son usados en radio control y, por supuesto, en robots.

2.1. Componentes

Un servomotor típico está compuesto por los componentes indicados en la Figura 7. Éstos son:

- Un motor eléctrico (Motor): encargado de generar el movimiento a través de su eje.

- Un sistema de regulación (Drive Gears): formado por engranajes, que actúa sobre el motor para regular su velocidad y el par. Mediante estos engranajes, se reduce la velocidad para aumentar el par de salida.
- Un sistema de control (Control Circuit): circuito electrónico que controla el movimiento del motor mediante el envío de pulsos eléctricos.
- Un potenciómetro (Potentiometer): conectado al eje central del motor que permite saber en todo momento el ángulo en el que se encuentra el eje del motor.

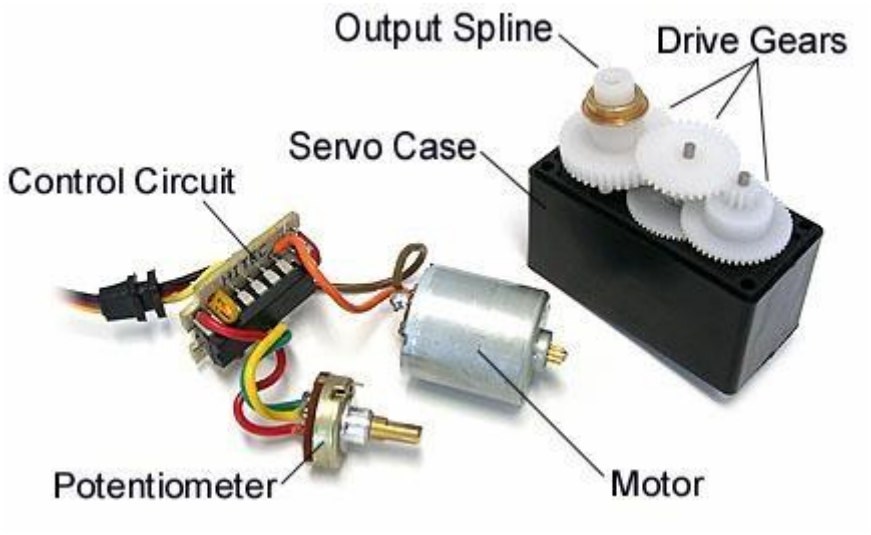


Figura 7: Componentes de un servomotor de radiocontrol clásico.

La corriente requerida por cada servo dependerá del tamaño del éste. La corriente depende principalmente del par y por norma general el fabricante indica cual es el par en función de la corriente de bloqueo ("Stall Torque").

2.2. Funcionamiento de servos RC. Control PWM

La **modulación por anchura de pulso**, PWM (*Pulse Width Modulation*), es el sistema por excelencia para el control de servos. El PWM se utiliza para controlar la referencia de posición del servo (en servos de radio control) o para controlar la corriente promedio a través del puente en H, y con ello la velocidad del motor. Se genera una onda cuadrada en la que se irá variando el tiempo que el pulso está a nivel alto, con el objetivo de realizar modificaciones en la posición del servo según se desee. Para la generación de una onda PWM en un micro controlador, lo más habitual es usar un timer y un comparador (interrupciones asociadas), de modo que el micro controlador quede libre para realizar otras tareas y la generación de la señal sea automática y más efectiva. El mecanismo consiste en programar el *timer* con el ancho del pulso (el período de la señal) y al comparador con el valor de duración del pulso a nivel alto. Cuando se produce una interrupción de *overflow* del *timer*, la subrutina de

interrupción debe poner la señal PWM a nivel alto, y cuando se produzca la interrupción del comparador, ésta debe poner la señal PWM a nivel bajo. En la siguiente figura se puede observar como el ancho del pulso varía en función de la posición a la que se pretenda enviar al servo.

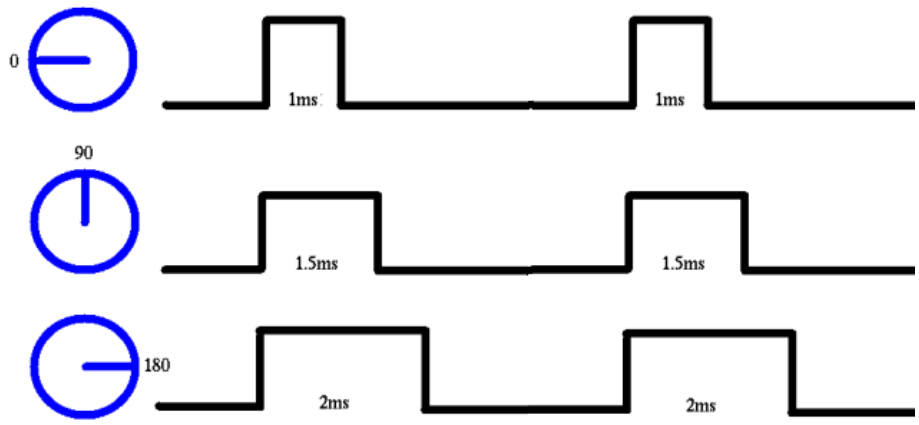


Figura 8: Posicionamiento PWM

El período que transcurre entre pulso y otro (tiempo que se encuentra a nivel bajo) no es crítico, e incluso puede variar entre un pulso y otro. Aunque en el caso de que el intervalo entre un pulso y otro sea inferior al mínimo, podría interferir con la temporización interna del servo, causando un zumbido, y la vibración en el eje de salida. Y en el caso de que este intervalo entre pulsos fuera mayor que el máximo, entonces el servo pasaría a un estado dormido, lo cual provocaría que se moviera a pequeños intervalos.

Cabe resaltar que para que un servo se mantenga en la misma posición durante un cierto periodo de tiempo, es necesario enviarle continuamente el pulso correspondiente. De este modo, si existe alguna fuerza que le obligue a abandonar esta posición, intentará resistirse. En el caso de que se deje de enviar pulsos (o el intervalo entre pulsos es mayor que el máximo) entonces el servo perderá resistencia y dejará de intentar mantener su posición, posibilitando que cualquier fuerza externa pueda desplazarlo.

2.3. Smart servo vs RC servo

La diferencia más reseñable entre servos RC y los Smart servos es la forma en la que son controlados. Con los servos comunes la comunicación es unidireccional, la información solo puede ir en un único sentido (desde la unidad de control hacia los servos). Las unidades de control, tipo Arduino, Raspberry PI, u otras, envían la posición

a la que se desea enviar al servo usando una señal PWM, al recibir esta señal, el servo la lee y se dirige a la posición indicada. El principal inconveniente de este método de control tan simple es que limita mucho el uso de estos servos, ya que no se puede recibir información del servo a la vez que se están enviando las órdenes de movimiento. La principal ventaja que tienen estos servos es que la señal PWM es común en la gran mayoría de servos de radio control (Futaba, Hitech…), sin restricciones en función de modelos o marcas, y todos responden a la misma señal.

En el caso de los Smart servos, en vez de usar la señal unidireccional PWM para el control de posición, estos servos han mejorado esta tecnología para que se puedan realizar comunicaciones bidireccionales. Aunque es cierto que en el caso de los Smart servos, el protocolo de comunicaciones varía mucho en función del modelo y del fabricante. Los más populares son TTL Half-Duplex (bidireccional pero no simultáneo), TTL Full-Duplex (bidireccional y simultáneo) y RS-485 (comunicaciones similares a las de Half-Duplex).

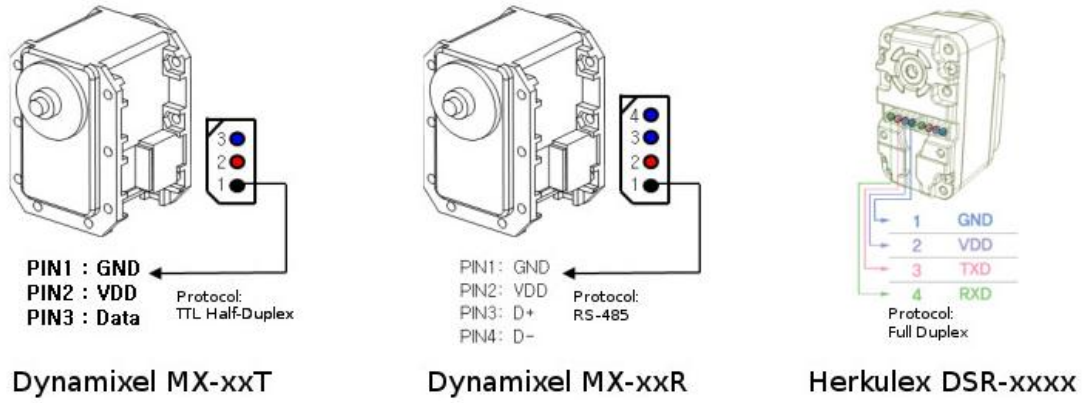


Figura 9: Interfaz física de comunicaciones en smart servos Dynamixel y Herkulex

Las ventajas que ofrecen estos servos son numerosasas. Una de ellas es la posibilidad de recibir información (feedback), lo cual permite conocer la posición del servo o la temperatura a la que se encuentra para evitar que se sobrecaliente. Esta funcionalidad es esencial en el desarrollo de manipuladores robóticos, ya que es necesario conocer la posición exacta en la que se encuentra cada una de las "articulaciones" del brazo en cada momento. Al contrario que con los servos regulares, los Smart servos no suelen estar limitados a +/- 90º (180º en total), si no que su rango de giro puede llegar hasta unos 350 grados aproximadamente. Otra de las grandes ventajas que se pueden encontrar al utilizar estos servos, es que ofrecen la posibilidad de conectarse en serie, lo que quiere decir que cada servo tiene dos conectores que ofrecen la posibilidad de conectarse unos a otros en vez de conectarlos todos a la unidad de control, y recibir y enviar la información a través del BUS de comunicaciones, lo cual es totalmente necesario a la hora de implantarlo en los brazos robóticos ya que los movimientos de cada servo deben estar coordinados, mientras que cuando se utilizan servos RC es necesario conectar cada uno de ellos a la unidad

de control, obligando a la conexión de multitud de cables, lo que dificulta el diseño final del brazo y su movimiento. Como se observa en la siguiente figura, cuando se conectan en serie es necesario que cada servo tenga una dirección única, para poder realizar las comunicaciones y que la unidad de control sepa identificar la a cada servo a través de su ID. La unidad de control enviará la orden y esta orden llevará asociado un identificador para que solo realice esta orden el servo al que va destinada.

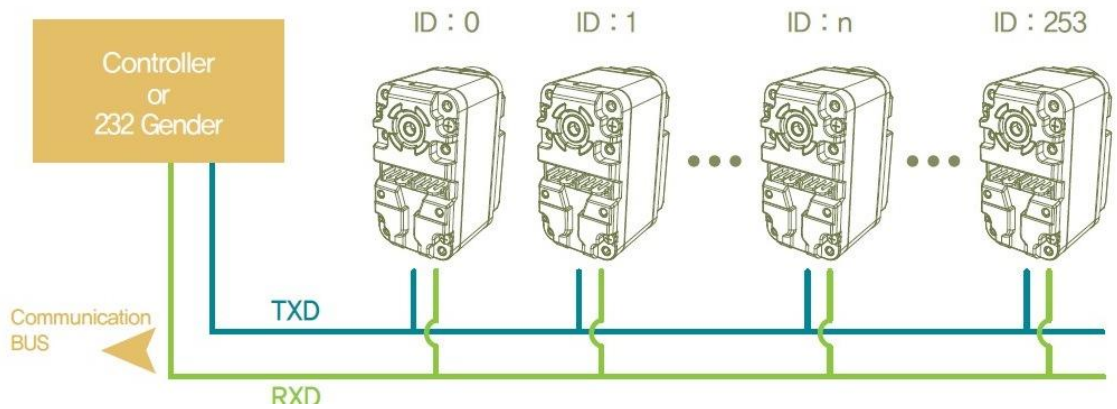


Figura 10: Conexión en serie de múltiples servos Herkulex ("Daisy chain").

2.3.1. Configuración

Los parámetros de control de los servos RC no son configurables directamente, lo cual dificultaría mucho su empleo en el campo de la robótica debido a sus limitadas posibilidades. En este área, los servos digitales sí que permiten configurar algunos de los parámetros internos de control, pero requiriendo el empleo de programas especiales para llevar a cabo dichas configuraciones.

Mientras que en los smart servos, al igual que ocurre en los protocolos de comunicación, los parámetros configurables varían en función de la marca y el modelo, pero por norma general, la mayoría de ellos permiten configurar parámetros fundamentales, como pueden ser la velocidad máxima, posición, límites de seguridad,…. Algunos de estos smart servos también pueden ser configurados para que realicen una rotación continua. La configuración de estos se realiza a través de la comunicación en serie y no se necesitan programas especiales como en el caso de los servos digitales.

2.3.2. Versatilidad y precisión

Gracias al uso de las comunicaciones en serie, diferentes tipos de comandos de entrada pueden ser enviados a los Smart servos, a diferencia de los RC que solo se le puede enviar la posición. Esto significa que con los Smart servos es posible mandar el servo a una posición a la velocidad a la que se desee. En los servos RC la velocidad va

controlada mediante la señal PWM. Para controlar la velocidad y el tiempo, se necesitan realizar todos los cálculos dentro de la unidad de control y la señal PWM necesita actualizarse constantemente para alcanzar la velocidad deseada. Como los servos RC carecen de feedbacks, si la señal PWM cambia de manera muy rápida, el servo siempre irá por detrás y la unidad de control no tendrá forma de controlar la velocidad sin usar sensores externos, como puede ser un potenciómetro o un encoder. Otra funcionalidad de gran usabilidad que ofrecen los Smart servos es, que al poder detectar la posición del servo, es posible apagar el control del motor y colocar el servo en la posición deseada. En el campo de la robótica, y más concretamente en el de los brazos robóticos, esta funcionalidad puede ser de gran utilidad, ya que se puede apagar el motor, mover las articulaciones del brazo a las posiciones que se desee para calibrarlo y luego simplemente registrar la posición actual.

Debido a que los Smart servos son la última tecnología que hay en el mercado, son los más precisos, aunque es cierto que algunos Smart servos de gama baja poseen una precisión similar a la de los servos RC. Como es lógico, cuanto mejor sea el modelo (y el precio), mayor será la precisión.

2.3.3. Desventajas de los Smart servos

Al ser una tecnología "novedosa", todavía no hay un estándar, por lo que cada compañía lo desarrolla a su manera, lo cual complica mucho al desarrollador decantarse por una marca, ya que una vez que se decide por una está "obligado" a continuar con ella. Como ya se ha mencionado anteriormente, esto pasa con los protocolos de comunicación, pero también pasa con las piezas mecánicas, ya que cada empresa tiene sus propias piezas (soportes, tornillos…).

Otra desventaja de los Smart servos es que no cubren todos los tipos de servos que existen, como por ejemplo los nano servos o los micro servos. También se puede considerar como desventaja el precio de estos servomotores, ya que aunque su precio no es excesivamente alto, siguen estando bastante por encima de los servomotores RC.

2.4. Herkulex vs Dynamixel

Tras una exhaustiva búsqueda en el mercado de los servomotores (Herkulex, Dynamixel, XYZRobot, SRS, …) se ha determinado que lo más parecido que se puede encontrar a los servo motores Herkulex son los servos Dynamixel, debido a sus características más similares y a su gran variedad de servos. Al igual que con los servos Herkulex, se pueden encontrar numerosos proyectos en el área de la robótica que utilizan estos servos, más concretamente en brazos robóticos [10], [11]. A

continuación se detalla una comparativa de los diferentes servos (separando entre modelos con características parejas) para ver las principales diferencias y cuáles serían los más adecuados en cada situación:

2.4.1. Herkulex DRS-0101 vs Dynamixel AX-12A

En primer lugar se muestran los servos de gama más baja que se pueden encontrar de estos dos tipos.

	Dynamixel AX-12A	Herkulex DRS-0101
Precio	**46€**	**33€**
Voltage	12V	7.4V (7 a 12V)
Stall Torque	15.3 Kg.cm 1.5 N.m	12 kg.cm 1.2 N.m
Máxima velocidad de rotación	59 RPM	57 RPM a 12V
Peso	55g	45g
Tamaño	32x50x40mm	24x45x32mm
Resolución	0.29º	0.325º
Corriente en Standby	50 mA	33 mA
Ángulos de operación	300º o Rotación continua	320º o Rotación continua
Protocolo	TTL Half Duplex Asynchronous Serial	Full Duplex Asynchronous Serial (TTl level)
Radio de reducción	1/254	1/266
Motor	Cored Motor	Metal Brush Cored DC
Baud Rate	1 Mbps	0.67 Mbps

Tabla 1: Comparativa Herkulex DRS-0101 vs Dynamixel AX-12A.

En esta gama ambos modelos ofrecen unas características muy similares a un precio no muy elevado, destacando que el precio de los servomotores Herkulex es significativamente más bajo que el de su homólogo (del orden de un 40 %). Por otro lado, los Dynamixel tienen un stall torque algo más alto que los Herkulex aunque también necesitan una tensión de entrada más elevada. Otro detalle a tener en cuenta es el de los ángulos de operación, que como se puede observar en la tabla es algo más elevado en el caso de los servomotores Herkulex y también ofrece la posibilidad de realizar el giro de manera continua.

2.4.2. Herkulex DRS-0201 vs Dynamixel MX-28T

Aquí se comparan los modelos de gama media de ambos servos. Se puede ver claramente que el MX-28T tiene un precio bastante más elevado.

	Dynamixel MX-28T	Herkulex DRS-0201
Precio	**228€**	**120€**
Voltage	12V (10 a 14.8V)	7.4V (7 a 12V)

Stall Torque	31.6 kg.cm 3.1 N.m	24 kg.cm 2.4 N.m
Máxima velocidad de rotación	67 RPM	63 RPM a 12V
Peso	72g	60g
Tamaño	35.6x50.6x35.5mm	24x45x32mm
Resolución	0.088º	0.325º
Corriente en Standby	100 mA	33 mA
Ángulos de operación	360º o Rotación Continua	320º o Rotación Continua
Protocolo	TTL Half Duplex Asynchronous Serial	Full Duplex Asynchronous Serial (TTl level)
Radio de reducción	1/193	1/266
Motor	Maxon RE-MAX	Metal Brush Coreless DC
Baud Rate	4.5 Mbps	0.67 Mbps

Tabla 2: Comparativa Herkulex DRS-0201 vs Dynamixel MX-28T.

En esta versión intermedia de servomotores vemos como los Dynamixel son bastante más caros, pero a su vez poseen mejores características en lo que a torque y velocidad de rotación se refiere. Otras de las ventajas que se puede observar en los Dynamixel es el Baud Rate, que en este caso es bastante más elevado, así como la resolución, ofreciendo un movimiento más preciso. De nuevo donde destacan los servomotores Herkulex en comparación con los Dynamixel es en el protocolo de comunicación. En resumen, se puede ver como en esta gama intermedia de servomotores los Dynamixel, están un poco por encima en lo que a características se refiere, pero hay una gran diferencia en el precio entre ambos (casi un 100% más caro).

2.4.3. Herkulex DRS-0402 vs Dynamixel RX-64

En esta sección se muestra la gama alta de ambos servos, y en la siguiente tabla se puede ver una comparativa de las características de ambos servos de manera más detallada.

	Dynamixel RX-64	**Herkulex DRS-0402**
Precio	**260€**	**264€**
Voltage	14.8V (12 a 18.5V)	12V (9.5 a 14.8V)
Stall Torque	64 Kg.cm a 18V 6.7 N.m a 18V	52 kg.cm a 14.8V 5.1 N.m a 14.8V
Máxima velocidad de rotación	64 RPM a 18V	59 RPM a 14.8V
Peso	125g	123g
Tamaño	40.2x61.1x41mm	35x56x38mm
Resolución	0.29º	0.028º
Corriente en Standby	50 mA	33 mA
Ángulos de operación	300º o Rotación Continua	900º o Rotación Continua
Protocolo	RS485 Asynchronous Serial	Full Duplex Asynchronous

		Serial (TTl level)
Radio de reducción	1/200	1/202
Motor	Maxon RE-MAX	Premium FAULHABER Coreless DC With Integrated Magnetic Encoder
Baud Rate	1 Mbps	1Mbps

Tabla 3: Comparativa Herkulex DRS-0402 vs Dynamixel RX-64.

A la hora de realizar la comparativa de esta gama alta, se observa que en este caso el precio de los servos es bastante similar. Como en las anteriores comparativas el torque es algo más elevado en los servos Dynamixel, aunque también lo es la alimentación necesaria. Cabe destacar que el ángulo de giro en control de posición es muy superior en los servomotores Herkulex. El punto más reseñable en esta comparativa es la resolución de los servos, que, como se puede ver, es muy superior la de los servos Herkulex, ofreciendo un control del servo mucho más preciso. En este caso la tasa de envío de ambos es la misma. El resto de características son bastante similares, con la diferencia de que de nuevo los servos Herkulex ofrecen comunicaciones Full Duplex.

2.4.4. Herkulex DRS-0602 vs Dynamixel MX-64T

Aquí se pueden apreciar la gama más alta de ambas marcas. Estos son los servos con mejores prestaciones que se pueden encontrar.

	Dynamixel MX-64T	**Herkulex DRS-0602**
Precio	**312€**	**300€**
Voltage	12V (10 a 14.8V)	12V (9.5 a 14.8V)
Stall Torque	74 kg.cm a 14.8V 7.3 N.m a 14.8V	77.5 Kg.cm a 14.8V 7.6 N.m a 14.8V
Máxima velocidad de rotación	78 RPM a 14.8V	59 RPM a 14.8V
Peso	126g	145g
Tamaño	40x62x41mm	35x56x46mm
Resolución	0.088º	0.028º
Corriente en Standby	100 mA	33 mA
Ángulos de operación	360º o Rotación Continua	900º o Rotación Continua
Protocolo	TTL Half Duplex Asynchronous Serial	Full Duplex Asynchronous Serial (TTl level)
Radio de reducción	1/200	1/202
Motor	Maxon RE-MAX	Premium FAULHABER Coreless DC With Integrated Magnetic Encoder
Baud Rate	4.5Mbps	1Mbps

Tabla 4: Comparativa Herkulex DRS-0602 vs Dynamixel MX-64T.

En esta última comparativa de las gamas más avanzadas, se puede observar que las características son prácticamente iguales, destacando por el lado de los servos Herkulex el ángulo de rotación, las comunicaciones Full Duplex, la resolución, un consumo de corriente inferior y un torque algo más elevado, mientras que los Dynamixel, al igual que en modelos anteriores tienen una tasa de envío de datos mucho más elevada.

Tras realizar una comparativa de los distintos modelos que ofrecen las distintas marcas, se llega a la conclusión de que poseen características muy similares, con algunas diferencias, como en las comunicaciones, que mientras que en los servos Herkulex es full dúplex (pueden comunicar en ambos sentidos al mismo tiempo, lo cual permite recibir información al mismo tiempo que se está enviando), en los Dynamixel es Half Duplex (primero comunica en un sentido y después en el otro), o en el stall torque, que en las versiones de gama más baja de ambos es algo superior en los Dynamixel, mientras que en las más avanzadas es prácticamente igual en ambos modelos. En cuanto a la tasa de envío (Baud Rate) de ambos, se observa que los Dynamixel ofrecen mejores características; aunque por norma general ambos servos trabajan con un Baud Rate inferior al indicado. Por último, cabe remarcar la diferencia que hay en el precio, que como se puede ver en las diferentes tablas comparativas, en las gamas más bajas los servos Herkulex el precio bastante más reducido que los Dynamixel, mientras que en las más avanzadas (los Herkulex ofrecen mejores características) se igualan.

3. Experimental Set-up

En esta sección se presentan los componentes que han sido necesarios para la puesta en marcha, así como las conexiones utilizadas.

Se procede a una explicación de cómo conectar el servo Herkulex usando los diferentes dispositivos, códigos y software (Arduino, Ubuntu C++, Herkulex Manager Kit Y Herkulex Manager Software) y se muestran los principales problemas encontrados en la puesta en marcha.

3.1. Arduino

Arduino Mega es una de las plataformas más utilizadas en el área de desarrollo de prototipos para la robótica (figura 11) [12], [13]. Esta placa utiliza un potente procesador de AVR ATMEGA2560 con un amplio espacio de memoria (256 KB of Flash Memory y 8KB de RAM) para programar y corre a 16Mhz. Es ideal para proyectos de robótica, ya que lo más destacado es su elevada cantidad de pines de entrada y salida (54 Digitales I/O, 16 entradas analógicas y 14 PWM) y sus 4 puertos UART por hardware. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino y el entorno de desarrollo Arduino. La comunicación entre el ordenador y Arduino se produce a través del Puerto Serie. Posee un convertidor usb-serie, por lo que sólo se necesita conectar el dispositivo a la computadora utilizando un cable USB como el que utilizan las impresoras. Arduino se enfoca en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos en proyectos multidisciplinarios. Toda la plataforma, tanto para sus componentes de hardware como de software, son liberados con licencia de código abierto que permite libertad de acceso a ellos. Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, ya que una vez este cargado el código con sus librerías en la placa ya solo será necesario conectar el dispositivo a la placa.

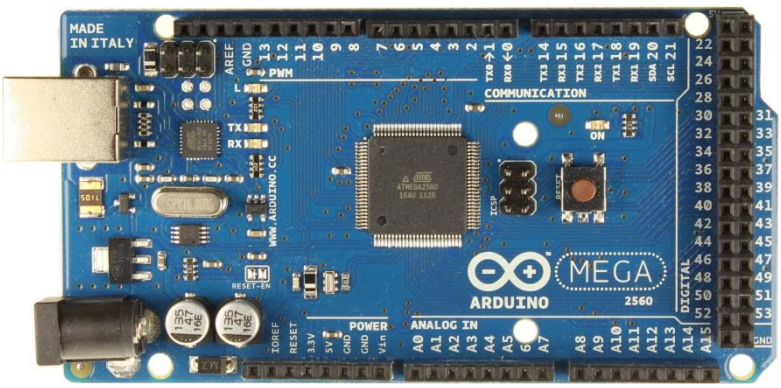


Figura 11: Arduino Mega 2560

3.1.1. Conexiones

Para poder comunicarse con el servo usando esta plataforma habrá que crear una librería que permita manipular el servo mediante el uso del lenguaje de programación Arduino. A través de una exhaustiva búsqueda se han encontrado diversas librerías que pueden utilizarse para la manipulación de estos servomotores, las cuales han resultado de gran utilidad a la hora de realizar las pruebas. Una vez incluidas las librerías y el código que se van a utilizar, se deberá conectar la placa al ordenador para poder ejecutar el código usando el entorno de desarrollo Arduino. Posteriormente, se deberá introducir el código en la placa Arduino como se muestra en el siguiente apartado; y por último, se deberá conectar el servo a la placa para poder realizar las pruebas deseadas.

El material utilizado para la puesta en marcha del servo ha sido el siguiente:

- Cable USB de tipo A/B para conectar la placa Arduino con el ordenador
- Placa Arduino MEGA
- Servo motor Herkulex DRS-0101
- Fuente de alimentación de 7,5V y hasta 2,250A que se conecta a la placa Arduino y a su vez se usa para alimentar el servo motor.
- También se ha usado una protoboard como apoyo para conectar los pines de Arduino con el servo.

Para conectar el servo a la placa Arduino se han realizado las conexiones que se muestran en la siguiente figura. Se ha alimentado la placa a 7.5V con un cargador regulable y se han usado los pines GND y Vin de la placa para alimentar el servo a 7.5V. Una vez alimentada la placa y el servo se conectan los pines Tx (transmisor) y Rx (receptor). En este punto, hay que tener cuidado y realizar las conexiones de manera que el Tx del servo esté conectado al Rx de la placa (pin 19) y el Rx al Tx (pin 18) para poder comunicarse de manera correcta. En las siguientes figuras se muestra cómo se realiza el conexionado de manera teórica (en la primera) y como se ha realizado en la práctica.

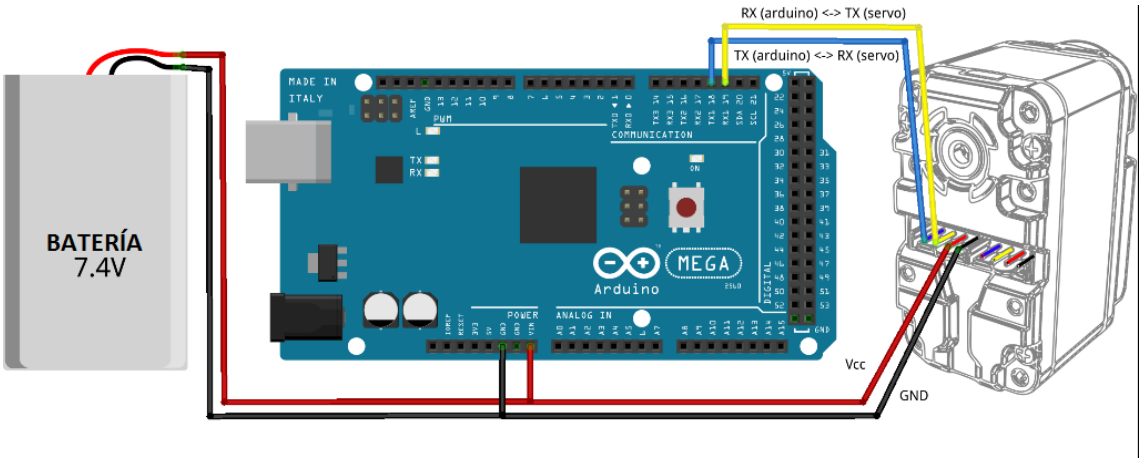


Figura 12: Conexiones Arduino indicado en el manual.

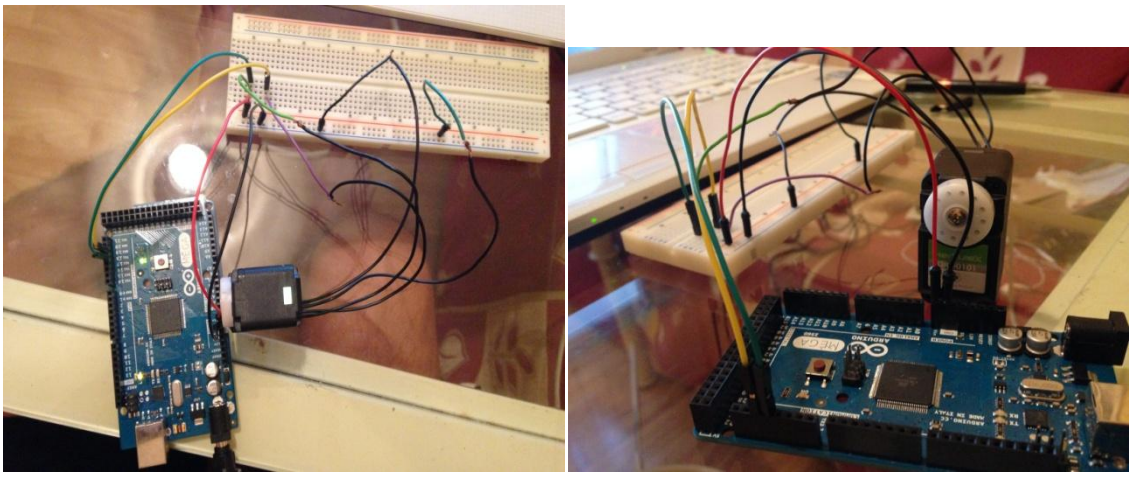


Figura 13: Conexionado del servo Herkulex DRS-0101 con la placa Arduino Mega.

3.1.2. Puesta en marcha

Una vez detallado el conexionado necesario para conectar el servo con la placa Arduino, se procede a explicar la puesta en marcha de este a través del software de Arduino. Lo primero que se debe hacer es comprobar el puerto COM al que está conectada la placa Arduino y configurar los valores de dicho puerto (baudrate, bits por segundo, bits de datos, paridad y control de flujo) con lo que se necesita para trabajar con el servo. El Baudrate ideal para trabajar con estos servomotores varía en función si se va a utilizar Arduino UNO o Mega. En este caso, se va a utilizar MEGA, por lo que se configurara el baudrate a 115200; en caso de que se utilice UNO convendría configurarlo a 57600. En la siguiente figura se puede observar cómo y dónde (Administrador de dispositivos de Windows) realizar estas modificaciones.

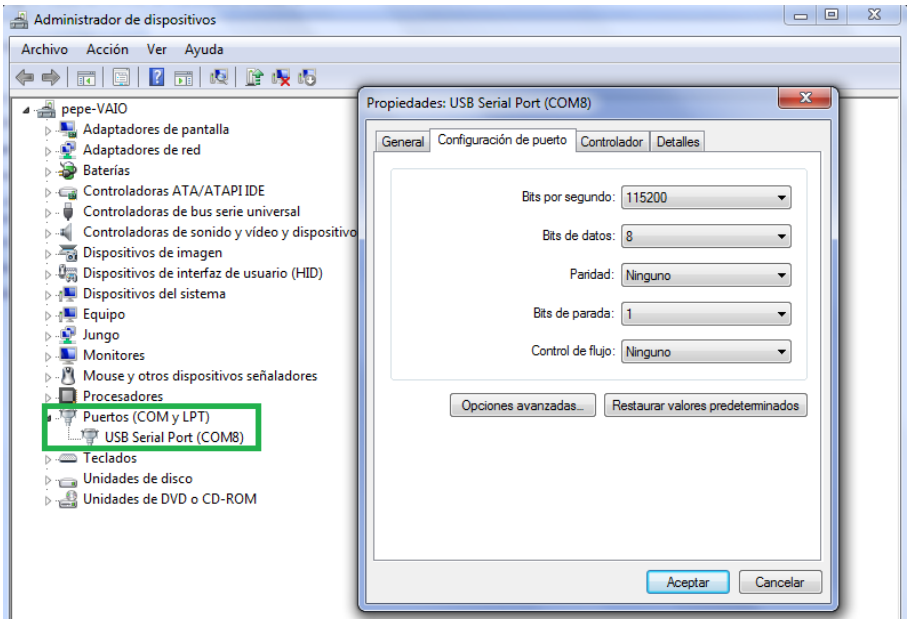


Figura 14: Configuración puerto COM

Cuando todo esté bien conectado se debe abrir el entorno de desarrollo de Arduino [14], donde se creará un proyecto en el que se encuentre el código (en caso de que la librería necesaria no esté incluida habrá que añadirla).Cuando se haya creado el proyecto, se deberá abrir el código, como se muestra en la siguiente figura.

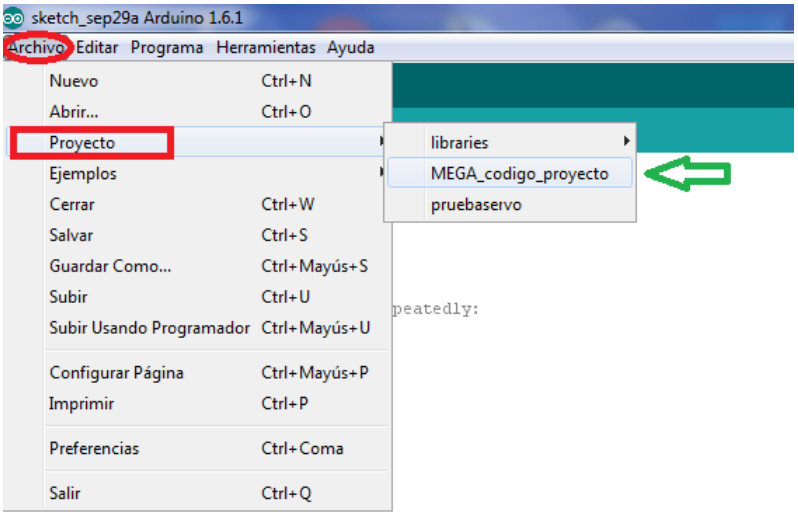


Figura 15: Creación de proyecto en Arduino.

Una vez abierto el código (proyecto), se deberá compilar para ver que no hay ningún error de código y posteriormente subir el proyecto a la placa Arduino, para que este a su vez mande las instrucciones al servo.

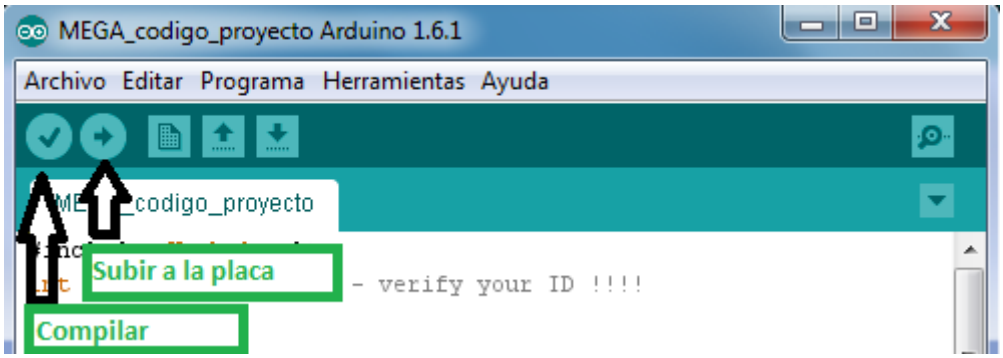


Figura 16: Compilación y carga del código en la placa

3.2. Herkulex Manager

A la hora del conexionado para la utilización del software que suministra el fabricante, se ha utilizado los dispositivos incluidos en el Manager Kit para Herkulex servo que lo suministra la misma empresa (Dongbu) que los servo motores. Este Kit se compone de:

- Cable RS232
- Adaptador RS232 a USB
- Cable de batería
- Herkulex Manager Kit para el conexionado de todos los componentes indicados anteriormente.

3.2.1. Conexiones

Se ha seguido el esquema de la siguiente figura para realizar el conexionado de estos componentes con el servo.

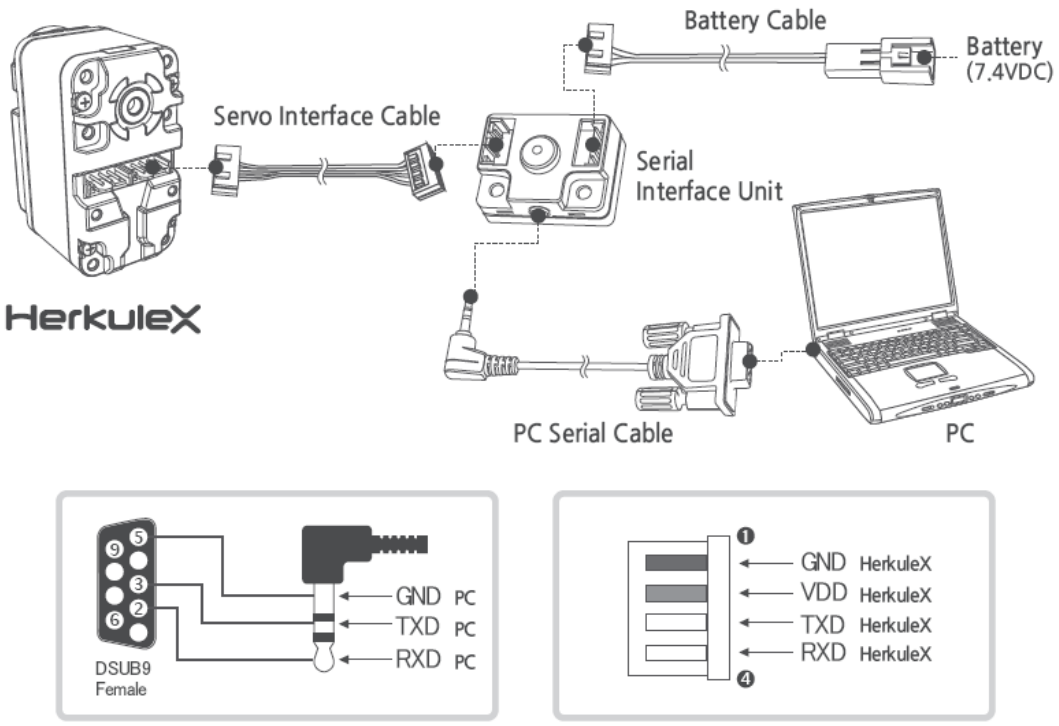


Figura 17: Conexionado Herkulex Manager Kit indicado en el manual.

A la hora de alimentar el servo motor (con 7.4 V) se ha utilizado la placa Arduino como apoyo para realizar la conexión de manera más sencilla, como se puede observar en la siguiente figura. Se alimenta la placa a 7,5V mediante el adaptador de corriente y se conecta los cables del Herkulex Manager Kit a los pines de tierra y Vin de la placa. Es importante comprobar que el PC Serial Cable queda bien conectado al Serial Interface Unit ya que se ha podido comprobar que es un error muy frecuente.

Figura 18: Conexionado Herkulex Manager Kit.

3.2.2. Puesta en marcha Herkulex Manager

Cuando se hayan realizado todas las conexiones de manera correcta, será necesario instalar el software Herkulex Manager, disponible de manera gratuita en la web del fabricante [15]. Dicho programa es un programa orientado al manejo de los servomotores y permite realizar experimentos, pudiendo visualizar los distintos parámetros de salida que ofrecen estos servomotores a través de graficas o archivos de salidas que pueden ser simulados en otros programas. Mediante el empleo de este software se pueden visualizar y modificar distintos parámetros relativos al servo como pueden ser los valores del controlador (proporcional, derivativo e integral), la temperatura máxima permitida en el servo, las tensiones máximas y mínimas, los limites de posición a las que puede llegar al servo, los valores máximos y mínimos del PWM, offset, modificar la ID del servo, y otras numerosos parámetros que pueden ser de gran utilidad a la hora de configurar el servo para que se adapte de la mejor manera posible a las necesidades de cada uno. Otra de las principales ventajas que se encuentra a través del uso de este programa es la posibilidad de realizar modificaciones en tiempo real. En los siguientes apartados se podrá observar como muchos de estos parámetros son modificados con el fin de analizar el servo en profundidad.

Una vez este el programa instalado y las conexiones realizadas, se procede a la puesta en marcha de los servomotores. Lo primero que se debe hacer, al igual que en la configuración con Arduino, es comprobar el puerto USB al que está conectado y configurar los valores del puerto (baudrate, bits por segundo, bits de datos, paridad y control de flujo) con lo que se necesita para trabajar con el servo.

El siguiente paso será abrir el programa y conectar el servomotor. Se debe comprobar que los parámetros del puerto corresponden a los configurados anteriormente en el administrador de dispositivos. Para realizar la conexión se deben seguir los pasos que se muestran a continuación:

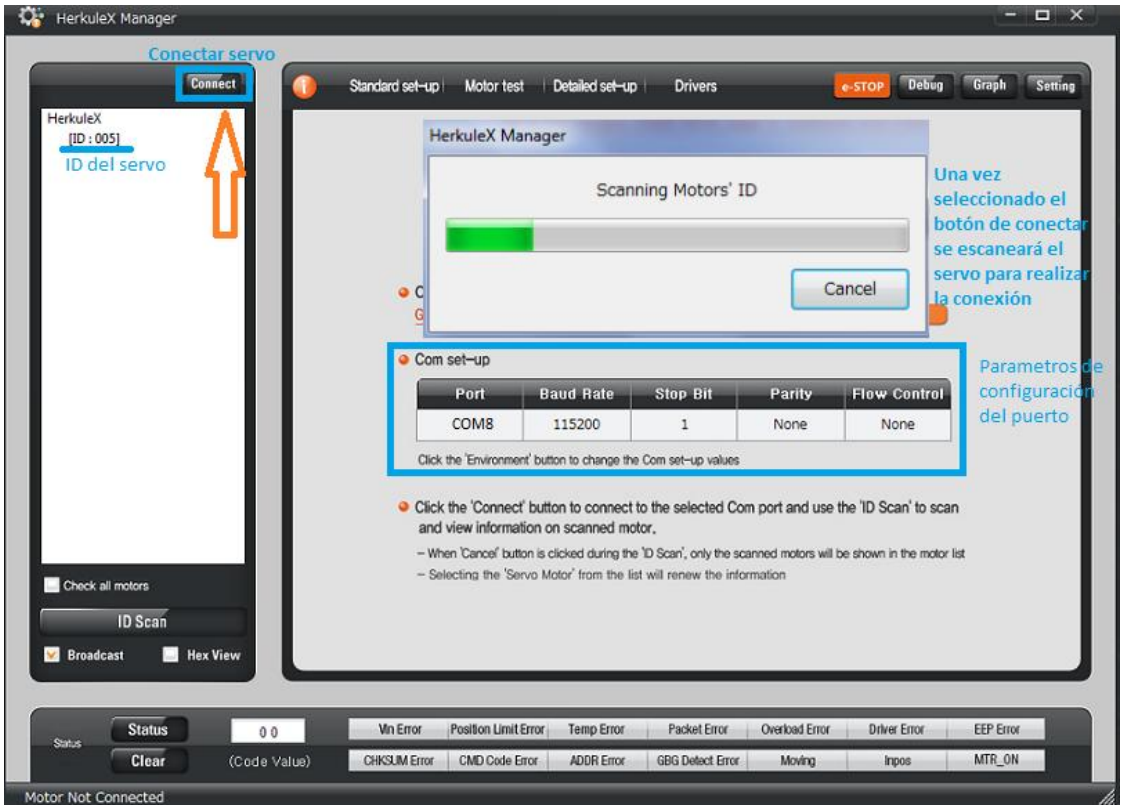


Figura 19: Puesta en marcha Herkulex Manager

Una vez el servo esté conectado correctamente y en el botón donde antes se podía leer “Connect”, ahora se lea “Disconnect”, se podrá comenzar a trabajar con el servo.

3.2.2.1. Funcionalidades

A continuación se analizan las distintas funcionalidades, algunas mencionadas anteriormente, que se pueden encontrar en dicho software. Como se puede observar en la figura anterior, en la interfaz principal se aprecian 3 secciones: la primera, se puede ver abajo, en “status”, y es donde se podrá observar continuamente el estado del servo, ver cómo está trabajando y si se produce algún error en este; la segunda se encuentra arriba a la izquierda, y como se ha explicado en el apartado anterior, es donde se realizaran las conexiones con el servo y donde se podrá ver los servos que están conectados; y, por último, en la tercera sección (arriba a la derecha), se aprecian distintas pestañas, las cuales serán detalladas a continuación:

- Standard set-up

Como su propio nombre indica en esta sección se puede realizar modificaciones en tiempo real de los distintos parámetros que afectan a la configuración del servo. Estos parámetros son: Color de los Leds, Ganancias del servo (PID), Aceleración (radio y tiempo límite), zonas de saturación (offset y zonas muertas), los límites de posición del

servo, los valores relativos al PWM (máximos, mínimos, offset y límites), límites máximos y mínimos de voltaje al que puede trabajar el servo, control de temperatura máxima y mínima, calibración, control de errores (que serán mostrados mediante el parpadeo de los LEDs), torque del servo y modificar la ID de estos. Todos estos datos pueden ser modificados durante las distintas pruebas, quedando almacenados en la memoria volátil RAM, por lo que una vez se desconecte el servomotor estos valores quedaran borrados, ya que no se almacenaran en la memoria no volátil EEP, aunque el software también ofrece la posibilidad de guardar estos valores de la prueba en la memoria EEP y que queden de esta manera guardados en el servo.

- Motor test

Esta pestaña puede ser de gran utilidad si se quiere obtener resultados de la prueba realizada, ya que se podrá generar archivos con los datos de salida. En la parte superior, se le indicara al servo los movimientos a realizar y en la parte inferior se podrá ver los diferentes outputs que se van recibiendo del servo, para finalmente generar el archivo en el apartado “Result”, dándole al botón “Save”.

- Detailed Set-up

En esta pestaña se pueden encontrar los valores de los distintos parámetros del servo almacenados en las dos memorias del servo (RAM y EEP). Mientras la memoria RAM es la memoria volátil en la que se almacenan los valores indicados en las pruebas realizadas, la memoria EEP es una memoria no volátil en la que se almacenaran los valores que quedaran guardados de manera permanente hasta que vuelvan a ser modificados. En la memoria RAM, al ser una memoria volátil, los valores se borraran una vez se desconecte el servo. Los valores que están almacenados en la memoria EEP se cargaran automáticamente en la memoria RAM cada vez que se alimente el servo, ya que los datos almacenados en la memoria no volátil no tendrán ningún efecto en las operaciones del servo, que utilizara los datos cargados en la memoria RAM.

Los valores que se pueden encontrar y modificar dentro de cada una de las dos memorias, son los que se pueden ver en las páginas 21/22 y 24/25 del manual del servo [16]. Los valores almacenados en la memoria RAM pueden ser cargados en la EEP presionando el botón “RAM 2 EEP” que se puede encontrar entre las dos memorias como se puede ver en la siguiente figura.

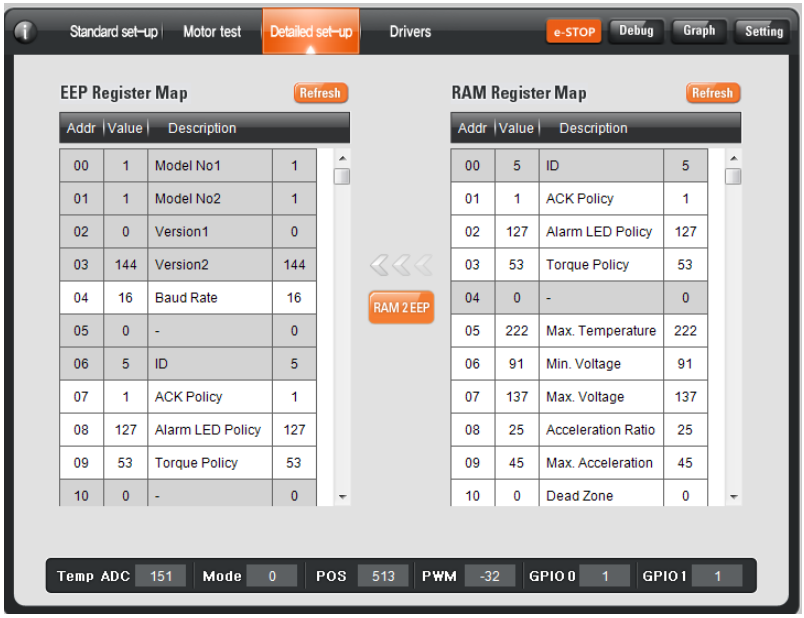


Figura 20. Registros internos en memoria RAM y EEPROM del servo

- Drivers

Esta pestaña se encarga de las versiones del programa y de todos sus drivers. Se pueden realizar las actualizaciones de software. De la misma manera se pueden cargar archivos hexadecimales que hayan sido generados en la pestaña “Motor test” para realizar experimentos con el servo.

- Debug

Esta pestaña puede resultar de gran utilidad para entender mejor como se realizan las comunicaciones entre el programa y el servo y ver como se envían o reciben los paquetes de datos que indican al servo las maniobras que ha de realizar. En la siguiente figura se puede observar como son enviadas las tramas. A su vez permite generar tramas y enviarlas al servo para que este realice la instrucción deseada.

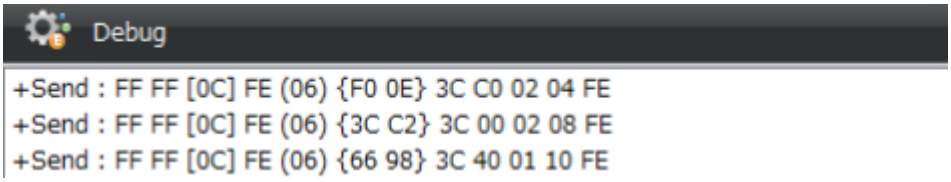


Figura 21. Envío de tramas

- Graph

Esta pestaña puede ser de las más utilizadas ya que mediante su uso se puede obtener información de gran utilidad de los servomotores de una manera muy visual. Lo primero que se debe hacer antes de mandar las órdenes al servo es seleccionar el botón de “Torque ON” para permitir que el servo pueda moverse a las posiciones indicadas. Una vez seleccionado dicho botón se generara la “ruta” deseada. Se pueden

realizar distintas pruebas en las que se envía al servo a una o varias posiciones determinadas seleccionando la opción “POS CTRL” (se le indica al servo la posición deseada, el tiempo en el que realizar el movimiento y el color del LED) o indicar al servo que rote de manera continua “CONT ROT”. Una vez se haya generado las ordenes que se enviaran al servo y se comience la prueba se podrán observar los resultados de esta en la gráfica que se encuentra en dicha pestaña. En estas gráficas se pueden visualizar los siguientes valores:

  - POS → Posición del servo
  - PWM → Señal pwm
  - aPOS → Posición absoluta
  - 2nd → Segunda posición absoluta (Posición absoluta del potenciómetro). Este campo solo se puede visualizar en los servos más avanzados, que tienen un codificador magnético integrado (Herkulex DRS 0402 y DRS 0602)
  - DSP → Velocidad
  - ASP → Velocidad angular
  - Temperature → Temperatura del servo
  - Vin → Tensión de entrada
- Setting

En esta pestaña se definen los valores de “Set-up” del servo; también se definen algunos de los aspectos relativos a la inicialización de la conexión con el servo.

3.2.2.2. Desventajas

A la hora de realizar un análisis del servomotor y de todas sus características, este programa resulta de gran utilidad, ya que se pueden analizar prácticamente todas las funcionalidades/características que ofrecen estos servomotores. Se pueden realizar experimentos, se pueden obtener gráficas relativas a los movimientos del servo, se pueden observar y modificar los valores almacenados en las dos memorias, generar archivos relativos a las pruebas realizadas, etc; pero la principal desventaja que se puede encontrar en el empleo de este programa es que no permite la posibilidad de generar un programa que realice los movimientos deseados sin estar conectado al PC. Esta desventaja puede ser muy importante si se pretende utilizar estos servomotores en brazos robóticos o cualquier otro tipo de robot, ya que en dichos casos será necesario utilizar microcontroladores en los que se carguen las tareas/movimientos que debe realizar el servomotor.

3.3. Programa de adquisición de datos en C++ sobre máquina virtual de Ubuntu

En esta sección se ha utilizado una maquina virtual para trabajar con Ubuntu. Ubuntu es un sistema operativo basado en Linux que se distribuye como software libre, el cual incluye su propio entorno de escritorio denominado Unity. Está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario. Está compuesto de múltiple software normalmente distribuido bajo una licencia libre o de código abierto, lo cual permite que para la realización diferentes tareas, se pueda descargar la aplicación informática correspondiente de manera gratuita.

3.3.1. Conexiones

Las conexiones realizadas para trabajar a través del código C++ son las mismas que se han utilizado para trabajar con el software Herkulex Manager. En la siguiente figura se puede apreciar cómo se han realizado las conexiones y como está conectado a la maquina virtual Ubuntu.

Figura 22. Conexionado en la prueba de adquisición de datos con el programa desarrollado en C++

La diferencia se encuentra a la hora de de poner el servo en marcha y será explicado en el siguiente apartado.

3.3.2. Puesta en marcha

Al realizarse las conexiones de la misma manera que en el apartado Herkulex Manager, los primeros pasos serán los mismos (comprobar que el dispositivo se encuentra conectado correctamente al puerto COM y que los parámetros están bien configurados).

Para cargar el programa en el servo, se deberá encender la maquina virtual y abrir el terminal para poder empezar a trabajar. Se deberá crear una carpeta con el nombre del proyecto y en dicha carpeta se incluirá el código en c++ y un archivo .txt donde se cargaran los resultados de las pruebas indicadas en el código. Para realizar la compilación se deberá acceder a través del terminal a la carpeta del proyecto y se ejecutara el siguiente comando:

g++ -W -Wall Main.cpp -o Test

Y una vez la compilación se realice de manera correcta se ejecutara el programa a través del siguiente comando.

sudo ./Test /dev/ttyUSB0 IDservo OutDataFile.txt

Al ejecutar el código lo que se le introduce al programa por línea de comandos son 4 argumentos:

- *Nombre del proyecto: ./Test*
- El puerto de entrada del servomotor: */dev/ttyUSB0*
- La ID del servo: *IDservo*
- El fichero en el que se almacenarán los datos obtenidos de las pruebas: *OutDataFile.txt*

3.4. Problemas de arranque

Durante los primeros meses del proyecto surgieron varios inconvenientes, que hacían imposible la correcta conexión y arranque de los servos DRS-0101 y DRS-0402. A continuación se explica detalladamente las pruebas que se realizaron, los problemas surgidos y la solución que se encontró a dichos problemas.

En un principio se considero que no era necesaria la adquisición del Herkulex Manager Kit para conectarlo con el ordenador, ya que la idea era conectarlo a través de diferentes dispositivos de los que ya se disponían (UART to USB, Arduino, Raspberry PI).

3.4.1. Arduino

Problema.

La primera conexión se intento realizar con una placa de Arduino Mega 2560. Se realizaron las conexiones que se muestran en el apartado 3.1, y usando el DRS-0101 se procedió a la inserción de un simple código que se muestra a continuación.

```
#include <Herkulex.h>

void setup()  {

 int n=0xfd; //motor ID

 delay(2000);

 Serial.begin(115200);    // Abrir serial communications

 Serial.println("Begin");

 Herkulex.beginSerial1(115200); //Abrir serial port 1

 Herkulex.reboot(n); //reboot

 delay(500);

 Herkulex.initialize(); //inicio motores

 Serial.println("Set Led Green");

 Herkulex.setLed(n,LED_GREEN); //Led en verde

 Serial.print("Status:");

 Serial.println(Herkulex.stat(n)); //control de errores

 Herkulex.end();

}

void loop(){ }
```

Este código pone los LED's del servo de color verde. Estas pruebas no funcionaron ya que los LED's no se encendían de manera correcta. Desde un principio se podía ver que el problema estaba en la comunicación del servo con la placa Arduino, ya que el código se compilaba y se enviaba a la placa sin ningún problema.

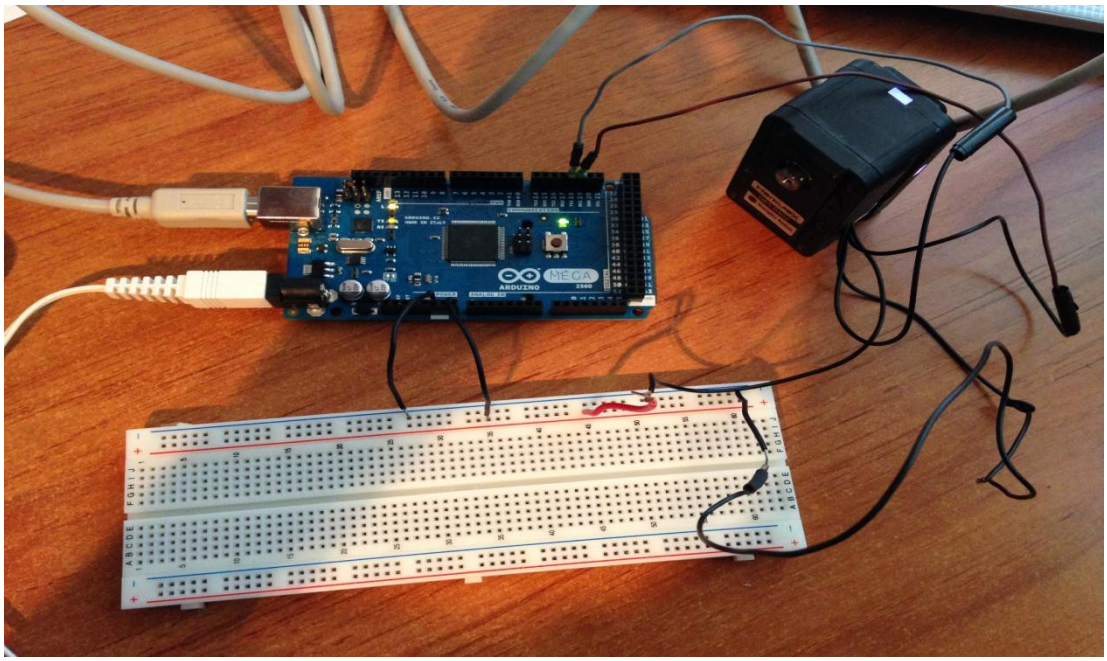


Figura 23: Primeras conexiones Arduino con servo Herkulex DRS-0402.

Solución

Ante la continuidad de los problemas se barajó la opción de que los servomotores hubieran llegado defectuosos, ya que no había manera de ponerlos en marcha, por lo que se procedió al envío de vuelta al fabricante para ver si ese era el caso, o es que se estaban realizando de manera incorrecta las correcciones. Finalmente, tras varias semanas el fabricante indicó que el servo se encontraba bien, pero que tenía la ID del servo modificada con respecto a la que debía tener por defecto. Asimismo indicó que para interactuar con el software Herkulex Manager era necesaria la adquisición del Herkulex Manager Kit. Con estas nuevas indicaciones se pudieron solucionar los diferentes problemas.

En el caso de Arduino, una vez conocida la nueva ID del servo, ya se podía comunicar la placa con el servo sin ningún problema.

3.4.2. Convertidor UART (UMFT230XB)

Problema

En esta ocasión se procedió a realizar la conexión usando un convertidor UART to USB para usar el software Herkulex Manager, así como para introducir un código realizado en C++. Siguiendo las instrucciones del datasheet del convertidor UART al igual que las del datasheet del servo DRS-0402, se realizaron las conexiones que se pueden ver en la figura 24.

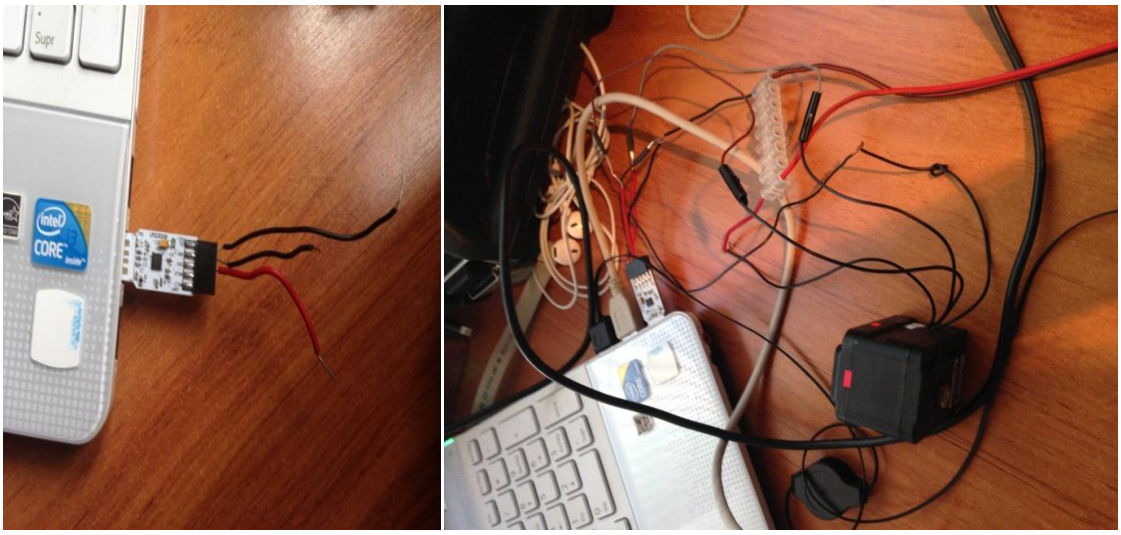


Figura 24: Conexiones UART. El LED del servo parapadea en rojo debido a un error de conexionado.

En este caso se puede ver que al conectar el servo a la alimentación se encendía automáticamente el LED rojo, el cual indicaba que algo no se estaba haciendo de manera adecuada.

Con estas mismas conexiones, al intentar detectar el servo con el Herkulex Manager, no se reconocía ningún servo y mostraba un error de conexionado.

Solución

En cuanto a lo del convertidor UART, se llego a la conclusión de que las conexiones que se estaban realizando no eran las correctas y tras varias vueltas al asunto se acabo encontrando el problema, y era que el manual usado no era el de la página oficial de Dongbu y las conexiones indicadas en dicho manual estaban invertidas lo cual estaba causando que nada mas conectarlo se encendieran los LED's de color rojo.

3.4.3. Raspberry PI

Problema

La última de las pruebas realizadas fue con la placa Raspberry PI. En esta ocasión se alimentó el servo sin conectarlo al puerto serie y se encendió el LED de color azul, que indica el correcto funcionamiento del servo. Posteriormente se comenzó a mandar datos al servo conectando el Tx de la Raspberry al Rx del Herkulex, pero sin conectar el Tx del Herkulex al Rx de la Raspberry, y entonces se encendió el LED rojo parpadeando, lo cual indica que detectó algún error en el protocolo de comunicaciones o que el nivel de tensión del puerto serie era 3.3V y no 5V. Lo

siguiente fue conectar el Tx del Herkulex al Rx de la Raspberry, y ahí fue cuando se fundió la Raspberry, con lo que se llegó a la conclusión de que los niveles TTL que usa el servo Herkulex son de 5V, algo que no se indica de manera clara en el manual, y esto terminó ocasionando una sobretensión en el puerto serie y fundiendo la Raspberry ya que los puertos de esta trabajan a 3.3v y no son tolerantes a 5V.

4. Desarrollo de software

En esta sección se procede a explicar los diferentes códigos que se han desarrollado y cómo funcionan las comunicaciones con los servo motores Herkulex (envío de tramas para escritura y lectura). De la misma manera, se muestran y explican las funciones más significativas que se han desarrollado para cada una de las plataformas indicadas en la sección anterior.

4.1. Envío de tramas

La primera tarea necesaria para comenzar a trabajar en los códigos para las diferentes plataformas utilizadas a lo largo del proyecto es conocer bien como se realizan los envíos de tramas, tanto para lectura, como para escritura.

Es importante comprender que estos servos se componen de dos memorias en las que almacenar los datos; una memoria RAM, la cual es volátil, y solo almacenara los datos mientras el servo esté conectado, y una memoria EEP no volátil, que guardará la información en el servo aunque este se desconecte.

La siguiente tabla muestra la composición de un paquete con cada uno de los campos que van en él. Estos campos serán explicados a continuación.

Type	Header	Packet Size	pID	CMD	Check Sum1	Check Sum2	Data[n]
Value	**0xFF 0xFF**	**7-223**	**0-0xFE**	**1-9**	**Se explica a continuación**	**Se explica a continuación**	**Depende del CMD**
Byte	1 1	1	1	1	1	1	Máx. 216

Tabla 5. Paquetes de datos

Los campos que componen los paquetes de datos son los siguientes:

- Header (Cabecero)

Indica el inicio de un paquete. Está compuesto por dos valores iguales (0xFE) de un byte cada uno.

- Packet Size (tamaño del paquete)

Este campo indica el tamaño del paquete en bytes, que podría variar desde 7 (paquete sin ningún dato) hasta 233. En caso de superar los 233 bytes no se reconocerá el paquete. El tamaño de los paquetes lo determinara el numero da datos que se pretendan enviar en cada paquetes, ya que el resto de campos tiene un tamaño fijo.

- pID

En este campo se indicara la ID del servo al que se le envía el paquete. Tiene que ser un valor entre 0 y 253 (número total de servos que pueden ir conectados). Hay que tener cuidado con indicar el valor 0xFE (253), ya que la orden afectaría a todos los servos conectados en la red.

- CMD

Este es el campo más "importante" del paquete, ya que marcará la instrucción que se pretende realizar con el envío de este. Para controlar el servo se puede diferenciar entre 9 tipos de CMD, de los cuales se van a resaltar los 4 más interesantes, ya que son los que permitirán realizar las lecturas o escrituras en memoria:

  - EEP_WRITE(0x01)

Se utiliza este "Request Packet" para almacenar información en la memoria EEP

  - EEP_READ(0x02)

Se utiliza este "Request Packet" para leer información de la memoria EEP. Este envío de paquete tendrá o no respuesta dependiendo de la política de envíos ACK

  - RAM_WRITE(0x03)

Se utiliza este "Request Packet" para almacenar información en la memoria RAM

  - RAM_READ(0x04)

Se utiliza este "Request Packet" para almacenar información en la memoria RAM. Este envío de paquete tendrá o no respuesta dependiendo de la política de envíos ACK

El resto de tipos de CMD son I_JOG(0x05), para comandar los tiempos de operación de los servos de manera individual; S_JOG(0x06), para comandar los tiempos de operación de los servos de manera simultánea, STAT(0x07), para indicar el estado de los errores; ROLLBACK(0x08), para devolver todos los valores de la memoria EEP a los valores que venían por defecto en el servo y REEBOOT(0x09), para reiniciar el servo.

Una vez el servo reciba el "Request Packet" con alguno de los CMD explicados incluido, el servo realizará la operación indicada y como resultado devolverá al controlador un paquete ACK como respuesta (esta respuesta no será siempre necesaria, ya que solo se dará en función de la política ACK que se defina). Los 9 tipos de "ACK Packet" son los siguientes: EEP_WRITE(0x41), como respuesta al EEP_WRITE(0x01); EEP_READ(0x42), para responder con el número de valores indicados en el "Request Packet"; RAM_WRITE(0x43), como respuesta al

RAM_WRITE(0x03), RAM_READ(0x44), para responder con el número de valores indicados en el "Request Packet". El resto de tipos (I_JOG(0x45), S_JOG(0x46), STAT(0x47), ROLLBACK(0x48) y REBOOT(0x49)), recibirán un paquete ACK como respuesta a cada uno de sus envíos.

- Check Sum1

Check Sum1 se usa para detectar errores en los paquetes. La fórmula que utiliza es la siguiente:

$$CheckSum1 = (Tamaño\_paquete \hat{} pID \hat{} CMD \hat{} Data[0] \hat{} Data[1] \hat{} \dots \hat{} Data[n]) \ \& \ 0xFE$$

- Check Sum2

Al igual que Check Sum1, Check Sum2 se utiliza para detectar errores en los paquetes y su formula se puede ver a continuación:

$$CheckSum2 = (\sim CheckSum1) \ \& \ 0xFE$$

- Data[n]

El número de datos enviados dependerá del CMD que se indique, y, como se menciona anteriormente, es posible que algún paquete no lleve ningún dato. El máximo número de datos que se pueden enviar es 216.

4.2. Arduino

Para poder manipular el servomotor a través de la placa Arduino se ha generado un código en el lenguaje de programación Arduino (lenguaje de programación de alto nivel, similar a C++). Dicho código se ha apoyado en una librería (encontrada en internet) para poder realizar las pruebas deseadas.

4.2.1. Código

En este apartado se explicará cómo se ha realizado el desarrollo del código y las librerías utilizadas para el control del servo, así como algunas de las funciones usadas para el control del servo motor.

Lo primero que se tiene que hacer es incluir las librerías utilizadas, así como la id del servo:

```
#include <Herkulex.h>

int ID=0x05;
```

Una vez se defina la ID del servo, se tendrá que comenzar las comunicaciones con el servo y habilitar este para poder realizar las instrucciones indicadas. En un primer paso se abrirá el puerto serie sobre el que se va a trabajar para realizar las comunicaciones, a continuación se reinician los motores a sus valores iniciales, y, por último, se inicializan los motores:

```
void setup()  {
    delay(2000);                    //Un tiempo de espera
    Herkulex.beginSerial1(115200);  //Función que se encarga de abrir el puerto serie 1 para las comunicaciones y que habilita los pines 18 y 19 de la placa Arduino
    Herkulex.reboot(ID);            // Reinicia los motores a sus valores iniciales
    delay(500);
    Herkulex.initialize();          //inicializa los motores
    }
```

Una vez abiertos los puertos para las comunicaciones e inicializados los servomotores se podrá empezar a trabajar en las instrucciones que se deseen enviar a los servos. A continuación se muestra un pequeño ejemplo de una orden de movimiento que se le envía al servo:

```
void loop(){
        Serial.println("Move Angle: POSITION 150");
        Herkulex.moveOneAngle(ID, 150, 1250, LED_GREEN);  //Se envía al servo a la posición 150 (en ángulos) en un "playtime" de 1250ms con los leds encendidos en verde.
        delay(1200);
        Serial.print("Get servo Angle:");
        Serial.println(Herkulex.getAngle(ID));            //Se recibe el ángulo del servo
        Serial.print("Get servo Position:");
        Serial.println(Herkulex.getPosition(ID));         //Se recibe la posición del servo
```

```
        Serial.print("Get servo Speed:");

        Serial.println(Herkulex.getSpeed(ID));          //Se recibe la
velocidad a la que se desplaza el servo

    }
```

Algunas de las funciones utilizadas en este breve código serán descritas en el siguiente apartado.

4.2.1.1. Funciones

En este apartado se explicarán algunas de las funciones usadas en el código para el control del servo motor que se observa en el apartado anterior.

La primera función que encontramos en el código es la encargada de empezar las comunicaciones entre la placa Arduino Mega y los servos a través del puerto serie 1:

```
    void  beginSerial1(long baud);
```

En esta función lo que se hace principalmente es asignar los pines de escritura y lectura de la placa Arduino Mega, al indicar Serial1, se señalan los pines 18 (transmisor) y 19 (receptor) a través de los cuales se realizaran las comunicaciones.

La siguiente función utilizada es *Herkulex.reboot(ID),* que como su propio nombre indica se encarga de resetear los valores del servo. En esta función se usa el envío de tramas explicadas en el apartado 4.1. Posteriormente, se inicializan los motores para que puedan comenzar a trabajar.

```
    void HerkulexClass::initialize()

    {

        delay(100);

// Se eliminan los errores en todos los servos. El valor es 0xFE.

        clearError(BROADCAST_ID);

        delay(10);

// se establece el valor ACK que solo dará respuesta a comandos CMD

        ACK(1);

        delay(10);
```

// Se activa el torque en todos los servos

torqueON(BROADCAST_ID);

}

Una vez terminados con las funciones que inicialicen las comunicaciones y ponen en marcha los servos, se explica la función que se ha utilizado para realizar las pruebas (movimientos) en el servo.

Herkulex.moveOneAngle(ID, pos, playT, LED);

Con esta función se manda al servo a la posición deseada en ángulos. El primer comando que se le manda a la función es la ID del servo, con la que se detecta a qué servo se dirige dicha orden de movimiento; el segundo comando es el de la posición a la que se envía el servo (en ángulos), al indicar está posición se tiene que tener en cuenta que el servo tiene una rango de movimiento de -160 hasta 160, ya que como se ha comentado previamente existen ángulos muertos a los que el servo no puede llegar; el tercer comando es el que indica el tiempo en el que se debe realizar el cambio de posición, y por último el cuarto comando indicará el color de los LEDs cuando realice el cambio de posición. A continuación se muestra una pequeña parte del código donde se puede ver las funcionalidades de dicha función.

void HerkulexClass::moveOneAngle(int servoID, float angle, int pTime, int iLed) {

if (angle > 160.0|| angle < -160.0) return;

int position = (int)(angle/0.325) + 512;

moveOne(servoID, position, pTime, iLed);

}

A su vez dentro de esta función se llama a la función moveOne, que es la que se encargará de dirigir el servo a la posición absoluta deseada (se detalla a continuación). Los parámetros que recibe esa función son: la identidad del servo, la posición (pero esta vez no en ángulos, *sino en la medida en la que trabaja el servo, la posición absoluta, indicada en el apartado 5.6*), utilizando la formula que se explica anteriormente para el cálculo de posición del servo, el tiempo de cambio de posición y el color de los LEDs.

void HerkulexClass::moveOne(int servoID, int Goal, int pTime, int iLed)

{

// Se comprueba que la posición a la que se envía el servo esta dentro de los márgenes permitidos

```
if (Goal > 1023 || Goal < 0) return;
if ((pTime <0) || (pTime > 2856)) return;

// Se define la posición a la que se envía al servo
int posLSB=Goal & 0X00FF;          //Se cogen los últimos 8 bits
int posMSB=(Goal & 0XFF00) >> 8;   // Se cogen los primeros 8 bits

//Se define el color del Led
int iBlue=0;  int iGreen=0;  int iRed=0;
switch (iLed) {
case 1:    iGreen=1;    break;
case 2:    iBlue=1;    break;
case 3:    iRed=1;    break;
}
int SetValue=iGreen*4+iBlue*8+iRed*16;

playTime=int((float)pTime/11.2);          // 8. Tiempo de ejecución
pSize = 0x0C;                             // 3.Tamaño del paquete
// 5. Se indica la acción que se quiere realizar (movimiento en este caso).
cmd   = HIJOG;
//Se incluyen los datos que se van a enviar en la trama (posición, velocidad, Id del servo y modo)
data[0]=posLSB;                   // 8. speedLSB
data[1]=posMSB;                   // 9. speedMSB
data[2]=SetValue;                 // 10. Mode=0;
data[3]=servoID;                  // 11. ServoID
```

```
        pID=servoID^playTime;
        lenghtString=4;                          // Tamaño de los datos
//6. Llama a la función Checksum1, donde se realizan las formulas indicadas en el apartado 4.1
        ck1=checksum1(data,lenghtString);
//7. Llama a la función Checksum2, donde Se realizan las formulas indicadas en el apartado 4.1
        ck2=checksum2(ck1);
//Se genera la trama que se va a enviar
        pID=servoID;
        dataEx[0] = 0xFF;                // Cabecero
        dataEx[1] = 0xFF;
        dataEx[2] = pSize;               //Tamaño del paquete
        dataEx[3] = pID;                 // ID del servo
        dataEx[4] = cmd;                 // Comando que determina la acción
        dataEx[5] = ck1;                 // Checksum 1
        dataEx[6] = ck2;                 // Checksum 2
        dataEx[7] = playTime;            // Tiempo de ejecución
//Se incluyen los datos que se desean enviar en la trama
        dataEx[8] = data[0];
        dataEx[9] = data[1];
        dataEx[10] = data[2];
        dataEx[11] = data[3];
//Se envía la trama
        sendData(dataEx, pSize);
      }
```

Una vez realizados los movimientos, el siguiente paso será conocer la posición, ángulo, velocidad,… y para ello se utilizaran las funciones *getSpeed, getPosition y getAngle*, las cuales a través del comando de lectura en RAM recibirán la información deseada.

4.3. Ubuntu (C/C++)

En este caso se ha decidido utilizar el sistema operativo Linux (Ubuntu) para desarrollar los algoritmos en C++ que se explicarán a continuación.

4.3.1. Código

Teniendo en cuenta las indicaciones que se pueden encontrar en el manual, se ha desarrollado un código en C++ para poder comandar en posición y leer los registros internos de los servomotores Herkulex. El código completo se encuentra en el Anexo A.

Dentro del código, lo primero que se encuentra es la inclusión de las librerías que se van a necesitar para el desarrollo de éste (en el anexo A pueden verse la librerías utilizadas). Asimismo, se definen las variables, estructuras generales que se necesitan y las diferentes funciones que se han generado y que serán explicadas en el siguiente apartado.

Una vez definidas las librerías, funciones y estructuras, se procede al main del código donde se realiza la prueba y adquisición de datos con el servo. Se mandará el servo a diferentes posiciones, a distintas velocidades, a la vez que se van almacenando los datos que se deseen en el archivo externo indicado anteriormente. Lo primero que se hace dentro del main es comprobar que el número de argumentos que entran por línea de comandos es correcto.

```
if(argc != 4)    {
```

// Argumentos: (1) Nombre del programa, (2) /dev/ttyUSB0, (3) servo ID, (4) Nombre del fichero externo

```
        cout << "ERROR: Número de argumentos invalido" << endl;

        error = 1;

}
```

Una vez comprobado que es correcto, se procede a abrir el puerto serie para poder establecer las comunicaciones con el servo.

```
else    {
```

// Apertura del puerto serie

```
        serialInterfaceHandler = open(argv[1], O_RDWR | O_NOCTTY |
O_SYNC);

        if(serialInterfaceHandler <= 0)         {

                error = 1;

                cout << "ERROR: No se pudo abrir el dispositivo USB." << endl;

        }

        else

                error = configureSerialInterface(serialInterfaceHandler);

        }
```

Y por último se deberá abrir el fichero externo (OutDataFile) donde se almacenaran los valores de la prueba y comprobar que no hay ningún error.

```
        outDataFile.open(argv[3]);

        if(outDataFile.is_open() == false)

                cout << "ERROR: No se pudo abrir el fichero externo" << endl;
```

Cuando se hayan realizado estas comprobaciones, y no haya ningún error, se realizará la prueba mediante el código que se muestra a continuación:

//Si no se ha producido ningún error de los mencionados, se accede a la ejecución de la prueba

```
        if(error == 0)

        {
```

//Se definen las variables que se van a utilizar

```
        float positions[4] = {30, 0, 90, 0};

        float playTimes[4] = {2, 2, 5, 5};

        ofstream outDataFile;

        SERVO_STATE servoState;

        struct timeval t0;
```

```
struct timeval t1;
// Se habilita el torque del servo
servoID = atoi(argv[2]);
setTorqueON(servoID, serialInterfaceHandler);
// Se recoge el tiempo exacto antes de comenzar la simulación
gettimeofday(&t0, NULL);
/**************** SECUENCIA DE SIMULACIÓN ****************/
for(int n = 0; n < 4; n++)
{
// Se manda al servo a las posiciones deseadas a la velocidad deseada. Se le envía posición, velocidad, ID del servo y puerto serie.
moveServo(positions[n], playTimes[n], servoID, serialInterfaceHandler);
// Se obtiene el dato del tiempo actual
gettimeofday(&t1, NULL);
t = (t1.tv_sec - t0.tv_sec) + 1e-6*(t1.tv_usec - t0.tv_usec);
double taux1 = t;
// Comienza la secuencia de escritura en fichero externo
do
{
//Se hace una lectura en la memoria RAM.
readRAMRegister(servoID, serialInterfaceHandler, &servoState);
//Se actualiza el tiempo
gettimeofday(&t1, NULL);
t = (t1.tv_sec - t0.tv_sec) + 1e-6*(t1.tv_usec - t0.tv_usec);
// Se escribe la información en el fichero externo cada 0.1s
```

```
                    outDataFile << t << "\t";
                    outDataFile << servoState.position << "\t";
                    outDataFile << servoState.ref << "\t";
                    outDataFile << servoState.pwm << "\t";
                    outDataFile << servoState.temperatura<< "\t";
                    outDataFile << servoState.voltage << "\t";
                    outDataFile << endl;   // Salto de linea
// Esperar un segundo después del play time
                } while(t < taux1 + playTimes[n] + 1);
            }
        }
```

Como se observa en el código y en las explicaciones incluidas en él, se ha realizado una prueba en la que se realizan distintos movimientos a distintas velocidades en el servo a través de la función *movesrvo* (se explica en el siguiente apartado), y posteriormente se hace una lectura en la memoria RAM, a través de la función *readRAMRegister* (se explica en el siguiente apartado), para finalmente guardar toda la información deseada en el archivo externo OutDataFile.

4.3.1.1. Funciones

Una vez visto el main del código, se procede explicar las funciones desarrolladas, algunas de ellas mencionadas en el apartado anterior. Las funciones que se han definido son las siguientes:

- *void* ***setTorqueON****(uint8_t servoID, int serialInterfaceHandler);*

Esta función se utiliza para activar el torque de los servomotores, y para ello se le envía la ID del servo al que se desea activar el servo y el puerto serie al que está conectado.

- *void* ***moveServo****(float position, float playTime, uint8_t servoID, int serialInterfaceHandler);*

Con esta función se realizan los movimientos en el servo y para ello es necesario conocer la posición a la que se desea enviar el servo, el tiempo de ejecución, la ID de

este y el puerto serie al que está conectado. Esta función se explicará con más detalle a continuación

- *void **readRAMRegister**(uint8_t servoID, int serialInterfaceHandler, SERVO_STATE * servoState);*

A través de esta función se realizan las lecturas en la memoria RAM del servo para conocer los datos deseados. Para poder realizar la lectura se ha de enviar la ID del servo, el puerto al que está conectado y la estructura donde almacenar los valores que se desean conocer. Esta función se explicará con más detalle a continuación

- *int **configureSerialInterface**(int serialInterfaceHandler);*

En esta función se realiza la apertura del puerto serie y se inicializan los valores de configuración deseados, como el baudrate. Para realizar esta configuración será necesario conocer el puerto serie.

- *void **computeChecksum**(char * packet, uint8_t packetSize);*

En esta función se realizan las comprobaciones necesarias para detectar errores en el servo. Se hace siguiendo las pautas marcadas en el manual del servo. Para realizar los cálculos del checksum se debe conocer los valores que se pretenden enviar en la trama y el tamaño del paquete.

- *int **sendData**(char * data, uint8_t dataLength, int serialInterfaceHandler);*

A través de esta función se realizan los envíos de los paquetes (tramas) que se desean enviar al servo. Para realizar estos envíos es necesario conocer todos los campos que van a ser enviados en el paquete, el tamaño de este y el puerto al que está conectado el servo.

Dentro de estas funciones se van a explicar con más detalle las funciones de *moveServo y readRAMRegister,* encargadas de la simulación y de la lectura de los datos.

Para realizar los movimientos deseados en el servo:

void moveServo(float position, float playTime, uint8_t servoID, int serialInterfaceHandler)

{

//Inicialización de las variables utilizadas en la función

uint8_t packet[16];

float playTimeByte = 0;

```
    float positionAux = 0;
    uint16_t position2Bytes = 0;
    int error = 0;
// Se completan los campos de la trama
    packet[0] = 0xFF;          // Cabecero
    packet[1] = 0xFF;
    packet[2] = 0x0C;          // Tamaño de la trama
    packet[3] = servoID;       //ID del servo
    packet[4] = 0x05;          // Comando I_JOG

/* ---------- Se envían los datos de la simulación recibidos en la función ----------*/
//Se convierten los grados en posición absoluta
    positionAux = 512 + position / 0.325;
    position2Bytes = (uint16_t)positionAux;
//Se almacenan los últimos 8 bits
    packet[7] = (uint8_t)(0x00FF & position2Bytes);
    position2Bytes = position2Bytes >> 8;
//Se almacenan los primeros 8 bits
    packet[8] = (uint8_t)(0x00FF & position2Bytes);
//Se activa el control de posición y se enciende el LED en verde
    packet[9] = 0x04;
    packet[10] = servoID; // ID
// Se calcula el play time teniendo en cuenta que el servo trabaja en ticks (11.2ms).
Se debe asegurar que el play time esta en el rango 1-255
    playTimeByte = playTime / 0.0112;
    if(playTimeByte < 1)
```

```
            playTimeByte = 1;
        if(playTimeByte > 255)
            playTimeByte = 255;
        packet[11] = (uint8_t)playTimeByte;
// Se comprueba que no hay ningún error a través del cálculo de checksum
        computeChecksum((char*)packet, packet[2]);
// Finalmente se envía la trama
        if(error == 0)
            sendData((char*)packet, packet[2], serialInterfaceHandler);
    }
```

Y por último se va a explicar cómo se realizan las lecturas en la memoria RAM del servo, para poder conocer los datos que deseemos. Esta función se divide principalmente en dos partes, una primera parte en la que habrá que enviar una trama al servo indicando que se desea hacer una lectura en la memoria RAM, y una segunda en la que se realizará la lectura.

```
    void readRAMRegister(uint8_t servoID, int serialInterfaceHandler, SERVO_STATE *
servoState)
    {
//Inicialización de variables utilizadas en la función
        char packet[16];
// Para almacenar los datos recibidos de la RAM a través de la función read
        char buffer[1024];
        CharArray2Int16 aux;
//Datos que se quieren recibir del servo
        float position = 0; float voltage = 0; float temperature = 0;
        float speed = 0; float positionRef = 0; float pwm = 0; float tick = 0;
```

/*-------- Se genera la trama que se enviará al servo, a través de la cual se pedirá realizar una lectura en la RAM ----------*/

packet[0] = 0xFF; ***// Cabecero***

packet[1] = 0xFF;

packet[2] = 0x09; ***// Tamaño del paquete***

packet[3] = servoID; ***// ID del servo***

packet[4] = 0x04; ***//Lectura de RAM***

// Dirección inicial de la que leer (Registro del voltaje en la RAM). A partir de esta dirección se comenzara la lectura en la RAM del servo

packet[7] = 54;

//Número de Bytes que se desean leer (hasta Absolute Goal Position) desde la posición que se comienza la lectura (packet[7])

packet[8] = 16;

// Se realiza el cálculo de erroes a través del checksum

computeChecksum((char)packet, packet[2]);*

// Se envía la trama

sendData((char)packet, packet[2], serialInterfaceHandler);*

/*-------- Una vez enviada la trama se realizan la lectura de datos en la RAM del servo a través de la función read ---------*/

//Se almacenan en el buffer todos los datos de la memoria RAM a través de la función read

bytesReceived = read(serialInterfaceHandler, buffer, 1023);

//Se calcula el voltaje realizando la conversión ADC indicada en el manual. Se empieza a partir de la posición 9 ya que los otros campos hasta el 8 son siempre los mismos por defecto

*voltage = 0.074*buffer[9];*

//Se recibe la temperatura

temperature = buffer[10];

*tick = 0.0112*buffer[12];* ***//se convierten los ticks en sg***

```
//Se calcula la posición absoluta en grados, tal y como se indica en el apartado 5.6
        position = 0.325*(buffer[15] + 256*buffer[16] - 512);
//Se calcula la velocidad de movimiento del servo como un init16_t (entero de 16 bits), ya que ocupa 2 bytes.
        aux.MSB_LSB[1] = buffer[18];
        aux.MSB_LSB[0] = buffer[17];
        speed = 29.09*aux.value;                //Se mueve a 29.09 grados/sg
// Se calcula el PWM como un init16_t (entero de 16 bits) ya que ocupa 2 bytes
        aux.MSB_LSB[1] = buffer[20];
        aux.MSB_LSB[0] = buffer[19];
        pwm = 9.76562e-4*aux.value;             //(1/1023)*Aux.value
//Se calcula la posición referencia en grados
        positionRef = 0.325*(buffer[23] + 256*buffer[24] - 512);
//Se almacenan los valores calculados en la estructura SERVOSTATE creada al inicio del código
        servoState->position = position;
        servoState->ref = positionRef;
        servoState->speed = speed;
        servoState->pwm = pwm;
        servoState->temperature = temperature;
        servoState->voltage = voltage;
    }
```

5. Modos de funcionamiento

En este apartado se explican las principales características y parámetros configurables de los servo motores Herkulex. Se podrá encontrar una explicación detallada de cada uno de estos parámetros y de cómo funcionan en los servomotores Herkulex.

5.1. Perfil de velocidad trapezoidal

Para el segmento de aceleración de un perfil típico trapezoidal, como se puede ver en la figura que se muestra a continuación**,** el movimiento comienza desde una posición de parada o desde un movimiento previo y sigue una rampa de aceleración indicada hasta que la velocidad alcanza la velocidad deseada para el movimiento. El movimiento continúa a la velocidad indicada por un periodo de tiempo hasta que el controlador determina que es tiempo de comenzar el segmento de desaceleración, y desciende la velocidad del movimiento para detenerse exactamente en la posición deseada. En caso de que el movimiento sea lo suficientemente corto, y el punto de desaceleración llegue antes de que se haya terminado la aceleración, el perfil tomara una forma triangular pudiendo no haber llegado a la velocidad deseada.

Aplicando este perfil a los servos, una vez que el servo recibe un comando de movimiento, se crea automáticamente un perfil de velocidad de tipo trapezoidal, como se muestra en el diagrama de abajo, para controlar la posición. Con el servo funcionando según el perfil de aceleración/deceleración se suprimen las vibraciones producidas por la aceleración y desaceleración repentina, como se encuentran en el perfil de velocidad de tipo cuadrado, y aumenta la eficiencia energética, ya que el movimiento se realiza de manera más suave. El servo elige el perfil de velocidad de tipo trapezoidal por defecto, pero el perfil podría ser cambiado al perfil que se desee (trapezoidal, cuadrado o triangular).

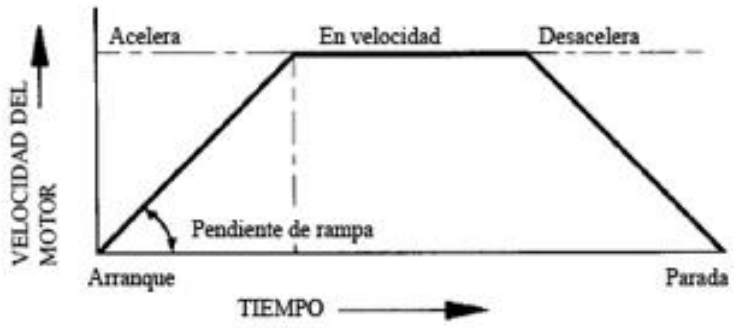


Figura 25: Perfil de velocidad trapezoidal

En el caso de los servomotores Herkulex, estos perfiles son controlados a través de los comandos de aceleración (**acceleration ratio**) y desaceleración (**deceleration ratio**) y pueden ser modificados. Como se observa en la figura que se muestra a continuación, se genera un perfil de velocidad trapezoidal ante una orden de movimiento en el servo, que provocará una aceleración inicial para desplazar el servo y una desaceleración final para detener a este. Se pueden apreciar 3 gráficas: en la línea azul se observa como cuando se reduce la proporción de aceleración se generan cambios drásticos en la velocidad, generando vibraciones en el servo; por otro lado, si se aumenta esta proporción de aceleración, se generará un aumento lento de la velocidad mediante un movimiento suave del servo, pero también se producirá un cambio drástico en el pico de velocidad, como se observa en la línea roja; y por último, en la línea de color amarillo se muestra el movimiento trapezoidal que sería el desplazamiento ideal.

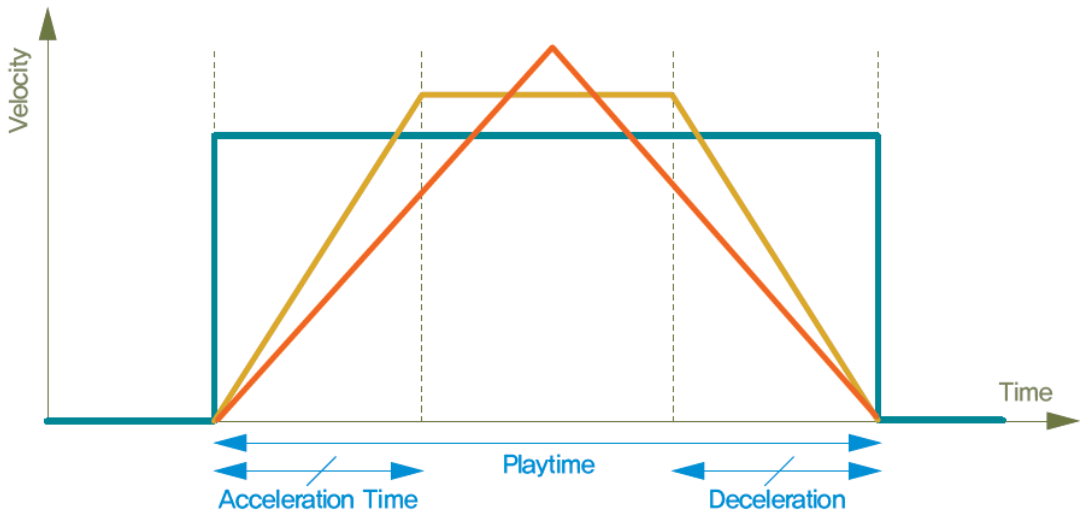


Figura 26: Tiempo de aceleración y desaceleración del servo Herkulex DRS 0101

5.2. Protecciones del servo

Una de las principales características de los servos Herkulex son sus sistemas de protección internos:

- Sensores de temperatura: Estos sensores permiten monitorizar la temperatura del motor y del circuito para indicarnos un error si la temperatura de estos es sobrepasada.
- Protección de alimentación: El motor indicará un error en caso de que la tensión del servo sea superior o inferior a la requerida, previniendo así de posibles sobrecargas que afecten al funcionamiento del servo.
- Protección contra sobrecargas manuales: Este error salta cuando se aplican fuerzas externas al servo superiores a las indicadas (este parámetro es

configurable y se puede indicar el porcentaje de detección que se quiere aplicar al servo) impidiendo que este puede ser manipulado. Esta funcionalidad es de gran utilidad a la hora de evitar que el servo sea manipulado manualmente y por consiguiente pueda ser dañado. Así se consigue prolongar su vida.

Además de todos estos sistemas de protección, los servomotores Herkulex son capaces de detectar 7 diferentes tipos de errores (Vin, Temperatura, Posición limite, Paquetes de envío invalido, Sobrecarga, drivers y registro en memoria EEPROM), que serán indicados mediante un parpadeo de LED en rojo.

5.3. Control PID

Un controlador PID (**P**roporcional, **I**ntegral y **D**erivativo) es un método de control por retroalimentación. Este calcula una señal de control a partir de la desviación o error entre un valor medio y un valor deseado.

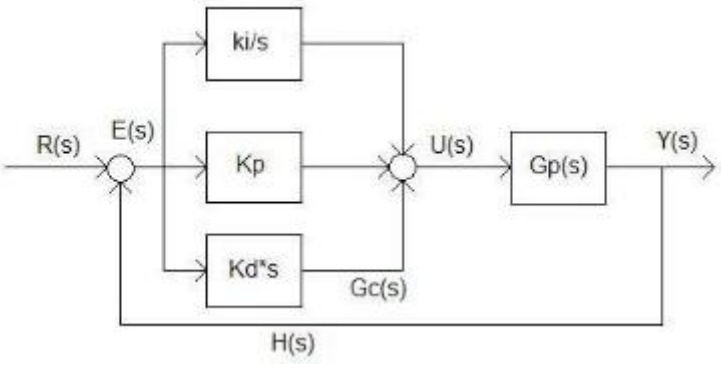


Figura 27: Control PID tradicional.

- **Proporcional** (depende del error actual): El modo proporcional consiste en el producto entre la señal de error y la constante proporcional para lograr aproximar el error en estado estacionario a cero, pero en la mayoría de los casos, estos valores solo serán óptimos en una determinada porción del rango total de control, siendo distintos los valores óptimos para cada porción del rango (sobreoscilación). La respuesta proporcional es la base de los tres modos de control. Si los otros dos, control integral y control derivativo están presentes, éstos son sumados a la respuesta proporcional. “Proporcional” significa que el cambio presente en la salida del controlador es algún múltiplo del porcentaje del cambio en la medición.
- **Integral** (depende del error futuro)**:** El término Integral tiene como propósito disminuir y eliminar el error en régimen permanente que no puede ser eliminado completamente por el modo proporcional (suponiendo que el sistema es de orden cero). El control integral actúa cuando hay una desviación

entre la variable y el punto deseado, integrando esta desviación en el tiempo y sumándola a la acción proporcional. El *error* es integrado, lo cual tiene la función de promediarlo o sumarlo por un período determinado; luego es multiplicado por una constante **Ki**. Posteriormente, la respuesta integral es añadida al modo Proporcional para formar el control P + I con el propósito de obtener una respuesta estable del sistema sin error estacionario.

- **Derivativo** (depende del error pasado): El modo de control derivativo aparece cuando hay un cambio en el valor absoluto del error (si el error es constante, solamente actúan los modos proporcional e integral). El *error* es la desviación existente entre el punto de medida y el valor deseado, o "*Set Point*".
  Dentro de un PID, el término derivativo permite aumentar la velocidad de convergencia de la señal controlada hacia la referencia deseada.
  Se deriva con respecto al tiempo y se multiplica por una constante **Kd** y luego se suma a las señales anteriores (P+I). Es importante adaptar la respuesta de control a los cambios en el sistema, ya que una mayor derivativa corresponde a un cambio más rápido y el controlador puede responder acordemente.

La ecuación típica, en el dominio del tiempo para un controlador PID es la siguiente:

$$PID(s) = Kp * e(t) + Ki * \int_0^t e(\tau)d\tau + Kd * \frac{de(t)}{dt}$$

Siendo e(t) la señal del error

En el área de los servomotores el control PID es el algoritmo de control más utilizado a la hora de regular la posición de éstos [17], y los servos Herkulex permiten realizar modificaciones en estos campos. En el apartado 6 podrán verse los resultados de las distintas pruebas que se han realizado variando los valores de las ganancias.

5.4. Control VOR (Velocity Over Ride).

Una de las opciones que ofrecen estos servomotores es la posibilidad de ir variando la velocidad a la que se mueve el servo sin tener que detenerse ante la entrada de un nuevo input. Usando el control del VOR (Velocity Over Ride) se puede continuar con el movimiento del servo, aunque éste no haya llegado a la posición indicada antes de recibir una nueva orden de desplazamiento. El servo no se detendrá y continuará su desplazamiento a la nueva posición a la velocidad indicada en la nueva orden.

Para el control de un brazo robótico esta opción puede ser de gran utilidad, ya que es muy posible que se necesite realizar un movimiento continuo del brazo sin que la velocidad llegue a cero. En el apartado 6 podrán verse distintas pruebas que se han

realizado y así entender cómo afecta el control VOR al funcionamiento del servo, de la misma manera que se verá como en versiones más avanzadas de los servos Herkulex (DRS 0402 y 0602) esta funcionalidad puede ser deshabilitada.

5.5. Torque.

El torque es una medida que indica cuanta fuerza es necesaria aplicar a un objeto para hacer que este rote. Como se puede ver en la siguiente figura, el objeto gira en torno a un eje (que será el punto central) 'O'. Dicho objeto estará situado a una distancia 'r' (distancia desde el eje hasta el lugar donde se aplica la fuerza) y se le aplicara una fuerza 'F' para realizar la rotación.

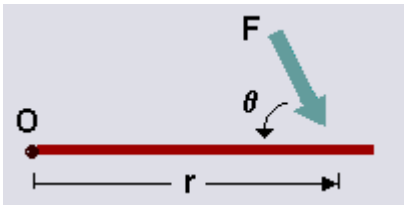


Figura 28: Definición del torque

Cuya fórmula queda definida como:

$$Torque = r * F * sen(\theta)$$

Aplicándolo al campo de los motores, el par motor o torque es el momento de fuerza que ejerce un motor sobre el eje de transmisión de potencia o, dicho de otro modo, la tendencia de una fuerza para girar un objeto alrededor de un eje, punto de apoyo, o de pivote. El torque del servo motor indica el peso que puede mover el servo en función de la distancia.

En los Herkulex DRS-0101 el control de los diferentes comandos solo funcionará cuando el torque este habilitado (Torque ON) y por defecto estos comandos no funcionarán cuando el freno este habilitado (Break ON). Las articulaciones del servo solo podrán ser manipuladas manualmente cuando se libere el torque (Torque Free).

En caso de que se produjera un error durante las pruebas, los LEDs parpadearían en rojo y el torque quedaría en modo Torque OFF, manteniéndose en este estado hasta que se le habilitara el Torque ON de nuevo.

5.6. Posición absoluta

Este parámetro muestra la posición actual del servo (sin calibrar) mediante datos en bruto. La relación entre los datos brutos y los grados es la siguiente:

$$Grados = (Datos\ brutos\ - 512)\ \times 0{,}325$$

En la siguiente gráfica se muestra los rangos de trabajo del servo cuando está en el modo de control de posición, ya que como se ha mencionado anteriormente, cuando está en modo de control de velocidad el giro puede ser continuo. Como se puede observar, el servo no puede llegar a determinadas posiciones (ángulos muertos), ya que peligraría su correcta funcionalidad.

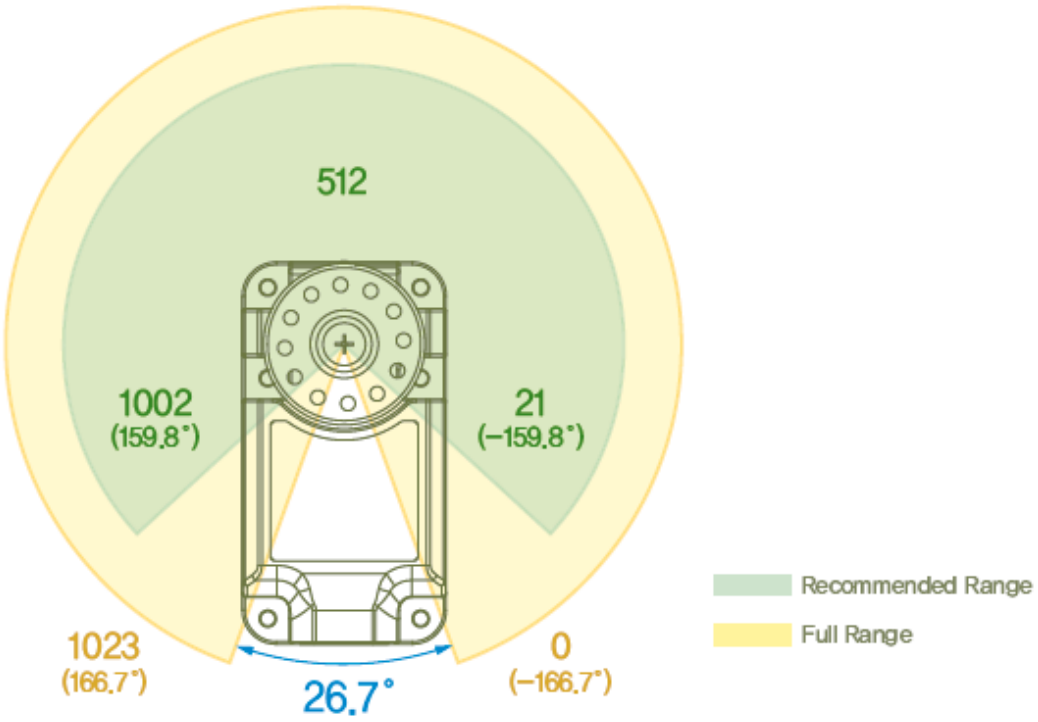


Figura 29: Posición absoluta

5.7. Posición diferencial

Este parámetro muestra una estimación de la velocidad angular a partir de la diferenciación de la medida de posición y en la siguiente ecuación vemos como se calcula:

$$PD = \frac{PA(t = k) - PA(t = (k - 1))}{tick}$$

Donde PD es posición diferencial, PA posición absoluta, k el instante actual, y tick es el reloj interno del servo y el intervalo de tiempo en el que trabaja es **11,2 ms**. Esta fórmula muestra cómo funciona el servo internamente.

A través de este campo se realizaran los cálculos de velocidad de desplazamiento del servo, como se puede observar en los códigos desarrollados en C++, explicados en la sección 4.3.

5.8. Posición indicada

Cuando se indica la posición deseada en bruto (**Absolute goal position)** el servo marca automáticamente la trayectoria más adecuada usando el perfil de velocidad indicado. La trayectoria deseada de posición absoluta (**Absolute desired trajectory position)** son las posiciones intermedias que se pretenden alcanzar para llegar a la posición final que se desee. En la siguiente gráfica se puede ver el funcionamiento del servo Herkulex cuando se le indica la posición deseada, y como marca puntos intermedios para definir la trayectoria deseada.

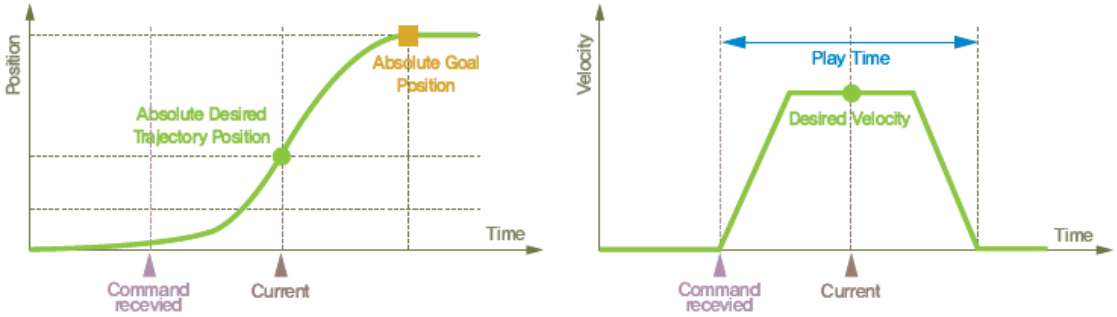


Figura 30: Trayectoria de posición indicada.

5.9. Tiempos de envío

Los tiempos de envío a los que trabajan los servomotores Herkulex se calculan teniendo en cuenta el número de bytes (cantidad de bits que se pretenden enviar) y la velocidad de transmisión a la que se trabaja (Baudrate). En la siguiente formula se puede observar la relación existente:

$$T = \frac{Bytes * 8}{Baudrate}[s]$$

En el caso de los servos DRS-0101 se trabaja por lo general con un baudrate de 115200bps, aunque el límite este de estos este a 0,67Mbps.

En el caso de los servos DRS-0101 se puede mostrar un breve ejemplo para ver cómo funcionan los tiempos de envío. Se muestra el ejemplo en el que se quiera enviar una orden de movimiento al servo (I_JOG): tendrá que desplazarse a la posición 512 (en datos brutos. Equivale a la posición 0º), con el LED encendido en verde y a una

velocidad de 60 (60*11,2ms = 672ms). Este paquete estaría compuesto de 11 bytes, por lo que siguiendo la formula indicada quedaría:

$$T = \frac{11 * 8}{115200}[s] = 0{,}764[ms]$$

6. Identificación experimental

6.1. Arduino

A través de la placa Arduino Mega, se ha desarrollado un breve código (Sección 4.2), apoyado por las librerías encontradas, para manipular los servomotores y en este apartado se podrán ver los resultados de las pruebas realizadas y la forma en la que se representan estos resultados.

6.1.1. Adquisición de datos

Finalmente se explica cómo se ha llevado a cabo las adquisición de los datos y como estos han sido mostrados en pantalla. Como se muestra en el apartado 4, el empleo de los microcontroladores Arduino se ha enfocado principalmente a la visualización de datos por pantalla en formato numérico. A través de las distintas funciones explicadas anteriormente se han ido adquiriendo una serie de datos. A través de las funciones *getSpeed, getPosition y getAngle* se recolectaran los datos de interés del servo.

6.1.2. Pruebas realizadas

Debido a la poca versatilidad que ofrecen los microcontroladores Arduino, las pruebas que se han realizado han estado enfocadas principalmente a la adquisición de datos (outputs más significativos). Se han recogido los datos más relevantes que se pueden obtener de estos servomotores: ángulo, posición, temperatura, velocidad (tiempo de respuesta), etc. En la siguiente figura se puede observar la manera en la que se muestran los resultados (en este caso se muestra el movimiento, en ángulos, que se desea realizar y el ángulo en el que se encuentra una vez realizado el desplazamiento).

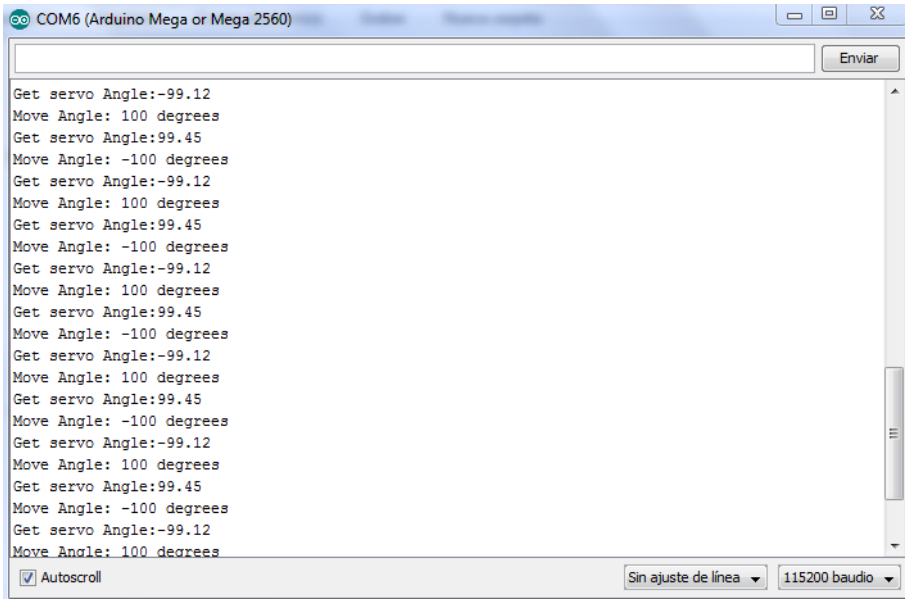


Figura 31: Resultados Arduino

En la siguiente figura se muestra un nuevo resultado, pero esta vez con más valores de salida, y con más ordenes de entrada. Se le ordena al servomotor a desplazarse a diferentes posiciones a diferentes velocidades, mostrando por pantalla la posición, ángulo y velocidad que tiene en cada momento:

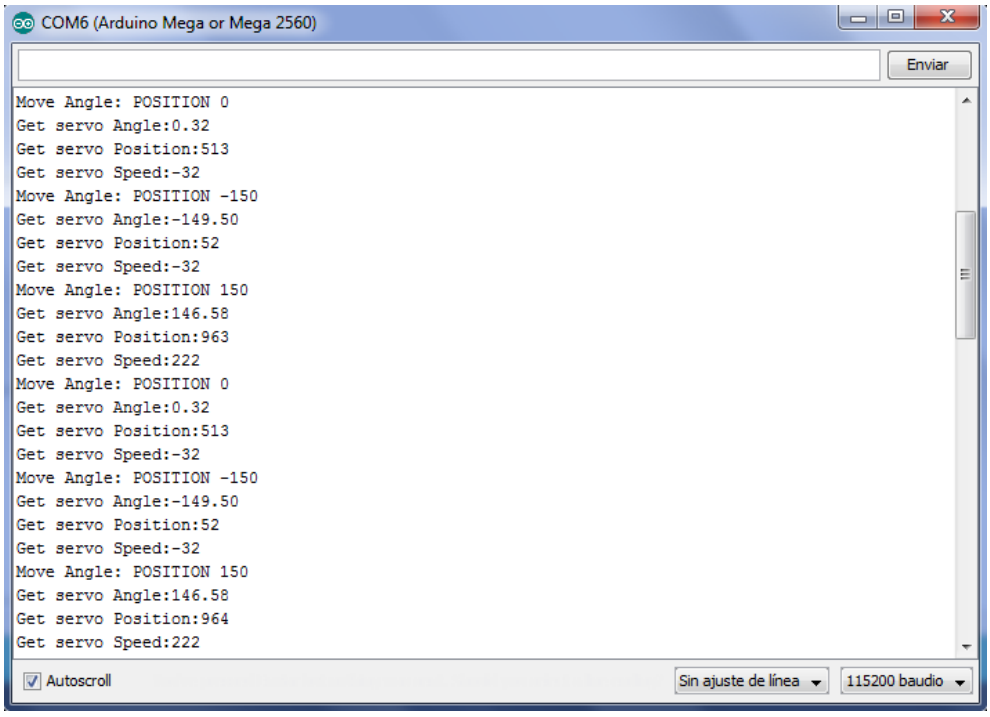


Figura 32: Representación de la posición y velocidad en la interfaz de ARDUINO.

6.2. Ubuntu (C/C++)

En este apartado se mostrarán los resultados obtenidos tras las pruebas realizadas a través del sistema operativo Ubuntu. Utilizando el código desarrollado en C++, explicado en el apartado 4.3, se han realizado varias pruebas, obteniendo datos que han sido representados mediante el uso del software Matlab. En los siguientes apartados se puede ver los resultados de una prueba en la que se ha generado una orden de movimiento del servo a distintas posiciones (30º, 0º, 90º, 0s) a distintos play time (2s, 2s, 5s, 5s). Se mostrará cómo y dónde se han recogido esos datos y cómo estos han sido representados.

6.2.1. Adquisición de datos

En este apartado se muestra cómo los distintos datos de la prueba realizada han sido recogidos en un fichero externo. Los datos que se han almacenado en el fichero externo (en la explicación del código del apartado 4.3.1 puede verse como se realiza la escritura de datos en el archivo externo) son los siguientes:

- Tiempo: Se recolectan datos cada 100 ms.
- Posición: Medida en ángulos
- Posición de referencia: Medida en ángulos.
- PWM: Amplitud de la señal PWM (PWM entre 1 y -1)
- Tensión
- Temperatura

6.2.2. Resultados experimentales

Una vez se hayan recolectado los datos deseados, se realiza una representación de estos a través del software Matlab. Lo primero que deberá hacerse es cargar el archivo.txt, con los datos de la prueba, en el software. Una vez esté el archivo cargado, habrá que asignar los valores de cada columna a una variable para poder realizar su representación:

Command Window

```
>> data = load('OutDataFile.txt');
Tiempo = data(:,1);
Posicion = data(:,2);
PosicionRef = data(:,3);
PWM = data(:,4);
Voltaje = data(:,5);
Temperatura = data(:,6);
```

Figura 33: Carga de datos en Matlab para representación.

Con los datos ya cargados en Matlab, solo quedará realizar las representaciones oportunas. En la siguiente gráfica se puede observar cómo la prueba se ha realizado de manera correcta. Se puede ver los resultados de las variables de posición (30º, 0º, 90º, 0º), posición de referencia, tensión y Temperatura:

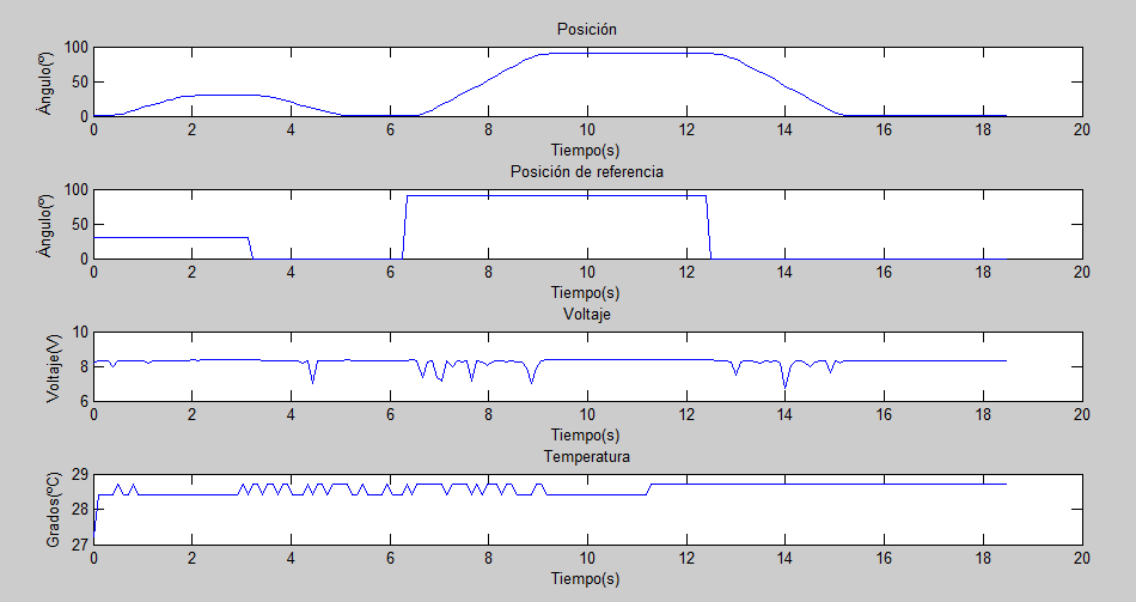


Figura 34: Representación en Matlab

Por último, en la siguiente figura se muestran los resultados obtenidos en la señal PWM, y se puede ver como ésta se activa cada vez que hay una nueva orden de movimiento.

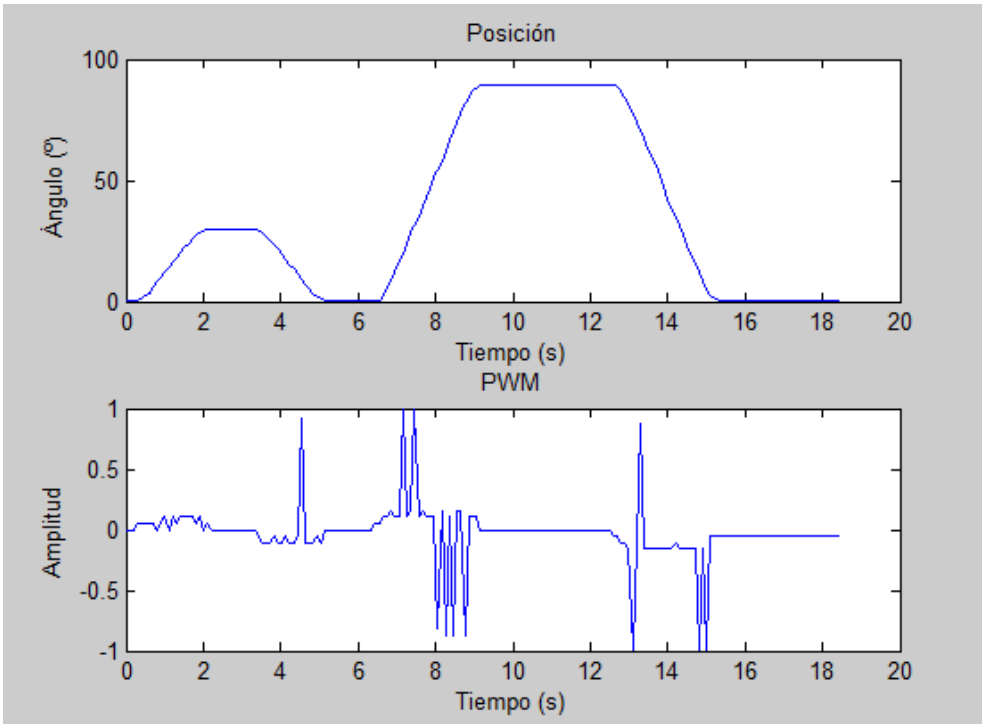


Figura 35: Representación en Matlab de señal PWM

6.3. Herkulex Manager

En este apartado se presentan las pruebas realizadas a través del Software Herkulex Manager. Se han realizado distintas pruebas de las distintas funcionalidades de estos servomotores que serán mostradas en los siguientes apartados.

6.3.1. Adquisición de datos

A la hora de realizar la adquisición de datos este apartado es diferente al de los apartados de C++ y Arduino, ya que en este caso no es necesario generar ningún código para la recolección de los datos que se deseen, aunque el mismo software ofrece la posibilidad de recoger todos los datos del experimento en un fichero txt para posteriormente poder trabajar con esos datos.

Como se observa en la siguiente imagen, en la pestaña de motor test se pueden realizar estas pruebas, cuyos resultados quedaran recolectados en un fichero que podrá ser guardado:

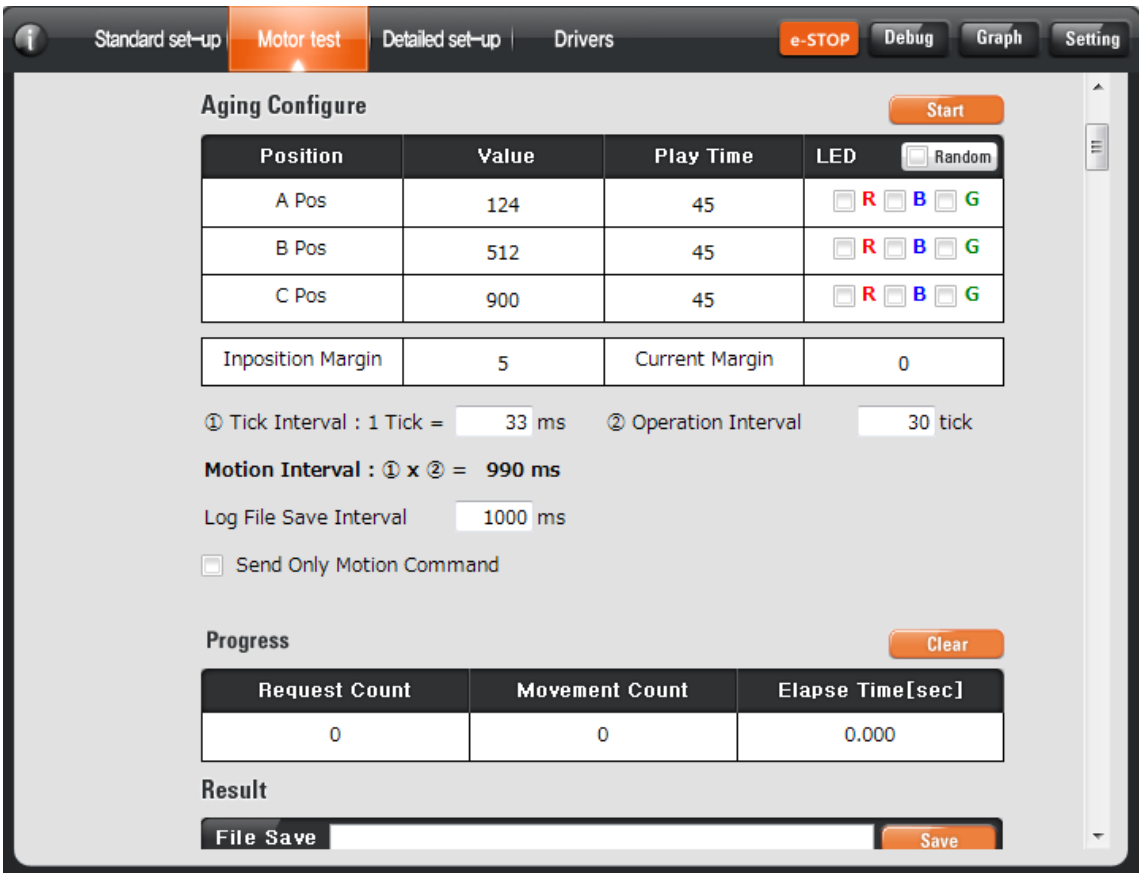


Figura 36. Generación de archivos Herkulex Manager

Estos experimentos permiten recolectar datos en los intervalos de tiempo que se desee.

6.3.2. Pruebas realizadas

En esta sección se pueden observar las distintas pruebas que se han realizado sobre las distintas funcionalidades descritas en el manual del servo, a través del software Herkulex Manager. Se han realizado las pruebas sobre las siguientes funcionalidades (se pueden ver los resultados de estas en los siguientes apartados):

- Señal PWM variando el play time
- Evaluación de modos de control con Velocity Over-Ride
- Control PID, modificando las ganancias

6.3.2.1. Señal PWM con diferentes velocidades y tiempos de ejecución

En este apartado se han realizado distintas pruebas variando los tiempos de ejecución (play time) para ver cómo responde el servo ante órdenes de movimiento en tiempos muy reducidos. Al observar la señal PWM del servo, se puede ver cómo afecta al torque de este, ya que según el manual del DRS 0101 el valor del PWM permite estimar el Torque en datos brutos, debido a que el torque es proporcional a la corriente y esta (en promedio) a su vez es proporcional al duty cicle del PWM.

A través del software Herkulex Manager se pueden realizar modificaciones en los máximos y mínimos a los que puede llegar la señal PWM (por defecto el mínimo será 0 y el máximo 1023), lo cual permitirá limitar o aumentar la señal en función de lo que se desee. La saturación de la señal PWM a un valor máximo es recomendable para evitar que el servo se dañe por sobrecalentamiento debido a sobrecarga. El umbral mínimo del servo reduce el consumo al evitar picos de corriente asociados a cambios de dirección del motor. De la misma manera se podrán configurar tanto los valores de offset del PWM, como el umbral de sobrecarga de la señal.

En las siguientes gráficas se muestra el PWM con distintos tiempos de movimiento (play time), así como con distintos tiempos de adquisición de datos (request time). Se puede observar que la señal PWM (en azul), se ajusta a la aceleración del servo (señal amarilla) y tiene una amplitud de señal mayor mientras mayor sea la velocidad de desplazamiento del servo.

En esta primera figura se puede observar el resultado con misma velocidad (misma aceleración) de movimiento en cada uno de sus desplazamientos, por lo que se puede ver que la señal PWM es muy similar en todos los desplazamientos.

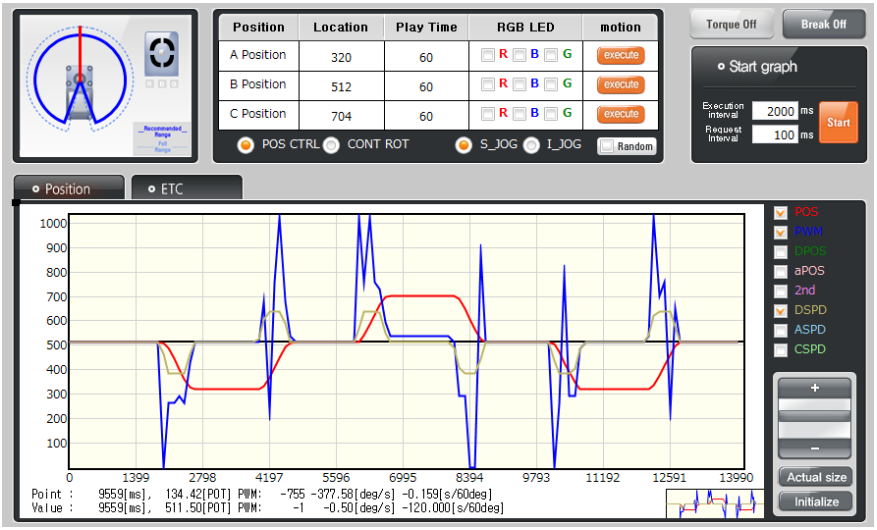


Figura 37: Señal PWM

En la figura que se muestra a continuación se han variado las posiciones a las que se envía al servo, la velocidad a la que se desplaza, y el tiempo de adquisición de datos (en el experimento anterior se recibían datos cada 100ms y ahora se reciben cada 10ms), por lo que como se puede observar en la figura el PWM varía mucho de un desplazamiento a otro, resaltando el desplazamiento a la posición 120 (en datos brutos), con un play time de 40ms.

Figura 38: Señal PWM con variaciones en la velocidad de desplazamiento.

6.3.2.2. Control del VOR (Velocity over ride)

En las gráficas que se muestran a continuación se puede observar como gracias al control del VOR de estos servomotores se pueden realizar cambios en los

movimientos de los servos sin necesidad de que este se detenga, impidiendo de esta manera que se produzcan movimientos bruscos a la hora de trazar las trayectorias deseadas. En la siguiente gráfica (figura 40) se puede observar el caso normal de cómo el servo acude a la posición indicada y se detiene **(línea amarilla → Velocidad** y **línea roja → Posición del servo)**, mientras que en la figura 41 se observa cómo, en el caso de que el servo no llegue a la posición deseada antes de que finalice el desplazamiento y reciba una nueva orden el servo acudirá a esta nueva posición sin necesidad de detenerse.

Figura 39: Funcionamiento normal del modo operacional con velocity over ride (se detiene al llegar a la posición).

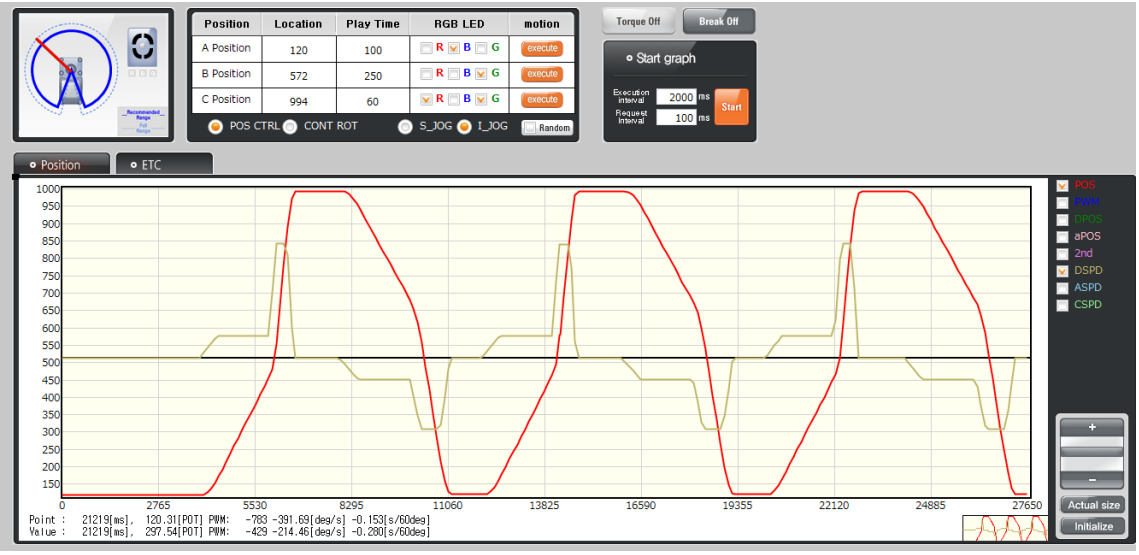


Figura 40: VOR habilitado (no llega a la posición deseada y continua sin detenerse ante la nueva orden)

En las versiones más avanzadas de esto servos, como pueden ser DRS-0402 o DRS-0602, existe la posibilidad de desactivar el VOR, obligando al servo a detenerse aunque no haya cumplido la orden de llegar al destino indicado, para, posteriormente,

realizar la nueva orden en caso de que no sea de nuestro interés utilizar esta funcionalidad.

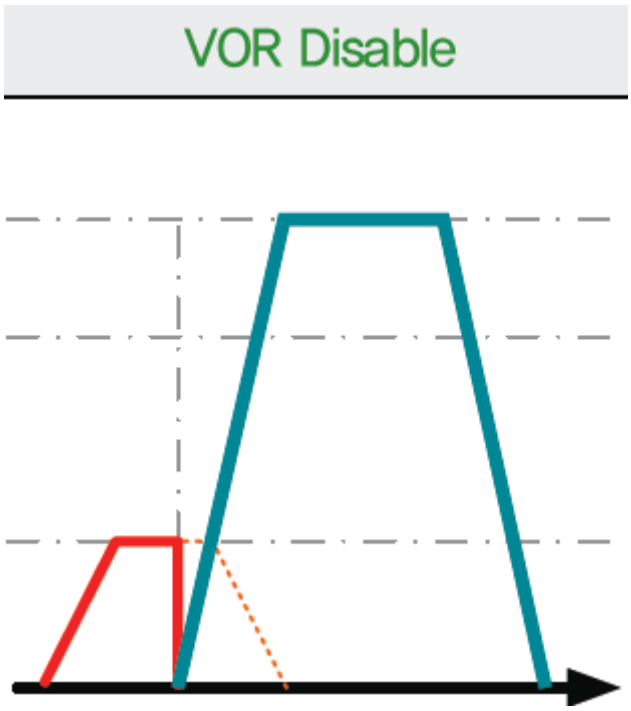


Figura 41: Perfiles de velocidad correspondientes a dos referencias de posición cuando el VOR está deshabilitado.

Esta funcionalidad (VOR enabled) es de gran importancia para conseguir trayectorias suaves en un brazo articulado, sin tener picos de aceleración que puedan afectar a la estabilidad de la plataforma aérea.

6.3.2.3. Control PID

En este apartado se muestran los resultados de las distintas pruebas realizadas en el controlador, variando los valores de cada una de las ganancias (Proporcial, Derivativa e Integral). Estos valores se modificarán en la pestaña de "Stardard set-up" del Herkulex Manager Software, como se puede ver en la siguiente figura. Los valores que se observan son los que el manual del servo Herkulex DRS 0101 indica por defecto. Durante las pruebas que vienen a continuación se irán variando estos valores para ver cómo responde el servomotor ante estas variaciones.

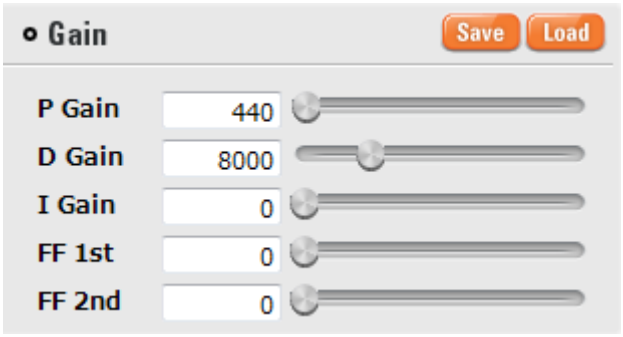


Figura 42. Valores por defecto de las ganancias del controlador PID del servo.

La gráfica que se obtiene con estos valores del PID es la siguiente. En la figura se puede observar como se ha generado un experimento en el que se envía el servo a 3 posiciones diferentes (posición en datos brutos, ya que el software no ofrece la posibilidad de trabajar en grados), a una determinada velocidad:

Figura 43: Gráfica con los valores del controlador indicados en el manual

- Al incrementar la **ganancia proporcional** debería reducir el tiempo de convergencia del servo a la posición deseada, pero a riesgo de aumentar las oscilaciones si el incremento es muy elevado.
  En la siguiente gráfica se observa como aumentando la ganancia del proporcional de 440 a 17000 se genera una fuerte oscilación y un error en estado estacionario generando ruido en la señal del servo, deteriorando así la señal deseada.

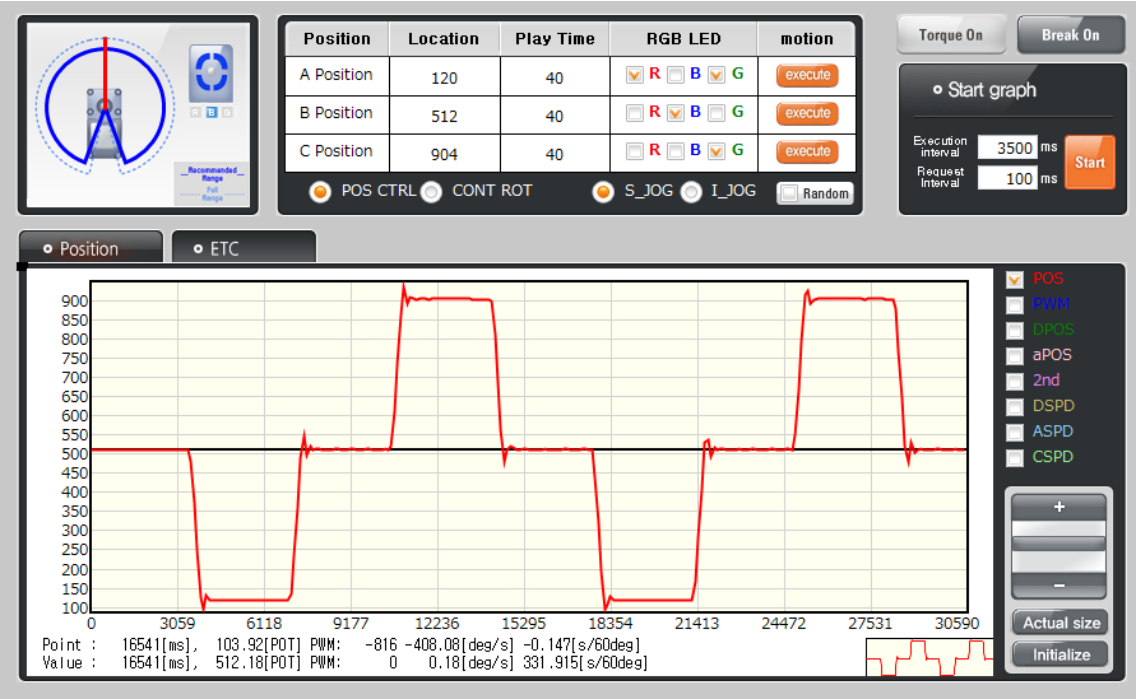


Figura 44: Posición angular en del servo medido en unidades naturales. Aumento del Proporcional.

Y si se le aplica un zoom a la gráfica de la señal se puede visualizar cómo el error permanece en estado estacionario.

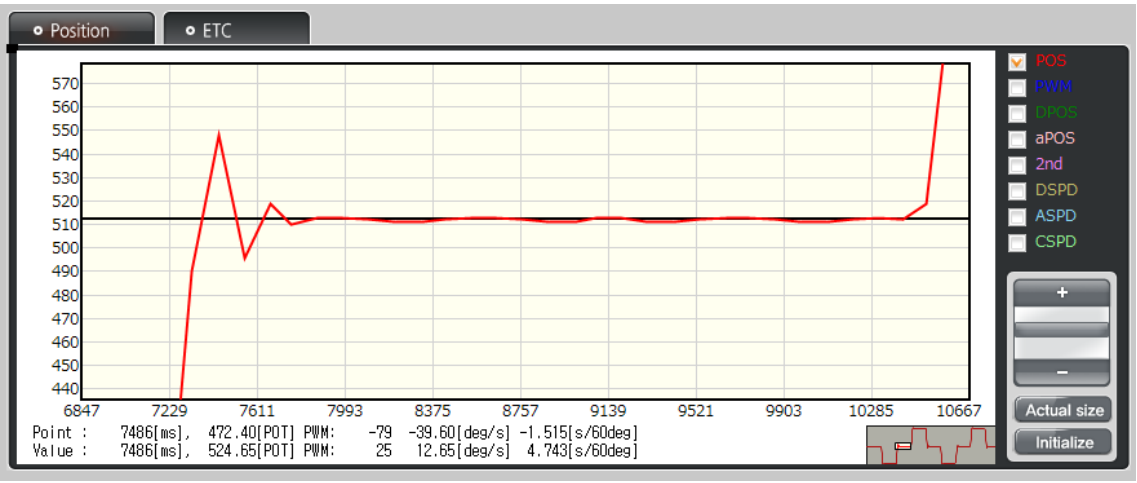


Figura 45: Zoom señal aumento Proporcional.

- La **ganancia integral** permite anular el error en régimen permanente, aunque si se incrementa demasiado puede generar un efecto indeseado de wind-up y desestabilizar el controlador. Al incrementar la ganancia integral de 0 a 2000 se pretende corregir el error en régimen estacionario como que se puede comprobar en la siguiente figura. Sin embargo, la sobreoscilación sigue siendo elevada.

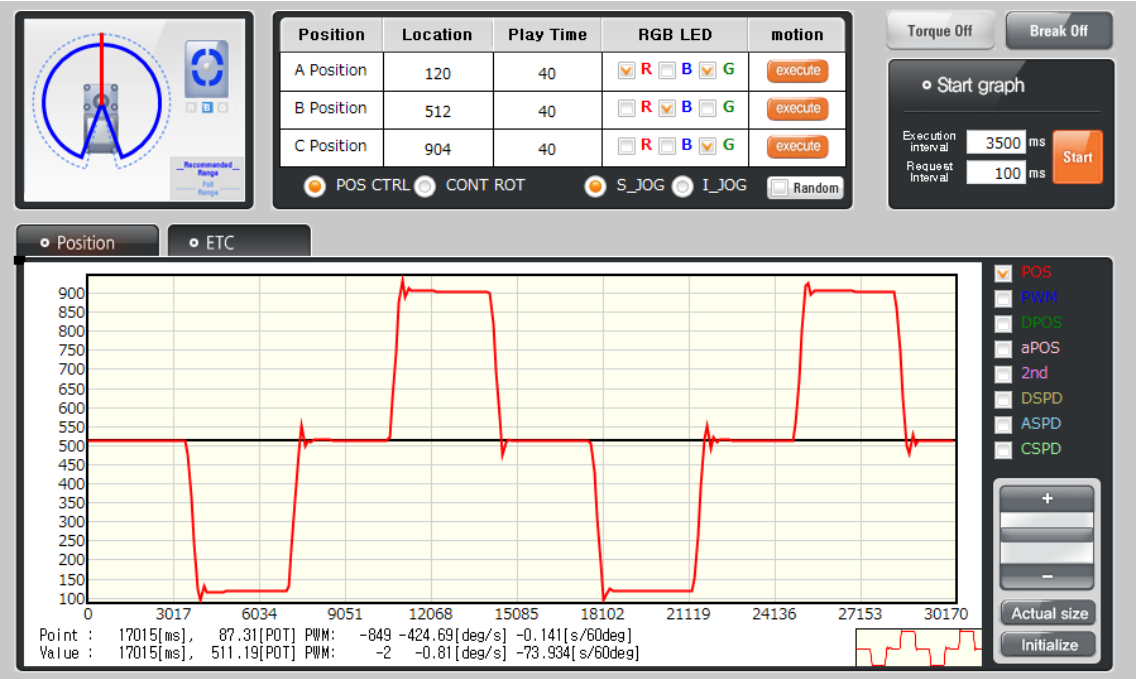


Figura 46: Incremento del integral para corregir error en régimen estacionario.

Aplicándole un zoom a la gráfica se puede observar mejor cómo dicho error en régimen estacionario queda eliminado:

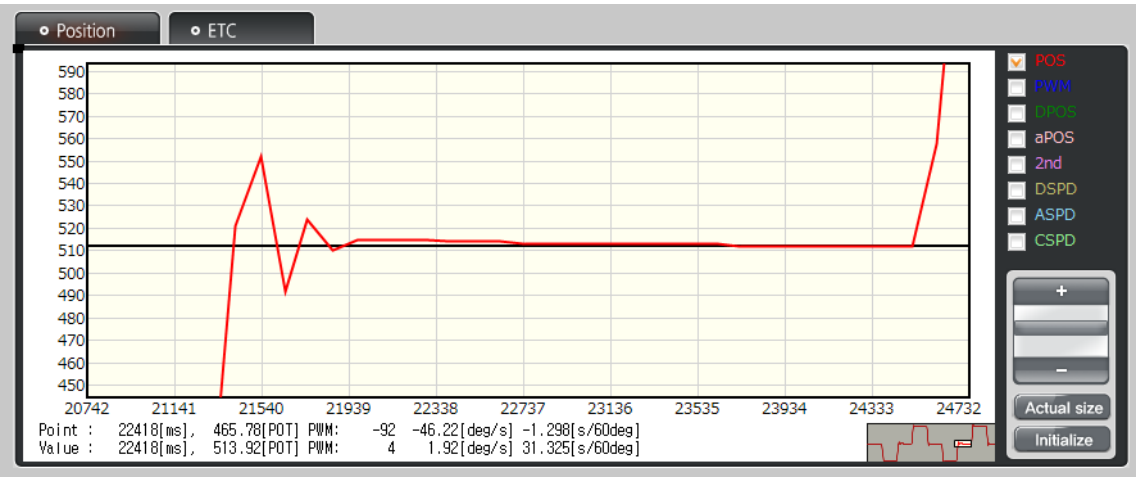


Figura 47: Zoom incremento de la ganancia integral.

- Al incrementar la **ganancia derivativa** se aumenta la convergencia hacia la referencia de posición, reduciendo así la amplitud de la sobreoscilación. Por último, se incrementa el derivativo para compensar las subidas en el proporcional y el integral y reducir así la sobreoscilación generada en esta. En la siguiente figura se observa cómo al aumentar la ganancia derivativa de 8000 a 30000, la sobreoscilación de la señal se reduce, dando lugar a una señal de control más deseable.

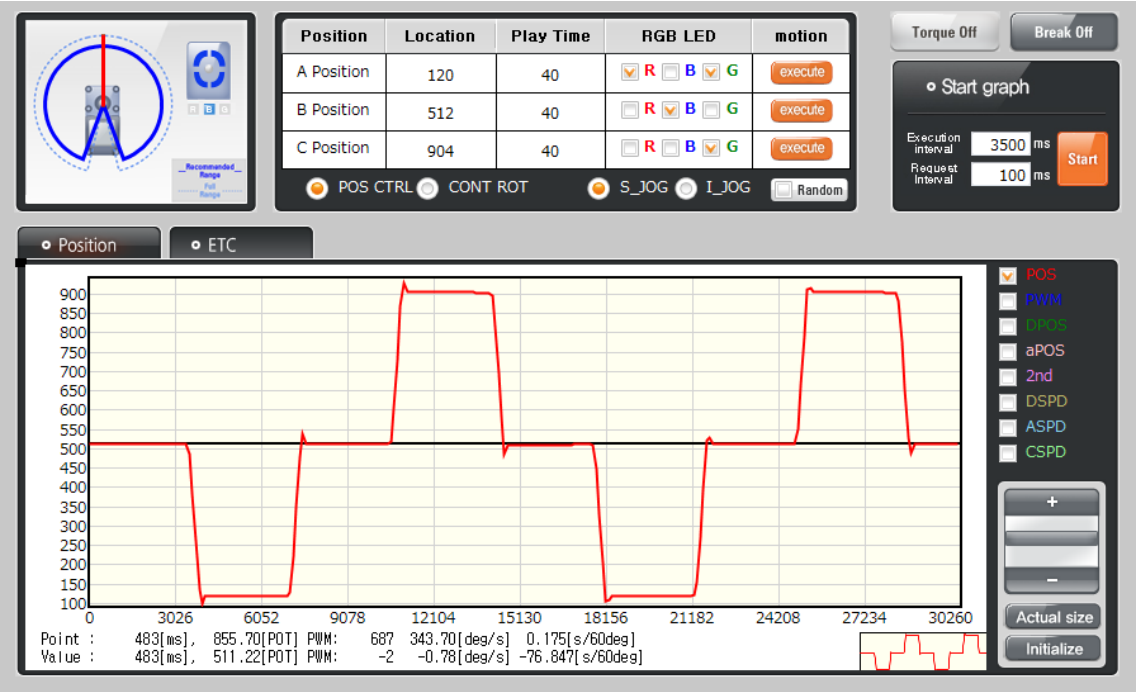


Figura 48: Incremento del término derivativo para compensar las subidas del proporcional e integral.

Haciendo zoom sobre esta gráfica se puede ver que la sobreoscilación ha sido reducida y que el error en régimen permanente es nulo, aunque sigue sin ser un comportamiento deseado como el que se obtiene con los valores por defecto del controlador (figura 44).

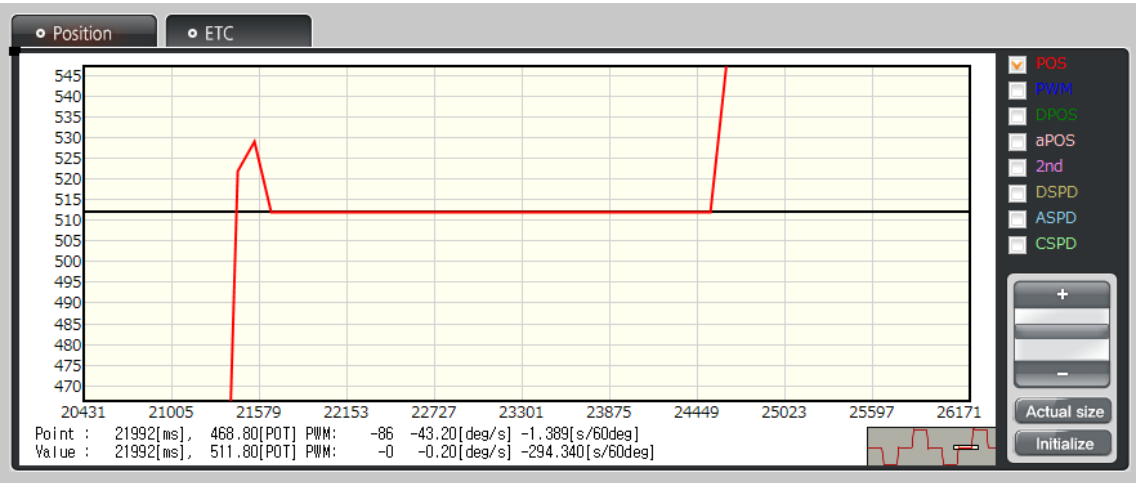


Figura 49: Zoom sobre la posición angular del servo cuando se incrementa el término derivativo.

7. Conclusiones

En este trabajo se ha realizado un análisis de los servos en general y de las principales opciones que se pueden encontrar en el campo de los Smart Servos para su posible aplicación en el área de la robótica aérea, donde se necesita principalmente servos configurables de bajo peso, con un coste no muy elevado y unas prestaciones relativamente buenas. Tras realizar el análisis de mercado, teniendo en cuenta dichas características, se ha determinado que los servomotores Herkulex son una de las opciones más interesantes actualmente para el desarrollo de manipuladores de bajo peso.

Una de las principales funcionalidades que ofrecen estos servos es el protocolo de comunicación Full-Duplex que permite recibir datos del servo a la vez que se está enviando órdenes a este, aumentando así la tasa de control. Otra característica destacada es la relativa los distintos mecanismos de protección que ofrecen basados en monitorizar la temperatura y detectar sobrecargas a partir de la señal PWM, alargando así la vida de estos servomotores.

Otra de las características que hacen a estos servos muy interesantes es la variedad de modelos disponibles: (DRS 0101, 0201, 0402 y 0602), permitiendo al usuario utilizar uno u otro en función de las necesidades de torque y de las restricciones de peso, pudiendo trabajar los distintos modelos de los servos entre sí, aunque se debe tener en cuenta que las versiones 0101/0201 funcionan a una tensión diferente a la de los modelos 0402/0602. Esto puede ser de gran utilidad a la hora de integrar estos servomotores en un brazo robótico, debido a que en cada una de las articulaciones del brazo requerirá un tipo de servo, con un torque diferente.

Dentro de la configuración del servo, los servos Herkulex ofrecen una gran variedad de parámetros que pueden ser controlados, como puede ser el controlador PID, los perfiles de velocidad trapezoidal, el control de velocidad en marcha (VOR)… (En la tabla 6, incluida en el Anexo C se pueden encontrar todos). Todos estos parámetros pueden ayudar a configurar y controlar estos servos de manera que facilite su integración en los sistemas aéreos.

En lo que al desarrollo de los distintos códigos y al uso de distintas plataformas se refiere, se llega a la conclusión que para realizar un análisis de las características que ofrecen estos servos y configurarlos, la mejor opción es utilizar el software que ofrece el fabricante (Herkulex Manager), pero a la hora de manipularlos, trabajar con ellos e integrarlos en un brazo robótico, la mejor opción es generar un código propio en C++, Arduino u otro lenguaje que permita su integración en un sistema de control en tiempo real. A la hora de desarrollar estos códigos para trabajar con el servo es muy importante comprender bien cómo funcionan los envíos de paquetes de datos, ya que

esto será necesario tanto para el control como para la lectura del estado interno del servo.

Los principales problemas que se han encontrado a la hora de realizar el análisis de estos servomotores es la falta de información que se puede encontrar en internet sobre estos, debido a que son relativamente nuevos y no están tan extendidos en el mercado como pueden estar los servos Dynamixel. Otro de los problemas que se han encontrado a la hora de trabajar con estos servos es la necesidad de utilizar los materiales de conexionado que suministra el fabricante Dongbu, así como los problemas de identificación de éste mencionados en la memoria. Por último, otro problema que apareció a la hora de la puesta en marcha fue debido a una errata en uno de los manuales que se pueden encontrar en internet (Robot Shop), ya que indicaba las conexiones a realizar de manera errónea. No obstante dichos problemas han podido superarse y no han impedido hacer un análisis del servo y llegar a las conclusiones que se están exponiendo.

En resumen, ha quedado demostrado que estos servos ofrecen unas prestaciones que pueden resultar muy interesantes en el área de la robótica aérea a un precio asequible, y es por eso que ya se están empezando a integrar en distintos proyectos de robótica y es de suponer que su uso se vaya extendiendo en el mercado.

Para finalizar, remarcar que con el trabajo realizado se han conseguido obtener los objetivos planteados al inicio del trabajo: analizar estos servomotores, realizando la puesta en marcha y desarrollando códigos en distintos lenguajes que permitan su manipulación.

Anexo A: Código en C++ completo

```
#include <iostream>
#include <fstream>
#include <sys/time.h>
#include <unistd.h>
#include <stdlib.h>
#include <termios.h>
#include <fcntl.h>
#include <sys/types.h>
#include <stdint.h>
#include <string.h>

typedef struct
{
        float position;
        float ref;
        float speed;
        float pwm;
        float temperature;
        float voltage;
} SERVO_STATE;

union CharArray2Int16
{
        char MSB_LSB[2];
```

```
        int16_t value;
};

void setTorqueON(uint8_t servoID, int serialInterfaceHandler);
void moveServo(float position, float playTime, uint8_t servoID, int
serialInterfaceHandler);
void readRAMRegister(uint8_t servoID, int serialInterfaceHandler, SERVO_STATE *
servoState);
int configureSerialInterface(int serialInterfaceHandler);
void computeChecksum(char * packet, uint8_t packetSize);
int sendData(char * data, uint8_t dataLength, int serialInterfaceHandler);

int main(int argc, char ** argv)
{
        int serialInterfaceHandler = -1;
        int servoID = -1;
        ofstream outDataFile;
        SERVO_STATE servoState;
        int N = 4;
        float positions[4] = {30, 0, 90, 0};
        float playTimes[4] = {2,2,5,5};
        struct timeval t0;
        struct timeval t1;
        double t = 0;
        int error = 0;

        cout << "Herkulex Program" << endl;
```

```
	cout << "TFG Project" << endl;


	if(argc != 4)
	{
		// Argument0s: (1) Program name, (2) /dev/ttyUSB0, (3) servo ID, (4) file
name
		cout << "ERROR: Número de argumentos invalido" << endl;
		error = 1;
	}
	else
	{
		// Apertura del puerto serie
		serialInterfaceHandler = open(argv[1], O_RDWR | O_NOCTTY |
O_SYNC);
		if(serialInterfaceHandler <= 0)
		{
			error = 1;
			cout << "ERROR: No se pudo abrir el dispositivo USB." << endl;
		}
		else
			error = configureSerialInterface(serialInterfaceHandler);
	}


	if(error == 0)
	{
		// Apertura del fichero
		outDataFile.open(argv[3]);
```

```
if(outDataFile.is_open() == false)
        cout << "ERROR: No se pudo abrir el fichero externo" << endl;

// Habilitar torque
servoID = atoi(argv[2]);
setTorqueON(servoID, serialInterfaceHandler);

// Tiempo del dia
gettimeofday(&t0, NULL);

/*-----Comienza la simulación-----*/
for(int n = 0; n < N; n++)
{
        // Orden de movimiento al servo
        moveServo(positions[n], playTimes[n], servoID,
serialInterfaceHandler);

        // Tiempo actual
        gettimeofday(&t1, NULL);
        t = (t1.tv_sec - t0.tv_sec) + 1e-6*(t1.tv_usec - t0.tv_usec);
        double taux1 = t;

        do
        {
                readRAMRegister(servoID, serialInterfaceHandler,
&servoState);
```

```
                                // Actualiza el tiempo
                                gettimeofday(&t1, NULL);
                                t = (t1.tv_sec - t0.tv_sec) + 1e-6*(t1.tv_usec - t0.tv_usec);
                                usleep(100000);
                                // Escribir en el fichero
                                outDataFile << t << "\t";
                                outDataFile << servoState.position << "\t";
                                outDataFile << servoState.ref << "\t";
                                outDataFile << servoState.pwm << "\t";
                                outDataFile << servoState.speed << "\t";
                                outDataFile << servoState.temperature << "\t";
                                outDataFile << servoState.voltage << "\t";
                                outDataFile << endl;   // Salto de linea

                        } while(t < taux1 + playTimes[n] + 1) ;
                }
        }
        return error;
}

void setTorqueON(uint8_t servoID, int serialInterfaceHandler)
{
        uint8_t packet[16];

        packet[0] = 0xFF;               // Cabecero
        packet[1] = 0xFF;
```

```
        packet[2] = 0x0A;               // Tamaño del paquete
        packet[3] = servoID;   // Servo ID
        packet[4] = 0x03;               // lectura en RAM
        packet[7] = 0x34;               // Registro
        packet[8] = 0x01;               // Bytes a escribir
        packet[9] = 0x60;               // Torque ON

        computeChecksum((char*)packet, packet[2]);

        sendData((char*)packet, packet[2], serialInterfaceHandler);
}

void moveServo(float position, float playTime, uint8_t servoID, int
serialInterfaceHandler)
{
        uint8_t packet[16];
        float playTimeByte = 0;
        float positionAux = 0;
        uint16_t position2Bytes = 0;
        int error = 0;

        // trama
        packet[0] = 0xFF;               // Cabecero
        packet[1] = 0xFF;
        packet[2] = 0x0C;               // Tamaño del paquete
        packet[3] = servoID;   // Servo ID
```

```
        packet[4] = 0x05;               // I_JOG

        // Cálculo y ajuste de goal position
        positionAux = 512 + position / 0.325;
        position2Bytes = (uint16_t)positionAux;
        packet[7] = (uint8_t)(0x00FF & position2Bytes);
        position2Bytes = position2Bytes >> 8;
        packet[8] = (uint8_t)(0x00FF & position2Bytes);

        packet[9] = 0x04;               // Control de posición. LED en verde
        packet[10] = servoID; // Servo ID

        // Cálculo y ajuste del play time en el rango indicado 0-255
        playTimeByte = playTime / 0.0112;
        if(playTimeByte < 1)
                playTimeByte = 1;
        if(playTimeByte > 255)
                playTimeByte = 255;
        packet[11] = (uint8_t)playTimeByte;

        computeChecksum((char*)packet, packet[2]);
        if(error == 0)
                sendData((char*)packet, packet[2], serialInterfaceHandler);
}
```

```
void readRAMRegister(uint8_t servoID, int serialInterfaceHandler, SERVO_STATE *
servoState)

{

        char packet[16];

        char buffer[1024];              // Para almacenar los bytes recibidos de la RAM

        CharArray2Int16 aux;

        float position = 0;

        float voltage = 0;

        float temperature = 0;

        float speed = 0;

        float positionRef = 0;

        float pwm = 0;

        float tick = 0;


        // trama

        packet[0] = 0xFF;               // Cabecero

        packet[1] = 0xFF;

        packet[2] = 0x09;               // Tamaño del paquete

        packet[3] = servoID;   // Servo ID

        packet[4] = 0x04;               // lectura RAM

        packet[7] = 54;                         // Dirección inicial (registro de Voltage en
la RAM)

        packet[8] = 16;                         // Bytes que leer (hasta la Absolute Goal
Position)

        // Comprueba checksum

        computeChecksum((char*)packet, packet[2]);

        // Envio del paquete
```

```
sendData((char*)packet, packet[2], serialInterfaceHandler);

// Lectura de datos en la RAM. Almacena los datos de la memoria RAM en el
buffer. Los datos que interesan estan a partir del buffer 9

bytesReceived = read(serialInterfaceHandler, buffer, 1023);


voltage = 0.074*buffer[9];

temperature = buffer[10];

tick = 0.0112*buffer[12];


position = 0.325*(buffer[15] + 256*buffer[16] - 512);

// Velocidad

aux.MSB_LSB[1] = buffer[18];

aux.MSB_LSB[0] = buffer[17];

speed = 29.09*aux.value;

// PWM

aux.MSB_LSB[1] = buffer[20];

aux.MSB_LSB[0] = buffer[19];

pwm = 9.76562e-4*aux.value;


positionRef = 0.325*(buffer[23] + 256*buffer[24] - 512);


servoState->position = position;

servoState->ref = positionRef;

servoState->speed = speed;

servoState->pwm = pwm;

servoState->temperature = temperature;
```

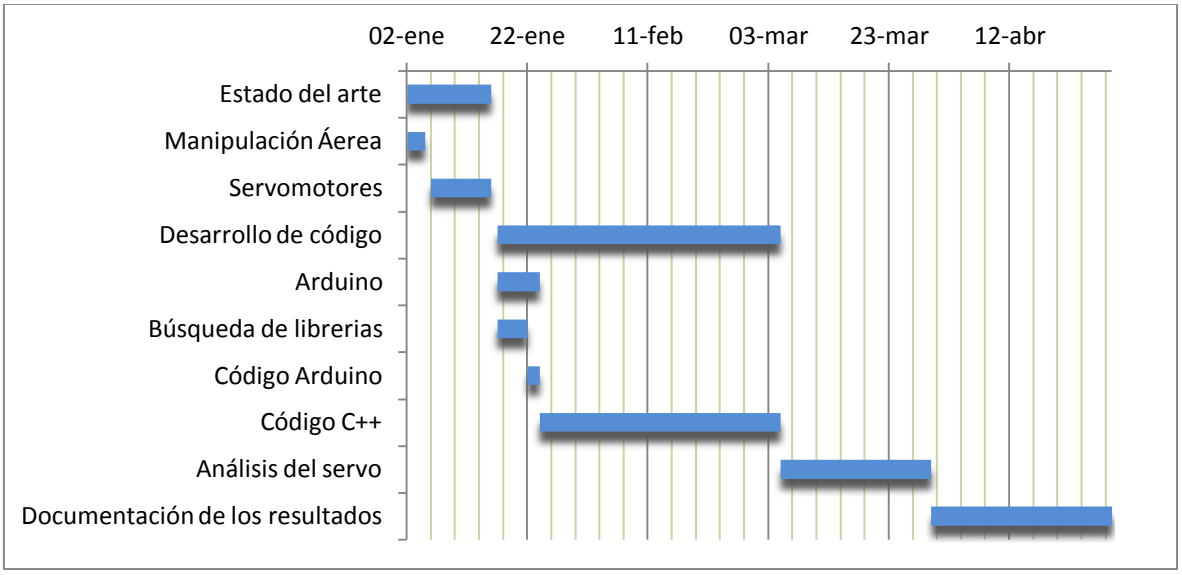
```
        servoState->voltage = voltage;
}


//https://stackoverflow.com/questions/19269847/how-to-read-dev-ttyusb0-device-in-linux
int configureSerialInterface(int serialInterfaceHandler)
{
        struct termios tty;
        int error = 0;


        if(serialInterfaceHandler < 0)
        {
                error = 1;
                cout << "ERROR: [in main::configureSerialInterface] could not open USB device. Try executing with sudo." << endl;
        }
        else
        {
                memset(&tty, 0, sizeof(struct termios));
                if(tcgetattr(serialInterfaceHandler, &tty) != 0)
                {
                        error = 1;
                        cout << "ERROR: [in main::configureSerialInterface] could not get attributes from USB device" << endl;
                }
                else
                {
```

```
// Set baudrate
cfsetospeed (&tty, B115200);
cfsetispeed (&tty, B115200);
// Set other attirbutes
tty.c_cc[VMIN] = 0;             // read doesn't block
tty.c_cc[VTIME] = 5;            // 0.5 seconds read timeout

tty.c_cflag &=  ~PARENB;        // Make 8n1
tty.c_cflag &=  ~CSTOPB;
tty.c_cflag &=  ~CSIZE;
tty.c_cflag |=  CS8;
tty.c_cflag &=  ~CRTSCTS;
tty.c_lflag = 0;
tty.c_oflag = 0;
tty.c_cflag |=  CREAD | CLOCAL;
tty.c_iflag &= ~(IXON | IXOFF | IXANY);
tty.c_cflag |=  CREAD | CLOCAL;
tty.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);    // make raw
tty.c_oflag &= ~OPOST;                  // make raw
// Flush port and apply attributes
tcflush(serialInterfaceHandler, TCIFLUSH);
if(tcsetattr(serialInterfaceHandler, TCSANOW, &tty) != 0)
{
        error = 1;
        cout << "ERROR: [in configureSerialInterface] could not
apply attributes on USB device" << endl;
```

```
                        }
                    }
                }
                return error;
}

void computeChecksum(char * packet, uint8_t packetSize)
{
                uint8_t checksum1 = 0;
                uint8_t checksum2 = 0;
                uint8_t checksumTemp = 0;

                // checksum1
                checksumTemp = packet[2] ^ packet[3] ^ packet[4];
                for(uint8_t n = 7; n < packetSize; n++)
                        checksumTemp ^= packet[n];
                checksum1 = checksumTemp & 0xFE;
                // Asigna el checksum1 a la trama
                packet[5] = checksum1;
                // checksum2
                checksum2 = (~checksumTemp) & 0xFE;
                // Asigna checksum2 a la trama
                packet[6] = checksum2;
}

int sendData(char * data, uint8_t dataLength, int serialInterfaceHandler)
```

```
{
        ssize_t bytesSent = 0;
        int error = 0;

        // Envio de datos
        bytesSent = write(serialInterfaceHandler, data, dataLength);
        if(bytesSent <= 0)
        {
                error = 1;
                cout << "ERROR: no se pudo enviar ningún dato" << endl;
        }
        else if(bytesSent < dataLength)
        {
                error = 1;
                cout << "ERROR: no se pudo enviar todos los datos" << endl;
        }
        Return error;
}
```

Anexo B: Planificación Temporal



Anexo C: Características servo motores Herkulex

Modelo Servo	DRS-0101	DRS-0201	DRS-0402	DRS-0602
Tamaño	24x45x32mm	24x45x32mm	35x56x38mm	35x56x38mm
Peso	45g	60g	123g	145g
Radio de reducción	1/266	1/266	1/202	1/202
Material de engranaje	Plástico	Metal pesado reforzado	Metal reforzado	Metal reforzado
Voltage de entrada	7.4V(7 a 12)	7.4V(7 a 12)	12V(9.5 a 14.8)	12V(9.5 a 14.8)
Corriente en Standby	33mA	33mA	33mA	33mA
Motor	Motor de metal DC con núcleo	Motor de metal DC sin núcleo	Motor PEMIUM FALHABER DC sin núcleo con un codificador magnético integrado	Motor PEMIUM FALHABER DC sin núcleo con un codificador magnético integrado
Stall Torque Máximo	12 kg.cm 1.2 N.m	24 kg.cm 2.4 N.m	52 kg.cm 5.1 N.m	77.5 kg.cm 7.6 N.m
Máxima Velocidad	58 RPM a 12 V	63 RPM a 12V	59 RPM a 14.8V	59 RPM a 14.8V
Resolución	0.325º	0.325º	0.028º	0.028º
Ángulos de operación	320º o rotación continua (en control de velocidad)	320º o rotación continua (en control de velocidad)	900º o rotación continua (en control de velocidad)	900º o rotación continua (en control de velocidad)
Temperatura	0-85º	0-85º	0-80º	0-80º
Protocolo	Full Duplex serial asíncrono (Nivel TTL)	Full Duplex serial asíncrono (Nivel TTL)	Full Duplex serial asíncrono (Nivel TTL)	Full Duplex serial asíncrono (Nivel TTL)
ID	0-253, 254 solo para envío global	0-253, 254 solo para envío global	0-253, 254 solo para envío global	0-253, 254 solo para envío global
Baud Rate	0.67Mbps	0.67Mbps	1Mbps	1Mbps
Feedback	4 posiciones, Velocidad, Temperatura, carga, voltaje, etc.	4 posiciones, Velocidad, Temperatura, carga, voltaje, etc.	5 posiciones, Velocidad, Temperatura, carga, voltaje, etc.	5 posiciones, Velocidad, Temperatura, carga, voltaje, etc.
Algoritmos de control	PID, Perfiles de velocidad trapezoidal, Control de velocidad en marcha, Control	PID, Perfiles de velocidad trapezoidal, Control de velocidad en marcha, Control	PID, Perfiles de velocidad trapezoidal, Control de velocidad en marcha, Control del	PID, Perfiles de velocidad trapezoidal, Control de velocidad en marcha, Control

	del offset y saturación en el torque, protección contra sobrecarga, Calibración, Zonas muertas y 54 parámetros ajustables.	del offset y saturación en el torque, protección contra sobrecarga, Calibración, Zonas muertas y 54 parámetros ajustables.	offset y saturación en el torque, protección contra sobrecarga, Calibración, Zonas muertas y 54 parámetros ajustables.	del offset y saturación en el torque, protección contra sobrecarga, Calibración, Zonas muertas y 54 parámetros ajustables.

Tabla 6: Características servomotores Herkulex

PLANOS

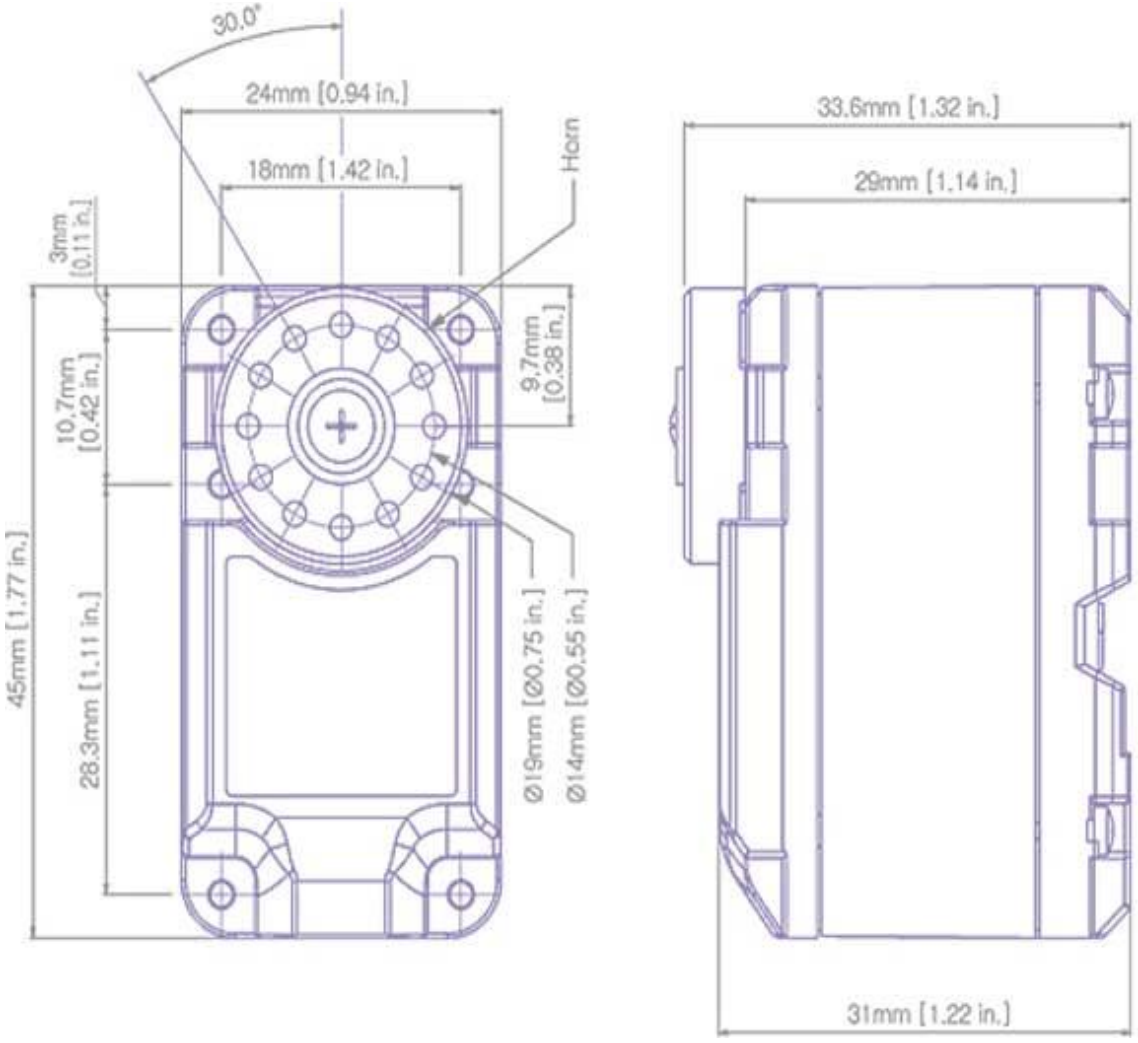


Figura 50: Dimensiones servo Herkulex DRS-0101

PRESUPUESTO

Material Utilizado	Precio	
Herkulex DRS-0101	35€	
Herkulex Manager Kit	36€	
Cable de conexionado para servos Herkulex	6€	
Cargador regulable de 3/12V	9.90€	
Arduino Mega 2560	40€	
Protoboard	4.95€	
	TOTAL	**131.85€**

Tabla 7: Presupuesto material utilizado

BIBLIOGRAFÍA

[1] Christopher Bolkcom: *Homeland Security: Unmanned Aerial Vehicles and Border Surveillance.* www.dtic.mil/get-tr-doc/pdf?AD=ADA477712

[2] ShinIchiro Higashino, Minoru Funaki: *Development and Flights of Ant-Plane UAVs for aerial filiming and Geomagnetic Survey in Antartida.* http://ojs.unsysdigital.com/index.php/just/article/view/19

[3] Francisca López-Granados: *Uso de Vehículos Aéreos No tripulados (UAV) para la evaluación de la producción agraria.* http://www.revistaambienta.es/WebAmbienta/marm/Dinamicas/secciones/articulos/UAV.htm

[4] G. Heredia, A. Ollero, J.A. Acosta, D. Llorente, V. Vega, J. Braga, I. Sanchez, A.E. Jimenez-Cano: *Control of a multirotor outdoor aerial manipulator.* http://ieeexplore.ieee.org/document/6943038/authors

[5] Paul I.E. Pounds, Daniel R. Bersak, Aaron M. Dollar: *The Yale Aerial Manipulator: Grasping in flight.* http://ieeexplore.ieee.org/abstract/document/5980477/authors

[6] George P. Kontoudis, Minas V. Liarokapis, Agisilaos G. Zisimatos, Christoforos I. Mavrogiannis, Kostas J. Kyriakopoulos: *Open-source, anthropomorphic, underactuated robot hands with a selectively lockable differential mechanism: Towards affordable prostheses.* http://ieeexplore.ieee.org/document/7354209/

[7] Michael Sfakiotakis, John Fasoulas, and Roza Gliva: *Dynamic modeling and experimental analysis of a two-ray undulatory fin robot.* http://ieeexplore.ieee.org/document/7353395/

[8] Wei Gao, Ke Huo, Jasjeet S. Seehra, Karthik Ramani, and Raymond J. Cipra: *HexaMorph: A reconfigurable and foldable hexapod robot inspired by origami.* http://ieeexplore.ieee.org/document/6943214/

[9] Jonahatan Tapia, Eric Wineman, Patrick Benavidez, Aldo Jaimes, Ethan Cobb, John Parsi, Dan Clifton, Mo Jamshidi and Benjamin Champion: *Autonomous mobile robot platform with multi-variant task-specific end-effector and voice activation.* http://ieeexplore.ieee.org/document/7583014/

[10] Rong-Jyue Wang, Jun-Wei Zhang, Jia-Ming Xu, Hsin-Yu Liu: *The multiple-function intelligent robotic arms.* http://ieeexplore.ieee.org/abstract/document/5277256/

[11] David Rivas, Marcelo Alvarez, Patricio Velasco, Javier Mamarandi, Jose Luis Carrillo-Medina, Víctor Bautista, Omar Galarza, Patricio Reyes, Mayra Erazo, Miltón Pérez and Mónica Huerta: *BRACON: Control system for a robotic arm with 6 degrees of freedom for education systems.* http://ieeexplore.ieee.org/abstract/document/7081174/

[12] John-David Warren, Josh Adams, Harald Molle: *Arduino for Robotics.* https://link.springer.com/chapter/10.1007/978-1-4302-3184-4_2

[13] A. Valera, A.Soriano y M. Valles: *Plataformas de Bajo Coste para la Realización de Trabajos Prácticos de Mecatrónica y Robótica.* http://www.sciencedirect.com/science/article/pii/S1697791214000557

[14] Entorno de desarrollo Arduino. https://www.arduino.cc/en/Main/Software

[15] Herkulex manager Software. http://hovis.co.kr/guide/herkulex_manager_eng.html

[16] Manual Herkulex servo DRS 0101/0201 http://www.robotshop.com/media/files/PDF/manual-drs-0201.pdf

[17] Hongfhu Zhou: *DC Servo Motor PID Control in Mobile Robots with Embedded DSP.* http://ieeexplore.ieee.org/abstract/document/4659500/