UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MÁSTER EN INGENIERÍA INDUSTRIAL

ESTUDIO Y DISEÑO DEL CONTROL DE MOVIMIENTOS DE UN ROBOT ÁPODO

AUTORA: CRISTINA ASENJO MADRIGAL

TUTOR: LEOPOLDO ARMESTO ÁNGEL

Curso Académico: 2020-21

A mi tutor, Leopoldo Armesto, por darme esta oportunidad
y guiarme durante el desarrollo de este trabajo;

a los profesores del Máster,
por compartir su conocimiento
y por descubrirme la pasión por esta área de la ingeniería;

a mis compañeros de clase,
por acompañarme durante estos años
y hacer mejor esta etapa tan importante de mi vida;

y a mi madre, mi hermana y a Victorio,
por creer siempre en mí
y por todo el apoyo y ayuda que me han ofrecido.

Resumen

La robótica móvil es un campo muy extenso que consiste principalmente en el desplazamiento de un robot de un punto hasta otro. Una forma de desplazamiento corresponde con los sistemas de locomoción ápodos, basados en los movimientos de serpientes, orugas o gusanos. Este tipo de locomoción presenta un difícil control debido a que se basa en la deformación de su propia estructura, y por tanto tiene un gran número de parámetros a controlar. Sin embargo, presenta un gran número de ventajas como su flexibilidad y adaptabilidad.

El objetivo del presente Trabajo de Final de Master consiste en el estudio, implementación y control de este tipo de locomoción ápoda.

Dicho trabajo abarca la investigación de los diferentes métodos de locomoción utilizados por distintas familias de animales ápodos, así como su posterior aproximación a un modelo matemático y configuración, y las consideraciones de diseño para realizar un prototipo y su arquitectura. Además, se realizará el diseño del sistema de control, y se implementará tanto en un prototipo real como en un entorno de simulación.

El proceso finalizará con la optimización de los parámetros del control del movimiento mediante diversos algoritmos.

Palabras Clave

Robot ápodo, robot móvil, serpiente, locomoción, control, serpentina, *snake-robots,* movimiento, senoidal, red neuronal

Resum

La robòtica mòbil és un camp molt extens que consisteix principalment en el desplaçament d'un robot d'un punt fins un altre. Una forma de desplaçament correspon amb els sistemes de locomoció àpods, basats en els moviments de serps, erugues o cucs. Aquest tipus de locomoció presenta un difícil control pel fet que es basa en la deformació de la seua pròpia estructura, i per tant té un gran nombre de paràmetres a controlar. No obstant això, presenta un gran nombre d'avantatges com la seua flexibilitat i adaptabilitat.

L'objectiu del present Treball de Final de Màster consisteix en l'estudi, implementació i control d'aquesta mena de locomoció àpoda.

Aquest treball abasta la investigació dels diferents mètodes de locomoció utilitzats per diferents famílies d'animals àpods, així com la seua posterior aproximació a un model matemàtic i configuració, i les consideracions de disseny per a realitzar un prototip i la seua arquitectura. A més, es realitzarà el disseny del sistema de control, i s'implementarà tant en un prototip real com en un entorn de simulació.

El procés finalitzarà amb l'optimització dels paràmetres del control del moviment mitjançant diversos algorismes.

Paraules Clau

Robot àpode, robot mòbil, serp, locomoció, control, serpentina, snake-robots, moviment, senoidal, xarxa neuronal

Abstract

Mobile robotics is a very broad field that mainly consists of the movement of a robot from one point to another. One form of movement corresponds to apex locomotion systems, based on the movements of snakes, caterpillars or worms. This type of locomotion is difficult to control because it is based on the deformation of its own structure, and therefore has a large number of parameters to control. However, it has a number of advantages such as flexibility and adaptability.

The aim of this Master's Thesis is the study, implementation and control of this type of apex locomotion.

This work includes the investigation of the different methods of locomotion used by different families of apex animals, as well as their subsequent approximation to a mathematical model and configuration, and the design considerations to create a prototype and its architecture. In addition, the design of the control system will be carried out and implemented both in a real prototype and in a simulation environment.

The process will end with the optimisation of the motion control parameters using various algorithms.

Key Words

Apod robot, mobile robot, snake, locomotion, control, serpentine, snake-robots, motion, sine wave, neural network

Índice General del Documento

MEMORIA DESCRIPTIVA

Índice de la Memoria Descriptiva

Índice de Figuras

Índice de Tablas

Índice de Ecuaciones

Nomenclatura

Variable	Descripción
A_{circ}	Arco de la circunferencia realizada por el prototipo para el desplazamiento.
a	Parámetro general relativo a la amplitud de la onda serpentina
a_h	Parámetro relativo a la amplitud de la onda horizontal serpentina
a_v	Parámetro relativo a la amplitud de la onda vertical serpentina
b	Parámetro general relativo al número de ondulaciones por unidad de longitud de la onda serpentina
b_h	Parámetro relativo al número de ondulaciones por unidad de longitud de la onda horizontal serpentina
b_v	Parámetro relativo al número de ondulaciones por unidad de longitud de la onda vertical serpentina
c	Parámetro general relativo a la dirección de la curva de la onda serpentina
c_h	Parámetro relativo a la dirección de la curva de la onda horizontal serpentina
c_v	Parámetro relativo a la dirección de la curva de la onda vertical serpentina
D_{circ}	Cuerda de la circunferencia realizada por el prototipo para el desplazamiento.
e	Vector de errores de la red neuronal
$epoch$	Número de iteración/época de la red neuronal
g	Gradiente
H	Aproximación a la matriz Hessiana
i	Número de módulo que se está haciendo referencia
J	Matriz Jacobiana que contiene las primeras derivadas de los errores de la red neuronal respecto a los pesos y desviaciones
K	Número de oscilaciones de la curva serpentinoide

Variable	Descripción
L	Longitud total de la curva serpentinoide
LM_{epoch}	Actualización para la iteración $epoch$ del algoritmo Levenberg-Marquardt
n	Número de módulos totales
R	Resultado del análisis de regresión
r	Radio de la circunferencia realizada por el prototipo para el desplazamiento
s	Variable continua que representa la distancia a lo largo del eje x de la curva serpentinoide
t	tiempo
v	Velocidad lineal
w	Velocidad angular
X	Proyección cartesiana de la curva serpentoidal sobre el eje X
x_i	Proyección cartesiana del módulo i sobre el eje X.
x_f	Proyección cartesiana de la posición final del prototipo tras la simulación sobre el eje X
Y	Proyección cartesiana de la curva serpentoidal sobre el eje Y
y_i	Proyección cartesiana del módulo i sobre el eje Y
y_f	Proyección cartesiana de la posición final del prototipo tras la simulación sobre el eje Y
α_0	Angulo inicial de la curva serpentinoide
β_i	Ángulo que representa la posición del módulo i respecto el eje X
δ	Ángulo relativo entre dos articulaciones consecutivas
θ	Angulo de doblaje de la curva serpentinoide
μ	Parámetro que cuantifica la precisión y rapidez del modelo de red neuronal.
σ	Ángulo girado por el prototipo para el desplazamiento

CAPÍTULO 1. INTRODUCCIÓN

1.1. CONTEXTO HISTÓRICO

La robótica es una ciencia que une diversas disciplinas de la tecnología, como la electrónica, mecánica e informática. Gracias al avance de la tecnología y al reciente auge de nuevas técnicas como la inteligencia artificial, se están consiguiendo lograr grandes hitos en el área de la robótica, que hasta hace nada se consideraban lejanas y futuristas.

A lo largo de la historia, el ser humano ha construido máquinas con el fin de facilitar todo tipo de tareas. Los antiguos egipcios ya unieron brazos mecánicos a las estatuas de sus dioses, y los griegos construyeron estatuas con sistemas hidráulicos.

Las primeras máquinas mecánicas que se pueden empezar a considerar robots surgen a principios del siglo XIX, cuando Jaquard inventa en 1801 un telar mecánico que utiliza tarjetas perforadas para seguir un patrón de costura. Al mismo tiempo, en 1805, Maillardet construye un muñeco mecánico que, por medio de un sistema de levas, realiza dibujos [1].

Paralelamente a los inventos anteriores, durante los siglos XVIII y XIX se comienzan a desarrollar sistemas de automatización en las fábricas. Se construyen sistemas que imitan las tareas de los obreros, como verter caucho líquido en moldes neumáticos o poner tapones en botellas. Sin embargo, estos mecanismos son muy sencillos y no permiten sustituir la mano de obra humana en trabajos más complejos.

En 1945, el mismo Leonardo Da Vinci contribuye en la revolución de la robótica con el diseño de un autómata humanoide. Años después, se construiría el diseño conceptual realizado por Leonardo, demostrándose su capacidad para realizar varios movimientos semejantes a los de los humanos como sentarse, mover los brazos, o girar el cuello [2].

Sin embargo, si se habla de la evolución de la robótica, uno de los padres de esta ciencia es George Devol, quien en 1948 construye el primer brazo robótico para el ensamblaje de vehículos en líneas de producción industrial.

Pocos años después, en 1978, Devol realizaría el primer robot programable de la historia, conocido como "PUMA" (*Programmable Universal Machine for Assembly*), capaz de mover un objeto y colocarlo en cualquier orientación. Este prototipo seria la base de los robots industriales que se conocen en la actualidad (véase la Figura 1.1).

Figura 1.1: Brazo industrial "PUMA" de George Devol (*Fuente: researchgate.net*)

Además, aparte de realizar grandes avances a nivel técnico en la evolución robótica, Devol también impulsa esta nueva tecnología creando, junto a Joseph F. Engelberger, la primera empresa de robótica de la historia. Más tarde, Devol desarrollaría otros negocios de consultoría científica con el fin de realizar e investigar sensores visuales y táctiles para robots [3].

De manera paralela al crecimiento de la robótica industrial, surge otra línea de desarrollo consistente en mecanismos con la habilidad de desplazarse. El primer dispositivo capaz de moverse por sí mismo se construye en 1953. Este mecanismo es considerado el primer robot móvil autónomo de la historia, y era capaz de seguir una fuente de luz utilizando un sistema mecánico realimentado, sin necesidad de ordenes externas.

Además, durante este tiempo se realizan grandes avances a nivel tecnológico y computacional. En 1936 Alan Turing sienta las bases de la informática y el concepto de algoritmo, y en 1941 se crea la primera computadora programable, considerado como el primer ordenador de la historia moderna.

Llegados a este punto de la historia, el desarrollo de la robótica, así como la evolución de tecnología existente, permite orientar esta nueva rama de la ciencia hacia la posibilidad de dotar a los robots de cierta inteligencia por medio de sensores.

En 1968 surge en un instituto estadounidense el robot *"Shakey"*, un robot capaz de desplazarse por su cuenta, gracias a la combinación de múltiples sensores táctiles junto con una cámara de visión para poder interactuar con el entorno. Con ello, *Shakey* sería capaz de recorrer de forma autónoma un espacio ocupado por obstáculos, convirtiéndose en el primer robot móvil inteligente del mundo [4].

Sin embargo, no sería hasta casi el siglo XXI, cuando al fin cobra un gran auge la inteligencia artificial. Esta tecnología es capaz de dotar de inteligencia a las máquinas y robots. La eficacia de esta ciencia queda comprobada en 1996, cuando una supercomputadora creada por IBM vence en una partida de ajedrez al campeón del mundo. [5].

A partir de este momento, la tecnología robótica ligada a la inteligencia artificial comienza a avanzar a pasos de gigante. En 2010, la empresa multinacional *Apple* crea la aplicación "*Siri*", un asistente virtual capaz de entender, hacer recomendaciones y ejecutar tareas por medio de procesamiento del lenguaje natural [6].

En 2012 se desarrolla un superoperador capaz de aprender, por medio tecnología *Deep Learning*[1], a identificar caras humanas [8].

Además, surgen grandes líneas de investigación abiertas, como el desarrollo de un coche autónomo capaz de percibir el medio que le rodea e imitar las capacidades humanas de conducción. Asimismo, la empresa estadounidense "*Boston Dynamics*" -gran referente en investigación y construcción de robots móviles- desarrolla un robot móvil llamado *"Spot",* que tal y como se observa en la Figura 1.2, imita el aspecto de un perro.

Figura 1.2: Robot cuadrúpedo "*Spot*" desarrollado por Boston Dynamics (*Fuente: bostondynamics.com*)

Spot es el robot cuadrúpedo más avanzado del mundo en la actualidad y es capaz de inspeccionar y capturar datos. Además, tiene un sistema de movimiento autónomo capaz de desplazarse por terrenos accidentados, bajar y subir escaleras, evitar obstáculos y recuperar el equilibrio [9].

[1] Se refiere al conjunto de algoritmos de aprendizaje automático -del inglés, *machine learning*- inspirados en la estructura y el funcionamiento del cerebro, llamados redes neuronales artificiales. [7].

En la actualidad, existen un sinfín de aplicaciones robóticas inteligentes utilizadas en el día a día. Desde robots aspiradores para el hogar, hasta máquinas como "*Da Vinci*", capaz de intervenir en operaciones quirúrgicas sumamente complejas [10].

Sin embargo, durante la última década ha surgido una nueva línea de investigación robótica con el propósito de desarrollar robots capaces de desplazarse por todo tipo de medios para la inspección y rescate en terrenos desconocidos o irregulares.

Esta vía se desmarca del término clásico que se conoce de robots, debido a que no puede basarse en un sistema de ruedas o patas como los construidos hasta el momento. Así es como surge el concepto de **robot ápodo**, un tipo de robot cuyo método de locomoción está bio-inspirado en animales carentes de extremidades.

A lo largo del CAPÍTULO 2, puede observarse con todo detalle un análisis de esta tipología de robots. No obstante, este campo se encuentra todavía en un enorme crecimiento, y su desarrollo vendrá marcado por las investigaciones de la próxima década.

1.2. OBJETO DE PROYECTO Y OBJETIVOS

El Objeto del presente Trabajo Final de Máster no es otro que el estudio e implementación de un robot ápodo capaz de desplazarse de forma autónoma y en dos dimensiones, independientemente del terreno por el que se realice el movimiento. Además, se pretende realizar su control mediante una red neuronal y su validación en un prototipo real.

Esta línea de investigación es muy amplia, por lo que es imprescindible realizar elecciones de los aspectos en los que se va a centrar el presente Trabajo de Final Máster. Los principales objetivos que se desarrolla en el documento son:

- Estudiar la viabilidad de la locomoción de los robots ápodos, así como su aplicabilidad e impacto en la sociedad.
- Diseñar un sistema de locomoción serpentino por medio de trayectorias sinusoidales que permita un movimiento en dos direcciones de avance.
- Seleccionar la configuración óptima para el caso de estudio concreto que se lleva a cabo.
- Programar una red neuronal para controlar los distintos parámetros de las ondas sinusoidales en función de una velocidad lineal y angular dada.
- Diseñar y ensamblar un prototipo a escala para la validación de la locomoción seleccionada, así como el correcto funcionamiento de la red neuronal.

Asimismo, otros objetivos secundarios que pretenden alcanzarse a lo largo del desarrollo del presente trabajo son:

- Desarrollar un software de simulación para la evaluación de las soluciones propuestas, así como su configuración con comportamientos similares a la realidad.
- Elaborar un modelo matemático para la generación de trayectorias sinusoidales que imiten el movimiento serpentino.
- Estudiar la relación de dependencia existente entre los distintos parámetros de avance y giro del robot diseñado.
- Analizar los distintos parámetros de entrenamiento de la red neuronal que optimicen las predicciones realizadas por la red.
- Diseñar una interfaz gráfica de usuario (GUI) para facilitar el control del prototipo a nivel usuario.
- Analizar y seleccionar los diferentes componentes mecánicos y electrónicos para la construcción del prototipo.

1.3. METODOLOGÍA Y HERRAMIENTAS

El presente apartado pretende describir la metodología de trabajo seguida, así como las herramientas empleadas en cada uno de los pasos para el desarrollo del robot ápodo objeto de estudio. El hilo lógico que se sigue a lo largo del trabajo, puede observarse de forma esquematizada en el diagrama de flujo propuesto en la Figura 1.3.

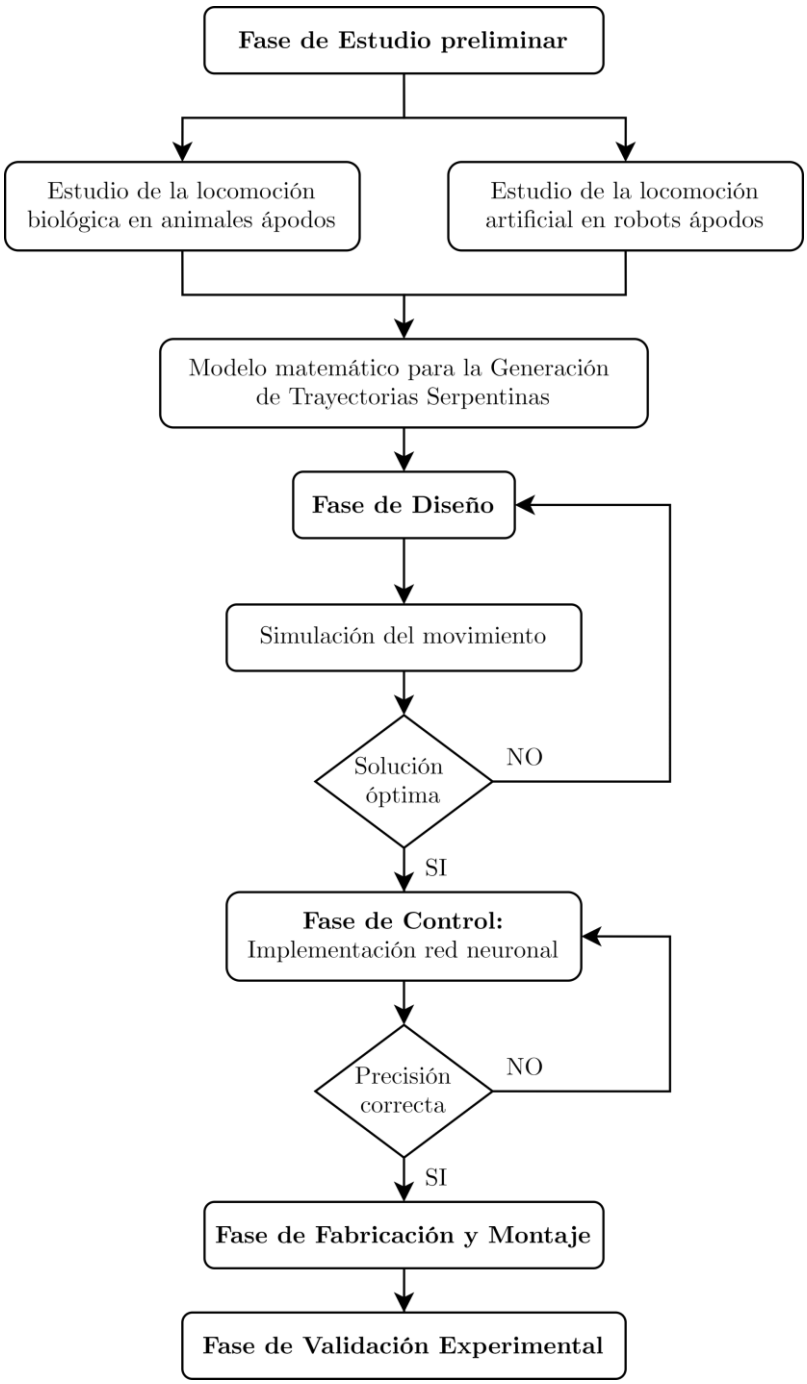


Figura 1.3: Diagrama de flujo de la metodología de trabajo

En primer lugar, se realiza un estudio de los diferentes métodos de locomoción ápoda que se dan en la naturaleza. Se estudia la viabilidad de llevar a cabo dicho estudio y su aplicabilidad en la sociedad actual.

Tras dicho estudio, se analiza y replica el modelo matemático para la generación de trayectorias serpentinas artificiales por medio de ondas sinusoidales.

Una vez realizada la documentación previa, se lleva a cabo la fase de diseño. Para ello se hace uso de un software específico para el diseño CAD como es *Solidworks*.

Posteriormente, se realiza la simulación del movimiento para el diseño mecánico realizado. Este proceso se repite hasta encontrar la configuración mecánica óptima que mejore el movimiento serpentino.

Para las simulaciones se utiliza el software *CoppeliaSim* (versión 4.2.0), también conocido como V-REP. Se trata de un simulador robótico con un entorno de desarrollo integrado. Su arquitectura se basa en un control distribuido, es decir, cada objeto puede ser controlado individualmente a través de un *script* embebido, un *plugin*, nodo *BlueZero*, nodo *ROS*, un cliente *API* remoto, etc. Además, cuenta con la ventaja de que los controladores pueden programarse en múltiples lenguajes como *C/C++*, *Python*, *Java*, *Lua*, *Matlab* u *Octave*, por lo que se trata de un software muy útil y versátil [11].

Una vez alcanzada la configuración óptima, se realiza la fase de control. Para ello se ha realizado una red neuronal por medio de *Matlab* `R2019B`, un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio. Además, para la programación de redes neuronales se utiliza la *Toolbox "Deep Learning Toolbox",* que incorpora todas las funciones necesarias para la implementación de una red neuronal [12].

Después de obtener la red neuronal que controla de forma óptima el movimiento, se realiza el montaje de los componentes del prototipo.

Por último, se realiza la fase de validación de todo el estudio y control realizado sobre el prototipo real, analizándose los resultados obtenidos. La implementación del movimiento en el prototipo real se realiza mediante el entorno de desarrollo integrado *Arduino IDE*. Esta herramienta cuenta con la ventaja de que es una plataforma de código abierto, flexible y fácil de utilizar. En concreto se ha utilizado la versión 1.8.15 [13].

1.4. JUSTIFICACIÓN DEL PROYECTO

Uno de los grandes problemas de la robótica móvil consiste en la estabilización del cuerpo a controlar. Hasta el momento, la mayoría de los robots se basan en un sistema de locomoción por patas o ruedas. A causa de ello, la estabilidad de dichos autómatas se ve comprometida en determinados terrenos y medios.

Es por ello por lo que durante las últimas décadas, ha surgido una nueva línea de investigación que trata de solventar el problema de la estabilidad. La solución más ampliamente utilizada debido a su eficiencia, es precisamente la utilización de robots ápodos. Como se aborda en capítulos posteriores, debido a la carencia de extremidades y ruedas, esta configuración utiliza el propio cuerpo para llevar a cabo el movimiento de avance, aumentando considerablemente la estabilidad y la capacidad de cumplir su propósito en cualquier entorno.

Es por ello que, como se explica con mayor detalle posteriormente, estos robots son potencialmente útiles en aplicaciones de exploración espacial y operaciones de búsqueda y rescate. Además, un aspecto fundamental en estos casos es la sustitución del ser humano por estos robots, para la realización de tareas de riesgo extremo como las mencionadas.

Y con la mirada en el futuro, en el momento en el que se consiga implementar e integrar esta tecnología de forma óptima, puede suponer un antes y un después en campos tan fundamentales como la medicina. De esta forma, operaciones costosas o inviables en la actualidad, podrían conseguirse con relativa facilidad.

Y es precisamente por el gran potencial que presenta esta tecnología para la sociedad, una de las razones principales por la que se decide desarrollar un estudio de este método de locomoción ápoda en el presente Trabajo Final de Máster. Además, más allá de las aplicaciones reales, supone un avance y una innovación en lo que se conoce hoy en día como robot, desmarcándose de la línea clásica de robots con patas o ruedas que ha predominado desde los inicios de la robótica móvil.

CAPÍTULO 2.
MARCO TEÓRICO

2.1. ROBÓTICA MÓVIL

A lo largo de la historia, la evolución de la robótica se ha visto claramente marcada por cinco generaciones de robots

La primera generación surge de la mano de los conocidos manipuladores: sistemas mecánicos con un sencillo sistema de control. La segunda generación incorpora la capacidad de memorizar y repetir procesos, mientras que la tercera generación añade por primera vez sensores para la adquisición de información del entorno.

La cuarta generación viene marcada por la incorporación de sistemas de visión artificial, otorgando al robot la capacidad de captar de forma visual el entorno, así como de detectar objetos en él.

Por último, la quinta generación -actualmente en desarrollo- es en la que interviene la inteligencia artificial. Los robots de esta generación también se denominan robots inteligentes, ya que son capaces de aprender y actuar de forma independiente ante situaciones no conocidas [14].

El paso a través de estas cinco generaciones a lo largo del tiempo, así como la unión de las diversas disciplinas y tecnologías que se han ido desarrollando paralelamente, es lo que ha dado lugar a lo que hoy en día se conoce como robot. Según la universidad de Oxford (2021) es *"toda máquina automática programable capaz de realizar tareas de forma autónoma y sustituir a los seres humanos en algunas tareas"* [15].

Por otro lado, en la robótica existen dos grandes áreas de investigación: la manipulación y la locomoción. La primera de ellas consiste en la capacidad de actuar sobre objetos, situándolos o cambiándolos de posición. La forma más común de este tipo de robots son los brazos robóticos industriales. Por otro lado, la locomoción consiste en la capacidad de trasladarse de un punto a otro del espacio. A este tipo de robots se les llama robots móviles, y serán el objeto de partida de estudio para el desarrollo del prototipo del presente Trabajo Final de Máster. A continuación, en la Sección 2.1.1, se lleva a cabo un análisis de las principales tipologías de robots móviles.

2.1.1. Robots Móviles

La robótica móvil es un campo de investigación de la robótica cuyo principal objetivo consiste en el estudio del desplazamiento de un robot de una posición a otra: la locomoción. Dicha locomoción, depende fundamentalmente del entorno en el que se desplaza el robot, así como su finalidad, pudiendo encontrarse diversas disposiciones en base a esto.

Las principales configuraciones son: configuración basada en ruedas, configuración basada en patas, robots aéreos, robots acuáticos y robots ápodos.

La **configuración basada en ruedas** es el sistema de locomoción más utilizado y el que mayor carga puede transportar. En función del objetivo a conseguir, es posible configurar las ruedas de distintas formas: diferencial, sincronizada, de triciclo o de Ackerman [16].

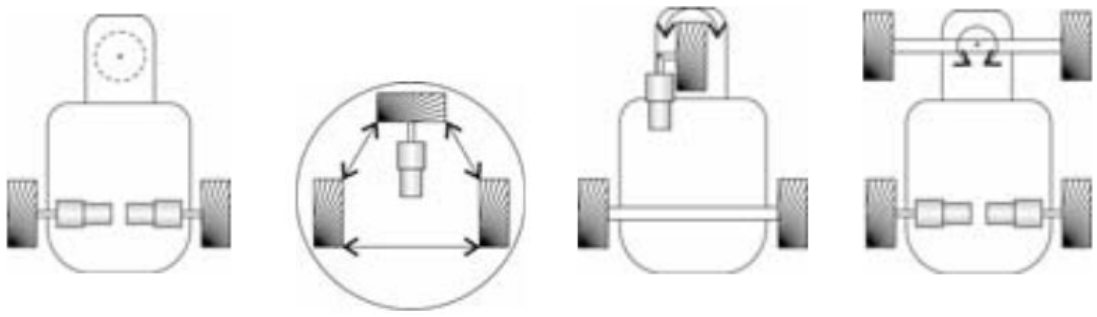
La configuración diferencial cuenta con dos ruedas motrices situadas diametralmente opuestas en un eje. Es la configuración más sencilla pero inestable, y es por ello que se incorpora un tercer punto de apoyo -delantero- adicional para ganar estabilidad.

Por el contrario, en la configuración sincronizada todas las ruedas rotan en la misma dirección y velocidad. Esta configuración tiene una elevada complejidad, ya que requiere una gran sincronización de todas las ruedas en todo momento. No obstante, a pesar de su dificultad, ofrece unas prestaciones muy eficientes.

Por otro lado, la configuración de triciclo consiste en dos ruedas motrices en combinación con una rueda delantera directriz. Es un mecanismo relativamente simple y no requiere el control de la velocidad.

Por último, la configuración de Ackerman cuenta con dos ruedas traseras y dos delanteras. Esta configuración es compleja pero muy estable, y es la tipología más utilizada debido a la capacidad de configurar las ruedas de diversas formas, según interese que se comporten como motrices o directrices.

Las cuatro configuraciones mencionadas para robots con ruedas, pueden observarse a continuación en la Figura 2.1 [17].



configuración diferencial | *configuración sincronizada* | *configuración de triciclo* | *configuración de Ackerman*

Figura 2.1: Principales configuraciones para robots móviles con ruedas (*Fuente: repositorioinstitucional.buap.mx*)

Por otro lado, la **configuración basada en patas o articulada** surge como consecuencia de la necesidad de llevar a cabo movimientos a través de superficies irregulares donde las ruedas no funcionarían adecuadamente. Su funcionamiento imita al de algunos animales, ya sean humanos -robots bípedos-, perros -robots cuadrúpedos- o algunos insectos -hexápodos u octópodos- como las arañas.

Para este tipo de locomoción, es imprescindible considerar ciertos parámetros como la posición, velocidad y equilibrio. Algunos ejemplos de robots con configuración basada en patas se muestran en la Figura 2.2 [18].

Figura 2.2: Robots con locomoción articulada basada en patas (*Fuente: researchgate.net*)

Los **robots aéreos y acuáticos** confieren la capacidad de desplazamiento en medios no terrestres. Los primeros son vehículos aéreos no tripulados, dirigidos por radio control o por un programa previamente cargado. Son más conocidos como UAV[2]. Igualmente, los acuáticos son también vehículos no tripulados diseñados para realizar diversas misiones bajo el agua.

Por último, la **configuración ápoda** es aquella en la que los robots carecen de extremidades o ruedas para desplazarse. Su locomoción se basa en la deformación de su propia estructura, e intenta imitar el movimiento de animales como serpientes, gusanos o caracoles. Concretamente, esta configuración es en torno a la cual gira el desarrollo del presente Trabajo Final de Máster, y por ello la que se desarrolla a continuación en mayor profundidad.

[2] Se refiere a un *Unmmanded Aerial Vehicle* -en castellano, vehículo aéreo no tripulado-.

2.1.2. Robots Ápodos

Uno de los principales retos de la robótica es conseguir desarrollar un robot capaz de ser estable y desplazarse por cualquier tipo de medio y terreno. Esta preocupación es la que ha dado lugar al desarrollo de los robots ápodos [19].

Los robots ápodos son aquellos que carecen de patas o extremidades para desplazarse. El movimiento lo producen gracias a la deformación de su propio cuerpo, aumentando considerablemente la estabilidad en comparación con los robots tradicionales que dependían de ruedas o patas. Este tipo de robots se bio-inspiran en el desplazamiento de diversos animales ápodos, predominando entre ellos los robots inspirados en los movimientos de las serpientes [20].

Su configuración mecánica se basa en la subdivisión longitudinal de la estructura principal en diversos módulos o segmentos. De esta forma, en cada punto de unión entre dos módulos consecutivos puede incorporarse una nueva articulación en el prototipo. Además, esto posibilita la implementación de un control diferenciado de cada uno de los mencionados segmentos.

La modularidad de dicha configuración, le confiere la capacidad de adoptar multitud de formas, permitiendo sortear terrenos al introducirse por huecos y agujeros.

Además, gracias a su reducida sección -en comparación con su longitud- posibilita ubicar el centro de gravedad del robot muy cerca del suelo, aspecto muy favorable en términos de estabilidad. No obstante, a pesar de las evidentes ventajas que confiere la locomoción ápoda, la caracterización de dichos movimientos no resulta nada sencilla, y es un campo en pleno desarrollo.

Y es esta la principal razón por la cual, hoy en día, no es posible encontrar prototipos comerciales de robots serpiente a un coste asequible, ni mucho menos algoritmos de locomoción eficientes -en comparación con el resto de tipologías-.

2.1.3. Aplicaciones

Tal y como se menciona en la anterior Sección 2.1.2, la principal ventaja de los robots ápodos consiste en su versatilidad de desplazamiento ante diferentes entornos y medios. Esto cobra un especial interés en aplicaciones donde el entorno no es conocido.

Una de las principales aplicaciones que tiene esta configuración, es su uso en misiones y maniobras de búsqueda y rescate en derrumbamientos o catástrofes. La combinación de una gran estabilidad junto con una pequeña sección, les confiere la capacidad de introducirse en agujeros pequeños y terrenos irregulares difícilmente accesibles para una persona.

Tras un derrumbamiento, la gente puede quedar atrapada bajo piedras y escombros. Mas allá de la clara ventaja de no poner en riesgo la vida de ninguna persona durante el reconocimiento inicial del terreno, el uso de robots apodos permitiría reducir los largos procedimientos y protocolos de maniobra, derivados de la irregularidad del terreno, la ausencia de espacio, o el desconocimiento de la situación interior del escenario [21].

Esto facilitaría la búsqueda de supervivientes, proporcionando información sobre las zonas en las que se encuentren personas atrapadas para así centrar el salvamento en estos puntos, pero también para identificar zonas libres de supervivientes, permitiendo así realizar labores de excavación que agilicen el proceso.

Otra de las grandes aplicaciones donde resultaría potencialmente útil, es en la exploración espacial. Debido al desconocimiento del terreno de los cuerpos espaciales que se quieren explorar, resulta muy útil enviar un robot con una configuración capaz de ser estable en cualquier medio y terreno por muy abrupto que sea. También resulta de gran utilidad contar con la posibilidad de introducirse por cualquier orificio para conseguir el mayor número de información posible del entorno. Por ello, actualmente el centro JPL de la NASA[3] está desarrollando un prototipo de serpiente robótica para la exploración en sus misiones espaciales [22].

Aparte de las aplicaciones mencionadas, la configuración ápoda tiene también utilidad en el ámbito industrial, en trabajos de mantenimiento para la inspección de tuberías o puentes. Así, el robot es capaz de introducirse por el interior de los conductos, evitando posibles obstáculos y detectando cualquier tipo de problema [23].

Por último, es una tecnología con muchas posibilidades a futuro. A medida que dicha tipología de movimiento carente de extremidades vaya desarrollándose y consigan implementarse técnicas más eficientes y económicamente competitivas, esta tecnología se extenderá como solución a un abanico de campos de aplicación mucho más extenso.

Un ejemplo es la medicina, donde la utilización de un robot tipo serpiente podría llegar a proporcionar la posibilidad de moverse por el interior del cuerpo humano con el fin de inspeccionar su estado o detectar posibles anomalías. Uno de los principales campos de aplicación en los que podría suponer un antes y un después de la medicina, sería en procedimientos de endoscopio o colonoscopia.

[3] Se refiere al *Jet Propulsion Laboratory* -en castellano, Laboratorio de Propulsión a Chorro-. Se trata del primer centro de la NASA dedicado a la exploración robótica del espacio.

2.2. BASE FISIOLÓGICA DE LA LOCOMOCIÓN APODA

Para poder entender la base fisiológica de la locomoción aplicada a robots ápodos, primero es necesario comprender el concepto de locomoción.

La **locomoción** es la capacidad de los seres vivos de desplazarse de un punto a otro del espacio. Dicha locomoción varía en función de la especie y del terreno en el que se lleve a cabo. Es por ello por lo que existen numerosos tipos distintos de locomoción animal en función de la especie.

No obstante, para poder considerarse locomoción, son necesarios dos factores: mecanismos de propulsión y mecanismos de control. Los mecanismos de propulsión son aquellos que involucran la estructura contráctil[4] de los animales, que en la mayoría de los casos utilizan sus propios músculos para generar fuerzas de propulsión. En cuanto a la cantidad, velocidad y posición de las contracciones, estas son coordinadas por el sistema nervioso, que es el responsable del control del movimiento [24].

En la naturaleza, la locomoción de cada especie se ha adaptado al ambiente en el que habita. Por ello, pueden distinguirse varios tipos de locomoción en función del medio por el que se desplazan. La principal subdivisión es: terrestre, acuática o aérea. No obstante, esta clasificación no es excluyente, ya que hay animales que pueden desplazarse por diferentes medios indistintamente.

En lo referente a la locomoción terrestre, es la más ampliamente desarrollada y estudiada de todas, puesto que es la que los propios seres humanos utilizamos. Se distinguen dos tipos de locomoción terrestre: la que es llevada mediante patas (seres humanos y otros) y la que se impulsa con el propio cuerpo o locomoción ápoda (serpientes, orugas, gusanos...) [25].

Con respecto a la locomoción ápoda, se entiende como aquella practicada por animales terrestres que se propulsan mediante su cuerpo, ya sea por la ausencia de patas, o porque les sea más útil utilizar el cuerpo para el desplazamiento debido a su disposición.

En la naturaleza hay una gran parte de los seres vivos que carecen de patas, ya sea porque las han ido perdiendo a lo largo de su evolución o porque nunca han dispuesto de extremidades. Los grupos con mayor predominio de animales ápodos son los anfibios, los réptiles y los insectos. En todos ellos, la evolución de su estructura ósea ha permitido observar una clara relación entre la carencia de patas y un alargamiento considerable de su cuerpo, como consecuencia de la necesidad de propulsarse [26].

Es por ello por lo que la locomoción ápoda es una técnica fundamentalmente basada en el deslizamiento o arrastre sobre la superficie en la que, gracias a las fuerzas de rozamiento, se logra impulsar el cuerpo hacia delante.

[4] Se refiere a la capacidad de contracción y elongación de la estructura de algunos seres vivos, utilizada como medio para su desplazamiento.

2.2.1. Ofidios: tipologías de movimiento

El presente Trabajo Final de Máster se centra en la locomoción ápoda de los ofidios[5] o serpientes. Las serpientes presentan diferentes modos de desplazamiento en función del entorno y terreno por el que se quieran desplazar. Los cuatro modos de locomoción de serpientes más comunes son: movimiento serpentino, movimiento lateral, movimiento de concertina y movimiento rectilíneo [27].

El Serpenteo o movimiento serpentino, es el movimiento típico de las serpientes, donde el animal se desplaza mediante movimientos ondulatorios. Mediante la contracción y dilatación de sus músculos, son capaces de curvar su cuerpo de un lado a otro, creando un movimiento en forma de S.

Para poder realizar este movimiento, es necesario que haya puntos de resistencia en la superficie para poder empujarse. Estos puntos de resistencia son los que se observan coloreados en azul en la siguiente Figura 2.3. Se trata de rocas, ramas, desniveles o incluso la propia agua si se trata de movimientos marinos. Sin embargo, la necesidad de tener puntos de apoyo hace que este movimiento no sea muy eficaz en superficies resbaladizas. En cuanto a la velocidad, esta depende de la longitud de onda que genera con su cuerpo para realizar el desplazamiento. Cuanto menor es la amplitud del movimiento, mayor es la velocidad de desplazamiento de avance.

Figura 2.3: Movimiento serpentino o Serpenteo (*Fuente: deserpientes.net*)

En lo referente al serpenteo lateral, este utiliza solo dos puntos de apoyo. La serpiente levanta la cabeza y la desplaza en forma de arco desde el punto de levantamiento u origen, hasta el punto de destino. Cuando apoya la cabeza, desplaza el resto del cuerpo moviéndose hacia los lados. Una vez ha llegado al destino, vuelve a levantar la cabeza y desplazarla hasta el siguiente punto, repitiendo todo el movimiento de nuevo. Al tener solo dos puntos de apoyo, la marca que deja en la arena por la que se desplaza tiene una forma de 'J'.

Esta forma de desplazarse es la que menos se ve afectada por el tipo de superficie, ya que puede ser usada para deslizarse por superficies más resbaladizas y no tan firmes.

Tanto el movimiento de avance como la marca que dejan tras su paso, pueden observarse a continuación en la Figura 2.4. En ella, se pueden observar -coloreados en azul- los puntos de apoyo en cada momento del movimiento lateral, así como las marcas finales en J que dejan en la arena tras su paso.

[5] Suborden de reptiles carentes de extremidades, que poseen una boca dilatable y un cuerpo largo y estrecho cubierto por escamas.

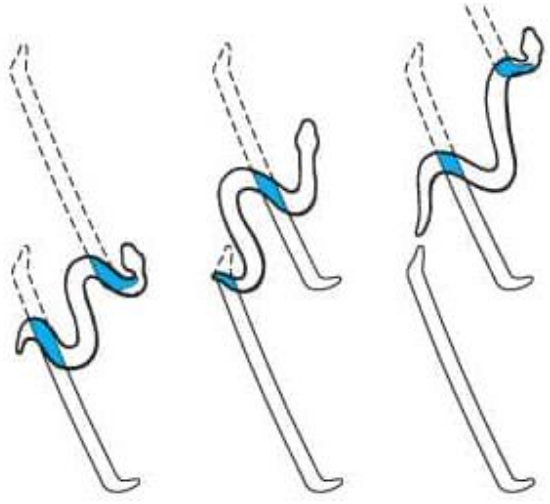


Figura 2.4: Movimiento lateral (*Fuente: deserpientes.net*)

A diferencia de los dos movimientos anteriores -que se realizan en el plano horizontal-, el movimiento de acordeón o concertina se utiliza principalmente cuando se pretende realizar un movimiento vertical, trepar o moverse por espacios reducidos. Este tipo de locomoción consiste en estirar la cabeza y parte posterior del cuerpo a lo largo de la superficie vertical hasta el punto de destino, donde se agarra a la superficie con sus escamas ventrales.

Una vez ha conseguido un agarre estable, tira de su extremo trasero hacia arriba. En este punto, la serpiente vuelve a estirar la cabeza por la superficie vertical para conseguir otro punto de agarre y volver a realizar todo el movimiento. Para conseguir un mejor agarre, la parte central del cuerpo se agrupa en curvas para permitir que las escamas ventrales puedan conseguir un agarre más firme a la superficie.

El funcionamiento descrito, se observa representado en la siguiente Figura 2.5.

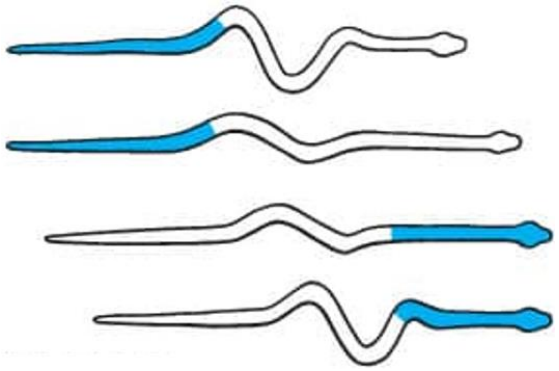


Figura 2.5: Movimiento de concertina (*Fuente: deserpientes.net*)

Por último, el movimiento rectilíneo consiste en un movimiento lento en línea recta. El funcionamiento se basa en curvas muy pequeñas y hacia arriba y abajo en vez de lado a lado, tal y como se observa en la Figura 2.6.

Figura 2.6: Movimiento rectilíneo (*Fuente: deserpientes.net*)

De esta forma, la serpiente contrae y extiende sus músculos en cada pequeña oscilación y utiliza esto para ir desplazándose en línea recta. Al tratarse de una locomoción muy lenta, es el movimiento que utilizan las serpientes cuando están acechando a sus víctimas [28].

2.2.2. Otros tipos de movimientos ápodos

Las serpientes no son los únicos seres vivos en utilizar una locomoción ápoda. Un ejemplo de ello son las lombrices o los caracoles, los cuales se desplazan por movimientos peristálticos. Es un movimiento similar al que produce nuestro esófago para empujar el bolo alimenticio por su interior [29].

El movimiento consiste en provocar cambios en las dimensiones de su cuerpo, donde el animal contrae los músculos de una parte específica del cuerpo -marcados en negro en la Figura 2.7-, aumentando el diámetro de esa sección y disminuyendo la distancia longitudinal.

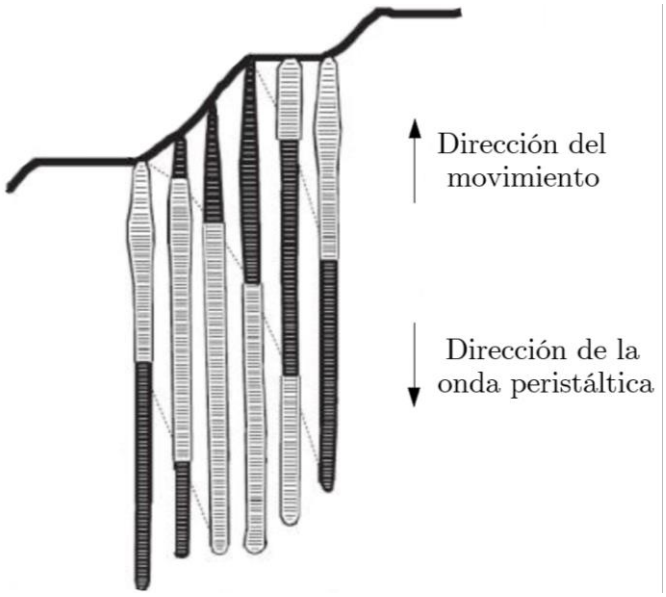


Figura 2.7: Movimiento peristáltico (*Fuente: ambientech.org*)

La onda de contracciones de músculos circulares se va propagando a lo largo del cuerpo. Cuando sobrepasa la mitad, los músculos de la parte anterior se relajan, consiguiendo una mayor longitud y por tanto un avance. Una vez la onda de contracciones ha llegado al final del cuerpo, el ciclo comienza de nuevo para así realizar un nuevo avance [30].

Por otro lado, animales como orugas o sanguijuelas utilizan un sistema de dos puntos de apoyo. La parte trasera de su cuerpo se ancla al suelo e impulsa la parte posterior hacia el frente. Una vez extendida, fija la zona posterior al suelo y retrae la parte trasera hasta la posición donde se encuentra el segmento posterior. En este punto, se vuelve a iniciar de nuevo el proceso para seguir avanzando [31].

Una representación del movimiento secuencial descrito, puede observarse a continuación en la Figura 2.8 [32].

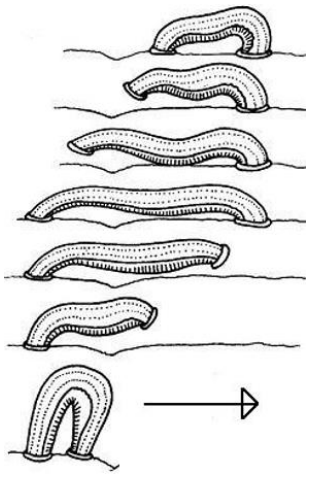


Figura 2.8: Movimiento de arrastre con dos puntos de apoyo (*Fuente: bioscripts.net*)

2.2.3. Comparativa de distintos métodos de locomoción ápoda

A modo de resumen y como conclusión de la presente sección, en la Tabla 2.1 que se muestra a continuación, puede observarse un análisis comparativo de los seis movimientos ápodos principales, en función de parámetros como la adaptabilidad a diferentes entornos, la velocidad del movimiento, la eficiencia y la necesidad de fricción. Este último es un parámetro importante a tener en cuenta debido a que el animal gasta una mayor energía al necesitar vencer la fricción.

Tabla 2.1: Comparativa de distintos métodos de locomoción ápoda

Método de Locomoción	Adaptabilidad al entorno	Necesidad de Fricción	Velocidad	Eficiencia
Serpentino	Media	Alta	Media	Alta
Lateral	Alta	Baja	Media	Media
Concertina	Media	Media	Media	Media
Rectilíneo	Baja	Media	Baja	Baja
Peristáltico	Media	Media	Baja	Media
2 puntos de apoyo	Alta	Baja	Baja	Baja

A pesar de que el movimiento serpentino tiene la desventaja de necesitar fricción para poder realizar el movimiento, tiene una gran eficiencia y el resto de los parámetros son aceptables, lo que hace que en su conjunto sea la mejor opción. Por ello, se ha escogido implementar dicho método de locomoción para la realización del presente trabajo.

2.3. LOCOMOCIÓN ÁPODA ARTIFICIAL: ANTECEDENTES

La locomoción ápoda artificial consiste en la imitación de los modos de movimientos descritos anteriormente en la Sección 2.2.1 de forma que puedan ser realizados por un robot o máquina móvil. Tienen el mismo principio de funcionamiento que las serpientes, ya que utilizan su cuerpo mecánico para transformar la fricción de su cuerpo contra el medio en un movimiento de avance.

Para poder entender el estado actual del estudio de robots ápodos, es necesario conocer su evolución y antecedentes durante las últimas décadas. A continuación, se exponen los modelos constructivos ápodos más importantes creados hasta la fecha, y que sin duda han marcado e influenciado enormemente el desarrollo de la industria de la robótica móvil que conocemos hoy en día.

2.3.1. Robot ACM III: Instituto Tecnológico de Tokio

El precursor del estudio de robots ápodos y la biomecánica de las serpientes fue Shigeo Hirose, profesor en el Instituto Tecnológico de Tokio. En 1976 implementó el primer prototipo de robot serpiente llamado ACM-III (*Active Cord Mechanims).*

El prototipo que puede observarse en la Figura 2.9, incluye un total de 20 articulaciones, cada una controlada por un servomotor independiente que fija el ángulo de rotación de su correspondiente segmento.

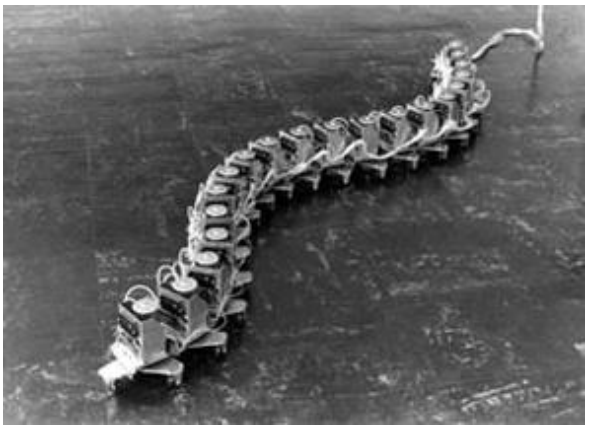


Figura 2.9: Prototipo ACM-III , Instituto Tecnológico de Tokio (*Fuente: researchgate.net*)

Además, para facilitar el movimiento, este robot está equipado con unas ruedas pasivas que envuelven todo su cuerpo y otorgan al prototipo la fricción característica de la piel de la serpiente al hacer que el coeficiente de rozamiento en la dirección tangencial sea muy pequeño frente a la dirección normal. Este robot posee una longitud de 2.00 m, un peso de 28 kg y es capaz de alcanzar una velocidad de hasta 40 cm/s [33].

A lo largo del tiempo, el equipo de investigadores de Hirose desarrollo diversas generaciones del prototipo de robot serpiente. Las más significativas fueron ACM-R3 y ACM-R5.

La primera de ellas, el ACM-R3 consiste en un robot modular con una longitud total de 1.80 m y un peso aproximado de 12 kg (véase la Figura 2.10). La configuración de esta versión consiste en alternar 90° la orientación de módulos consecutivos, de forma que se consigue un movimiento en las tres dimensiones. De forma análoga a la versión inicial, cuenta con ruedas para facilitar el movimiento. Sin embargo, la incorporación de estas ruedas limita el giro de cada articulación a 60°.

Figura 2.10: Prototipo ACM-R3, Instituto Tecnológico de Tokio (*Fuente: researchgate.net*)

En cuanto al prototipo ACM-R5, este fue diseñado y construido en el año 2005 y es uno de los robots serpiente más avanzados hasta la actualidad. Tal y como se puede observar en la Figura 2.11, se trata de un prototipo de robot anfibio que cuenta con propulsión para el movimiento terrestre y además cuenta con un total de cuatro aletas fijas para permitir el desplazamiento acuático [34].

Figura 2.11: Prototipo ACM-R5, Instituto Tecnológico de Tokio (*Fuente: researchgate.net*)

2.3.2. Robot *Uncle Sam*: Universidad *Carnegie Mellon* (CMU)

El trabajo realizado por Hirose sirvió de inspiración para muchos científicos alrededor del mundo, que comenzaron a seguir su línea de investigación de robots ápodos creando sus propios prototipos. Uno de ellos es Kevin Downling, quien estudió los robos ápodos desarrollando la generación automática de modos de caminar de las serpientes robóticas, así como la planificación de los movimientos y su posicionamiento.

Durante los últimos años han desarrollado diversos prototipos, algunos de ellos compuestos por 16 módulos idénticos. El modelo más destacado es el mostrado en la Figura 2.12, robot conocido como "*Uncle Sam*" [35].

Figura 2.12: Prototipo "*Uncle Sam*", CMU (*Fuente: biorobotics.ri.cmu.edu*)

2.3.3. Robot *Amphibot*: Grupo de robótica bio-inspirada del EPFL

La Escuela Politécnica Federal de Laussane (*EPFL*) desarrolló en el 2004 el robot llamado *Amphibot*. Dicho robot fue diseñado con el fin de construir un robot serpiente para tareas al aire libre y de inspección. Además, también tenía unos objetivos secundarios como utilizarlo como banco de pruebas para nuevos tipos de controladores adaptativos[6] basándose en el concepto de generadores de patrones centrales e investigación de hipótesis sobre cómo se implementan las redes neuronales de control de la locomoción en animales reales.

Amphibot es un robot planar compuesto por 8 módulos, de un grado de libertad cada uno, que se mueven paralelamente al suelo (véase la Figura 2.13). Cada módulo cuenta con un microcontrolador, un motor y una batería.

Figura 2.13: Prototipo *Amphibot*, EPFL (*Fuente: EPFL*)

[6] Se refiere a aquellos controladores no lineales donde el estado del proceso puede dividirse en dos escalas de tiempo que evolucionen a diferente velocidad [36].

Su diseño es intencionadamente estanco, con el objetivo de permitir su movimiento por un medio tanto terrestre como acuático. El movimiento consiste en realizar ondulaciones con su propio cuerpo, permitiéndole nadar en el agua y, con ayuda de unas ruedas, desplazarse por la superficie terrestre.

Posteriormente se hizo una nueva versión, *Amphibot II* (véase la Figura 2.14), en la que se mejoraron las características mecánicas con respecto al modelo anterior, haciendo más compactos los módulos y añadiendo patas para su movimiento por tierra. De esta forma, el movimiento terrestre se aleja del movimiento de una serpiente y se acerca más a una salamandra [37].

Figura 2.14: Prototipo *Amphibot II*, EPFL (*Fuente: EPFL*)

2.3.4. Robot S5: Gavin Miller

A lo largo de su etapa profesional, el ingeniero especializado en inteligencia artificial Gavin Miller ha desarrollado hasta siete versiones distintas de robots ápodos. Uno de los más destacables es el prototipo S5 realizado en 1998 (véase la Figura 2.15).

Este robot está compuesto por 64 articulaciones, 8 servomotores, 42 baterías y 4 canales de radio control. Sus piezas están realizadas a medida con fresadora de control numérico que le otorga una mayor precisión y sección transversal más pequeña.

Figura 2.15: Prototipo S5, Gavin Miller (*Fuente: snakerobots.com*)

Al contar con un número tan elevado de articulaciones y la disminución de la sección transversal, el prototipo S5 adquiere unas dimensiones similares a las serpientes reales. Sin embargo, este incremento de longitud provoca que el diseño plantee mayores exigencias en cuanto a cableado y capacidad de control. Respecto a su configuración, el robot consta de 64 articulaciones, cada una de las cuales con dos grados de libertad.

Cada segmento incorpora dos servomotores orientados de forma opuesta[7], lo que le permite realizar ondulaciones tanto en el plano horizontal como vertical. Asimismo, cuenta con una rueda en la parte inferior central de cada módulo, para facilitar el movimiento terrestre [38].

2.3.5. Otros robots ápodos de interés

A medida que el estudio de los robots ápodos ha ido avanzando, esto ha dado lugar a la aparición de una nueva rama de investigación en torno a la posibilidad de realizar el movimiento mediante ruedas u orugas. Este tipo de locomoción no está bio-inspirado en ningún animal existente en la naturaleza, por lo que queda fuera del alcance de este proyecto. Sin embargo, al tratarse de robots ápodos modulares, su disposición y conexión entre segmentos sí que resulta de gran interés.

El creador del ACM-III, también desarrolló un prototipo de robot ápodo propulsado consistente en varias estructuras encadenadas con módulos autopropulsados. Este robot destaca por la versatilidad de sus módulos, ya que se pueden montar, desmontar o intercambiar de forma rápida y sencilla.

Por otro lado, el laboratorio de robótica móvil de la Universidad de Michigan desarrolló el "*OmniTread*". Se trata de un robot autopropulsado muy avanzado, ya que utiliza articulaciones neumáticas que le otorgan una gran fuerza. Dicho robot está formado por 5 módulos hexaédricos, donde en 4 de sus caras exteriores se han colocado orugas para el movimiento [39].

Igualmente, el laboratorio de sistemas inteligentes del EPFL, también ha seguido esta línea de investigación con la realización del "*Swarm-Bot*". Este prototipo está formado por pequeños robots uni-modulares con capacidad de auto ensamblarse en función de la tarea a realizar [40].

[7] El autor no proporciona detalles de la configuración bajo la cual se disponen los mencionados servomotores enfrentados entre ellos.

2.4. GENERACION DE TRAYECTORIAS SERPENTINAS: MODELO MATEMÁTICO

El presente apartado trata de enfocar la generación de trayectorias desde un punto de vista analítico. En concreto, se realiza el estudio del movimiento serpentino ya que, como se concluye en la Sección 2.2.3, es el que comparativamente demuestra ser, para los parámetros de estudio[8], el más interesante y susceptible de implementarse.

A modo de recapitulación, dicho movimiento consiste en utilizar el movimiento del cuerpo para impulsarse, a través de la realización de curvas en forma de S. La implementación de un modelo matemático permite parametrizar los valores necesarios para obtener el algoritmo que controla el movimiento del robot. Este algoritmo, debe ser capaz de adaptarse al movimiento que se quiere realizar y varía en función del alcance del movimiento.

A continuación, se desarrolla el modelo matemático desarrollado para el robot serpiente objeto de estudio. Al final del presente capítulo, en la Sección 2.4.4 puede encontrarse un breve glosario con la identificación de todas las variables y parámetros utilizados en dicho modelo.

2.4.1. Curva serpentinoide

Los primeros estudios sobre el movimiento serpentino fueron desarrollados por el profesor Shigeo Hirose. Hirose realizó múltiples estudios empíricos sobre el movimiento biológico de las serpientes, creando un modelo matemático para explicar la curva serpentinoide. Dicha curva, modela el ángulo de doblaje (θ) que forma la dirección longitudinal del cuerpo de la serpiente, cuando esta se mueve formando una onda senoidal como la que genera la Ecuación 1 [41].

$$\theta(s) = \left(\frac{2\pi K \alpha_0}{L}\right) \cdot \sin\left(\frac{2\pi K}{L} \cdot s\right) ; \qquad \text{[Ec. 1]}$$

Donde el ángulo de doblaje 'θ' define el ángulo que forman las tangentes que pasan por dos puntos consecutivos de la senoide. La variable 's' es una variable continua que representa la distancia a lo largo del eje, y el parámetro 'K' define el número de oscilaciones de la senoide. El parámetro 'L' define la longitud total, mientras que 'α_0' es el ángulo inicial de la curva.

[8] Se refiere a parámetros como la adaptabilidad a diferentes entornos, la necesidad de fricción, o la velocidad y eficiencia del movimiento

Todos estos parámetros, se pueden observar con mayor claridad en la Figura 2.16.

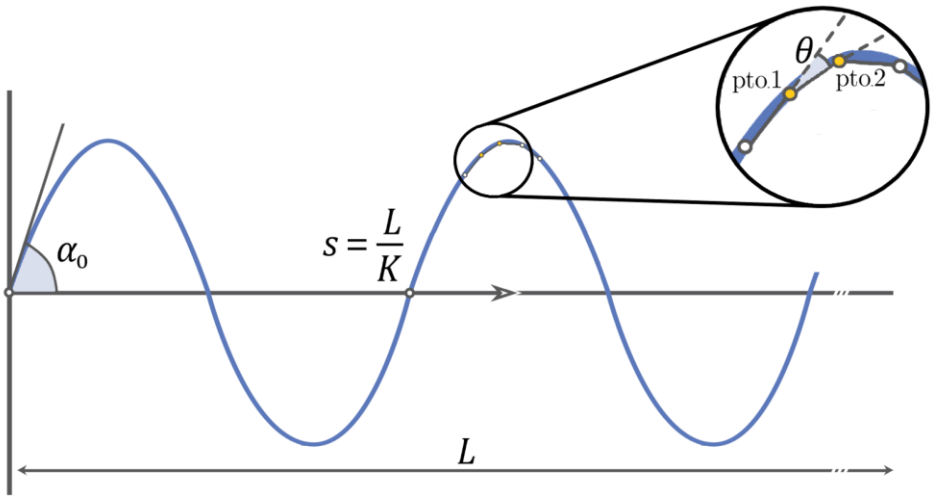


Figura 2.16: Parámetros de la curva serpentinoide, con detalle del ángulo de doblaje

2.4.2. Proyecciones cartesianas

Al realizar una curva serpentoidal ideal, los ángulos relativos de cada articulación varían. Para calcular la posición de cada articulación, se proyecta la curva serpentoidal sobre los ejes X e Y [42]. Dicha proyección se lleva a cabo mediante la integración de los senos y cosenos en función de la posición relativa de cada articulación (δ), tal y como se puede observar en las Ecuaciones 2 y 3 [43].

Ambas ecuaciones dependen de dos variables inciertas: la distancia 's' a lo largo del eje X -estudiada en la Sección 2.4.1- y un parámetro 'c' referente a la curvatura de la onda.

$$x(s,c) = \int_0^s \cos(\alpha_0 \cos\left(\frac{2\pi K}{L} \cdot \delta\right) + c\delta) d\delta \qquad \text{[Ec. 2]}$$

$$y(s,c) = \int_0^s \sin(\alpha_0 \cos\left(\frac{2\pi K}{L} \cdot \delta\right) + c\delta)\, d\delta \qquad \text{[Ec. 3]}$$

Para facilitar la parametrización, se agrupan los parámetros de la siguiente forma en función de su efecto. El parámetro 'a' determina la amplitud de la onda (véase la Ecuación 4), el parámetro 'b' determina el número de ondulaciones por unidad de longitud (véase la Ecuación 5) y el parámetro 'c' determina la dirección de la curva, en términos de una trayectoria circular (véase la Ecuación 6).

$$a = \alpha_0 \qquad \text{[Ec. 4]}$$

$$b = \frac{2\pi K}{L} \qquad \text{[Ec. 5]}$$

$$c = c \qquad \text{[Ec. 6]}$$

Por ende, las ecuaciones que expresan el movimiento de alcance de las curvas serpentoidales proyectadas sobre los ejes X e Y quedan tal y como se observa en las Ecuaciones 7 y 8.

$$x(s) = \int_0^s \cos\left(a\cos(b\delta) + c\delta\right) d\delta \qquad \text{[Ec. 7]}$$

$$y(s) = \int_0^s \sin(a\cos(b\delta) + c\delta)\, d\delta \qquad \text{[Ec. 8]}$$

Análisis del efecto de los parámetros sobre la curva serpentinoide

Para entender mejor el significado de los parámetros a, b y c descritos en el apartado anterior, se realizan diversos experimentos y comprobaciones para evidenciar su efecto en la ecuación con una muestra de 500 puntos.

Para ello, se fijan los parámetros de partida desde los que se va a realizar las variaciones, buscando valores enteros y fácilmente comparables entre ellos. Se toma una única onda (b=1) con una amplitud de onda igual a la unidad (a=1), imponiendo la condición de avance longitudinal sin giros (c=0).

Variando por separado cada uno de los parámetros dejando el resto fijo, pueden ir obteniéndose los sucesivos resultados que se muestran en la Figura 2.17 para el parámetro a, la Figura 2.19 para el parámetro b, y la Figura 2.20 para el parámetro c.

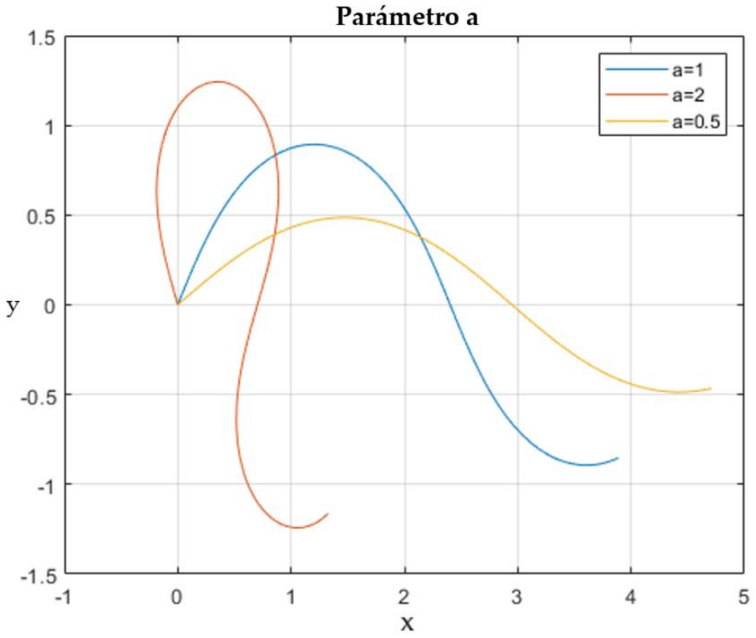


Figura 2.17: Estudio de dependencia de la amplitud (a) en el modelo (*Matlab*)

Puede observarse como, para valores de a inferiores a la unidad, el valor de amplitud se va reduciendo. Por el contrario, para valores de a superiores a la unidad, el valor de la amplitud aumenta al mismo tiempo que pierde la forma de senoide 'pura'.

A partir de dicha deformación, puede concluirse que el parámetro a posee un límite superior que no se podrá superar: donde coincidan los puntos de dos ondas consecutivas, tal y como se muestra en la Figura 2.18 coloreados en rojo. Destacar que este límite no es fijo, puesto que es dependiente del parámetro c. Si se fija un movimiento unidireccional sin rotaciones ($c = 0$), el valor límite de la amplitud es 2.11.

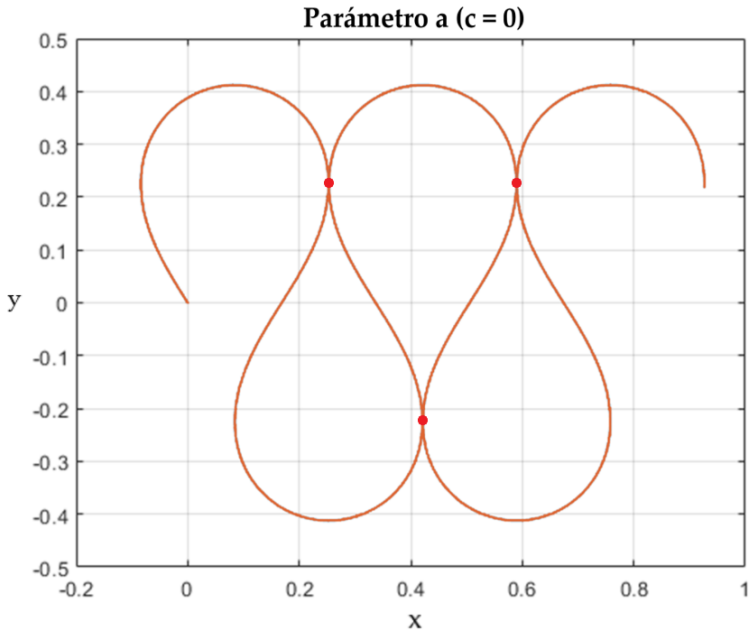


Figura 2.18: Estudio de la amplitud para un valor límite de $a = 2.11$ (*Matlab*)

Por otro lado, al comprobar el efecto del parámetro del número de ondas (b), se observa que, tal y como era de esperar, es proporcional al número de ondas que se realizan: cuanto mayor es dicho parámetro, más ondas se realizan en la trayectoria[9].

[9] Cabe destacar que cada unidad del parámetro b no corresponde con una onda completa, sino a aproximadamente tres cuartas partes de la onda.

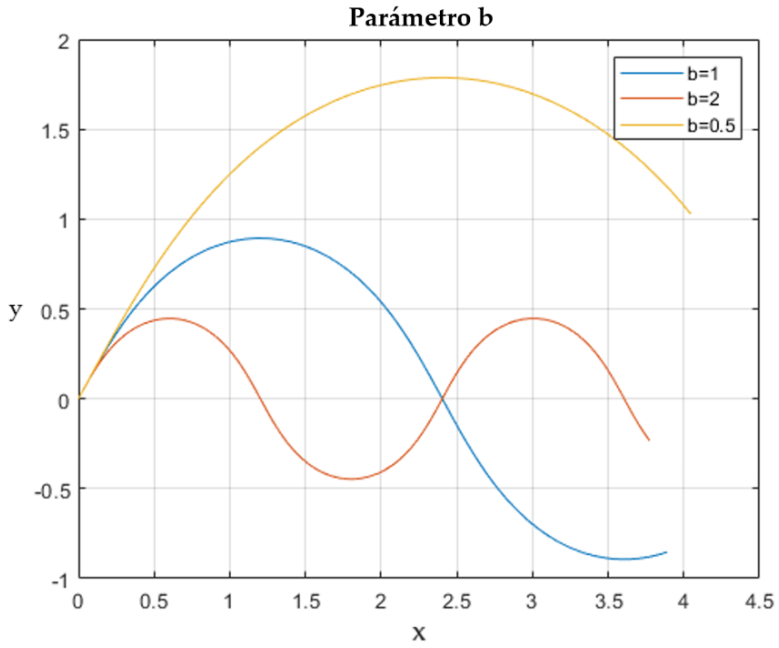


Figura 2.19: Estudio de dependencia del número de ondas (b) en el modelo (*Matlab*)

Por último, para estudiar la influencia que posee la dirección del movimiento en los resultados del modelo, se estudian seis casos distintos -ya que este es el parámetro más abstracto de los tres-.

Tal y como se observa en la Figura 2.20, para valores del parámetro c positivos la trayectoria va curvándose en el sentido inverso a las agujas del reloj, mientras que para valores negativos la trayectoria se curva siguiendo el sentido de las agujas del reloj.

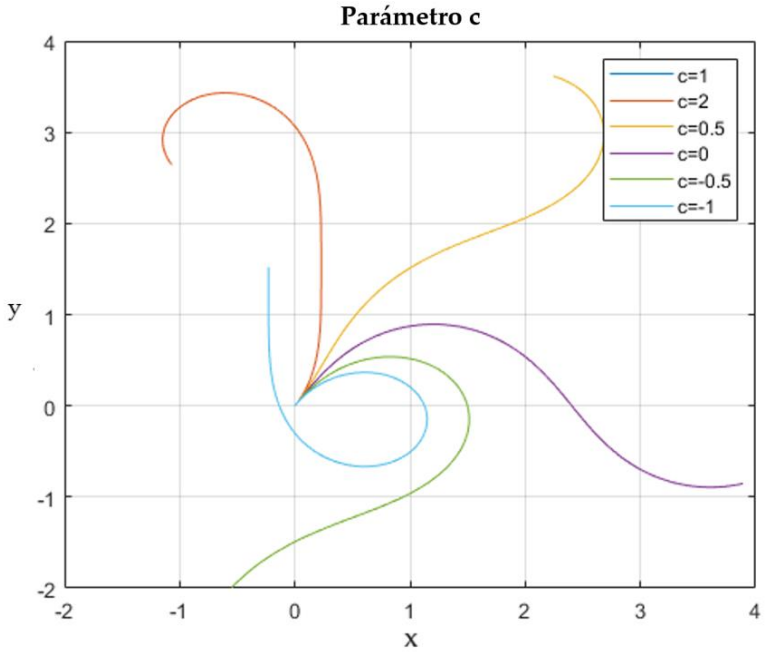


Figura 2.20: Estudio de dependencia de la dirección del movimiento (c) en el modelo (*Matlab*)

2.4.3. Proyecciones cartesianas aplicadas a cada articulación

Las anteriores Ecuaciones 7 y 8 corresponden al modelo del cuerpo entero de la serpiente. Dado que el presente Trabajo Final de Máster se centra en implementar un prototipo de robot modular con 'n' módulos rígidos independientes, es necesario conocer la posición X e Y de cada uno de ellos. Si se aproxima dicha curva teniendo en cuenta los n segmentos rígidos conectados, la posición X e Y de cada uno de los segmentos puede formularse tal y como se observa en las Ecuaciones 9 y 10 [43].

$$x_i = \sum_{k=1}^{i} \frac{1}{n} \cos\left(a \cos\left(\frac{kb}{n}\right) + \left(\frac{kc}{n}\right)\right), \qquad i \in \{1 \ldots n\} \qquad \text{[Ec. 9]}$$

$$y_i = \sum_{k=1}^{i} \frac{1}{n} \sin\left(a \cos\left(\frac{kb}{n}\right) + \left(\frac{kc}{n}\right)\right), \qquad i \in \{1 \ldots n\} \qquad \text{[Ec. 10]}$$

Representando ambas ecuaciones, podemos obtener un gráfico como el de la Figura 2.21, donde se ha aproximado la curva serpentinoide por medio de 'n' segmentos rectos, cada uno de los cuales definido por las coordenadas x_i e y_i.

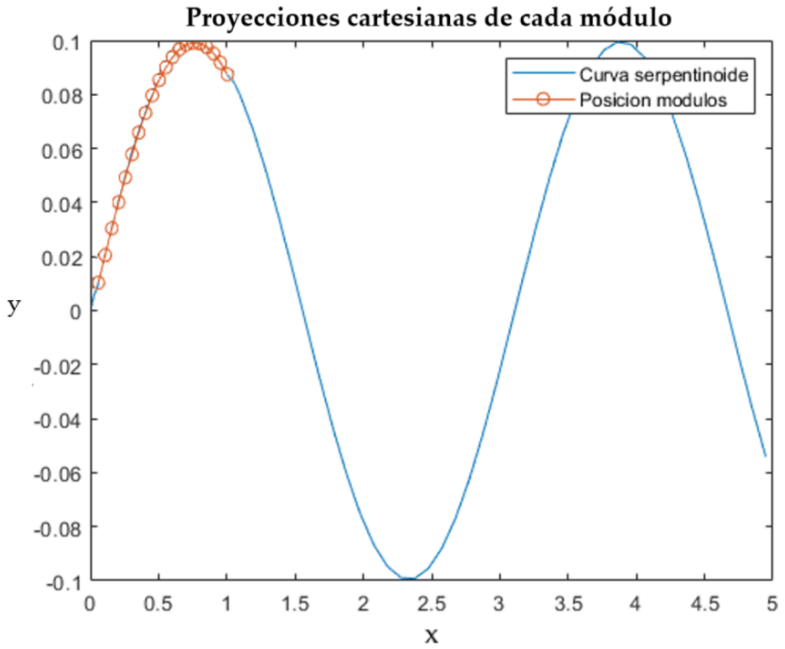


Figura 2.21: Proyecciones cartesianas de cada módulo sobre la curva serpentinoide (*Matlab*)

Donde, para cada módulo 'i', es posible definir un ángulo β_i que representa la posición de dicho módulo con respecto al eje horizontal.

Tal y como se observa en la Ecuación 11, la obtención de dicho ángulo puede obtenerse a partir de su tangente como:

$$\tan(\beta_i) = \frac{y_i - y_{i-1}}{x_i - x_{i-1}} = \frac{\sin\left(a\cos\left(\frac{ib}{n}\right) + \left(\frac{ic}{n}\right)\right)}{\cos\left(a\cos\left(\frac{ib}{n}\right) + \left(\frac{ic}{n}\right)\right)}, \qquad i \in \{1 \dots n\} \qquad \text{[Ec. 11]}$$

donde puede obtenerse de forma directa el ángulo β_i (véase la Ecuación 12) [44].

$$\beta_i = a\cos\left(\frac{ib}{n}\right) + \left(\frac{ic}{n}\right), \qquad i \in \{1 \dots n\} \qquad \text{[Ec. 12]}$$

2.4.4. Resumen de parámetros

Para una mayor comprensión de las ecuaciones descritas, pueden encontrarse recogidas a continuación en la Tabla 2.2, todas las variables y parámetros utilizados en el modelado matemático del presente capítulo.

Tabla 2.2: Parámetros utilizados para el desarrollo matemático de la locomoción ápoda

Parámetro	Definición
θ	Ángulo de doblaje: ángulo entre las tangentes de dos puntos
s	Variable continua que representa la distancia en cada momento
K	Número de ondas
L	Longitud total de la onda
α_0	Ángulo inicial de la onda
a	Parámetro que caracteriza la amplitud de la onda
b	Ondulaciones por unidad de longitud
c	Dirección de la curva
i	Número de módulo
x_i	Posición x de cada módulo
y_i	Posición y de cada módulo
n	Número de módulos
δ_i	Distancia relativa de cada segmento
β_i	Ángulo de cada segmento con la horizontal

CAPÍTULO 3. DISEÑO MECÁNICO DEL PROTOTIPO

3.1. ESTRUCTURA BÁSICA

Con el objetivo de llevar a cabo una validación experimental de las simulaciones y de los modelos desarrollados, se lleva a cabo el diseño de un prototipo de robot ápodo que, tal y como se expone en la Sección 2.1.2, se bio-inspira en una serpiente.

La estructura consiste en una arquitectura modular, ya que permite una mayor articulación con un nivel de complejidad relativamente bajo. Para ello, se utilizan un total de ocho módulos idénticos. La elección de dicho número de módulos tiene en cuenta diversos criterios. La razón principal es, que se estima que para conseguir representar una onda sinusoidal completa, hacen falta como mínimo cuatro módulos. Sin embargo, cuanto mayor sea el número de módulos, más se asemejará su comportamiento al de una serpiente real. Por otro lado, se tiene en cuenta la facilidad y adaptabilidad del montaje, así como minimizar el presupuesto.

Por ello, se llega a una solución de compromiso entre ambos criterios y se escoge ocho módulos, debido a que se pretende realizar el estudio del movimiento en dos direcciones, por lo que es necesario realizar dos ondas sinusoidales (véase con mayor detalle en la Sección 4.4).

Cada módulo está compuesto por cuatro piezas. La pieza principal es el servomotor, que ejerce el papel del "sistema locomotor" del módulo y es el que se encarga de realizar el movimiento. El resto de piezas son unas coberturas metálicas, que en lo que sigue denominaremos *Brackets*, que se encargan de proteger el servomotor y establecen el punto de conexión entre módulos consecutivos para formar la estructura del robot.

Respecto al resto del diseño, en el CAPÍTULO 6 se describe la elección del resto de componentes electrónicos, así como sus criterios de elección, cuya selección se lleva a cabo tras realizar la elección de la configuración óptima del prototipo.

3.2. SELECCIÓN DEL SISTEMA DE ACTUACIÓN

El sistema de actuación es uno de los elementos más importantes del diseño de prototipo, puesto que es quien se encarga de realizar el movimiento. Si se compara el prototipo con la serpiente real en la que se inspira, los actuadores equivalen a los músculos de la serpiente que ejercen las fuerzas necesarias para el desplazamiento.

3.2.1. Análisis de alternativas

Los actuadores de uso más común en la actualidad se pueden dividir en tres grupos: hidráulicos, neumáticos y eléctricos.

Los actuadores hidráulicos son los más antiguos de los tres, y su funcionamiento se basa en transformar la energía de presión de los líquidos -principalmente agua o aceite- en energía mecánica. Este tipo de actuador tiene la principal ventaja de producir una gran potencia. Sin embargo, se requieren instalaciones muy complejas para su funcionamiento y presentan un precio muy elevado [45].

Por otro lado, los actuadores neumáticos convierten la energía del aire comprimido en trabajo mecánico. Este tipo de actuadores cuenta con la ventaja de que son muy silenciosos y no requieren un gran mantenimiento. Suele utilizarse para aplicaciones industriales en las que no hace falta un nivel de precisión muy estricto [46].

Por último, los actuadores eléctricos tienen un funcionamiento mucho más simple que los anteriores. Se trata de un dispositivo que solo requiere de energía eléctrica para realizar el movimiento. Son actuadores altamente versátiles y con una amplia gama de modelos. Cuenta con la ventaja de ser un actuador muy preciso [46].

Además de los tres tipos mencionados, existen otros tipos de actuadores como los piezo-eléctricos o térmicos. No obstante, por facilidades de adquisición y coste para la implementación en un demostrador real, estos quedan fuera del alcance del proyecto.

Para las principales tipologías mencionadas, se lleva a cabo un análisis comparativo de las características de mayor interés para el demostrador, tal y como se observa a continuación en la Tabla 3.1.

Tabla 3.1: Tabla comparativa de las principales tipologías de actuadores

Parámetro	**Actuador Hidráulico**	**Actuador Neumático**	**Actuador Eléctrico**
Energía	Presión Fluidos	Presión Aire	Energía Eléctrica
Potencia	Elevada	Media	Media
Precisión	Medio	Media	Elevada
Precio	Elevado	Medio	Variedad de rango

Dado que el prototipo con el que se está trabajando no tiene un gran peso, la potencia del actuador no es un parámetro relevante para la aplicación desarrollada. Sin embargo, al tratar de reproducir el movimiento coordinado del prototipo en su conjunto a partir del control individual de cada módulo, la precisión de posicionamiento del actuador cobra un papel fundamental. Por todo esto, se decide incorporar actuadores de tipo eléctrico en el demostrador de serpiente ápoda.

Entrando más en detalle, de entre los distintos tipos de actuadores eléctricos, se selecciona un servomotor, ya que permite llevar a cabo un control preciso en términos de posición angular, aceleración y velocidad. Además, el servomotor incorpora la sensorización necesaria para llevar a cabo una retroalimentación de la posición, aspectos que no siempre pueden conseguirse con un motor eléctrico convencional.

Por último, mencionar que los servomotores tienen un costo reducido en comparación con otros actuadores, lo que permanece en la línea de minimizar el presupuesto que se está siguiendo durante todo el proyecto.

3.2.2. Modelo comercial seleccionado

Además del análisis anterior, es fundamental considerar que para poder reproducir una onda sinusoidal, es necesario que el ángulo de rotación del servomotor sea lo suficientemente amplio para permitir la rotación en ambos sentidos de las agujas del reloj.

Por todo lo mencionado, se selecciona el servomotor *'Hitec HS-422'*. Este modelo posee unas dimensiones de 40.6×19.8×36.6 mm y permite una rotación de hasta 90° en cada sentido, cumpliendo sobradamente los requerimientos necesarios. El resto de las especificaciones técnicas de interés del componente, se muestran a continuación en la Tabla 3.2. Para una información más detallada del modelo seleccionado, se recomienda la visualización del Anexo A.*1*.

Tabla 3.2: Especificaciones técnicas del servomotor "*Hitec HS-422*"

Parámetro	**Magnitud [ud.]**
Tipo de motor	3 polos
Rango de voltaje de operación	4.8-6.0 V
Velocidad de operación	0.2 s /60°
Par	46/57 oz./in[10].
Corriente con carga	8 mA
Corriente sin carga	150mA
Dimensiones	40.6×19.8×36.6 mm
Peso	45.5 g

[10] Se refiere a las unidades de onzas/pulgada -en inglés, *ounce/inch*-. Se trata de un sistema de unidades alternativo al Newton/metro.

3.3. SELECCIÓN DE COMPONENTES MECÁNICOS

Como se explica en la Sección 3.1, el prototipo a realizar cuenta con tres piezas mecánicas básicas, los '*Brackets*'. Para la elección de los *Brackets* se sigue el siguiente procedimiento.

En primer lugar, se tiene en cuenta la funcionalidad de cada *bracket*. Se necesita un *bracket* que abrace el servo motor a modo de protección, ya que es la parte más sensible del prototipo y la más susceptible a recibir golpes. Además, se necesitan dos piezas más que sirvan de conexión entre los *brackets* y de transmisor del movimiento del servomotor.

Por otro lado, es importante considerar el tamaño del servomotor. Dado que los *brackets* tienen que envolver al servomotor, este es quien marca el tamaño. Como se explica en la Sección 3.2, los servomotores escogidos son los "Hitec HS-422", con unas dimensiones de 40.6×19.8×36.6 mm. Por tanto, los brackets seleccionados deben ser coherentes con dichas medidas.

Otro de los aspectos a tener en cuenta, es la robustez de la pieza. Inicialmente se plantea llevar a cabo la fabricación de los brackets por impresión 3D. La impresión 3D ofrece una resistencia suficiente para cargas aplicadas perpendicularmente al plano de las capas de impresión, pero si la carga se aplica en cualquiera de las direcciones que conforman dicho plano, la fractura esta casi asegurada. Considerando además que en todos los brackets hay ángulos rectos en la pieza y que la sección de los mismos es muy reducida, hace que esta opción de fabricación quede descartada.

En base a todos los criterios anteriores, se escogen tres soportes de la familia de "ASB" del proveedor *Lynxmotion*. Estas piezas están fabricadas en aluminio anodizado negro de alta calidad, con lo que la robustez del prototipo queda garantizada. Además, dichos componentes tienen la ventaja de ser muy versátiles y reconfigurables para distintas configuraciones, lo cual es de gran utilidad para el estudio que se va a realizar.

En primer lugar, para la protección del servomotor, se elige el soporte de servo multiusos ASB-04 (véase la Figura 3.1). Este *bracket* se ubica en el centro de cada módulo, y está diseñado para contener un servo estándar en su interior.

Figura 3.1: Soporte de servo multiusos ASB-04 *(Fuente: Robotshop)*

En lo que sigue, esta pieza puede observarse referida en el documento con las siglas "*BA-X*", donde la letra '*B*' corresponde al tipo de pieza que es -en este caso un Bracket-, la letra '*A*' se refiere a la posición de la pieza -en este caso, posición Anterior-, y por último un número '*X*', que hace referencia al módulo al que pertenece.

Para una mayor comprensión de esta nomenclatura, se facilita un resumen de la terminología en la Sección 3.5.

Para la transmisión del movimiento, así como la unión entre módulos consecutivos se poseen dos soluciones constructivas dentro de la familia de *Lynxmotion*. Ambas son soportes en "C", con referencias ASB-09 y ASB-10. La única diferencia existente entre ambas piezas reside en la longitud de las zonas laterales. La primera de ellas posee unas dimensiones de 51×24×40mm, mientras que la segunda tiene unas dimensiones de 51×24×57mm.

Se opta por el modelo ASB-09 (véase la Figura 3.2) con el objetivo de tratar de minimizar la longitud no útil -considerando que el modelo ASB-10 proporciona un aumento de longitud que no aporta ningún beneficio, sino más bien genera un hueco vacío -.

Además, al tener una menor distancia desde la articulación hasta el punto de contacto con el siguiente modulo, las ondas sinusoidales reproducidas presentan un comportamiento más gradual y menos brusco.

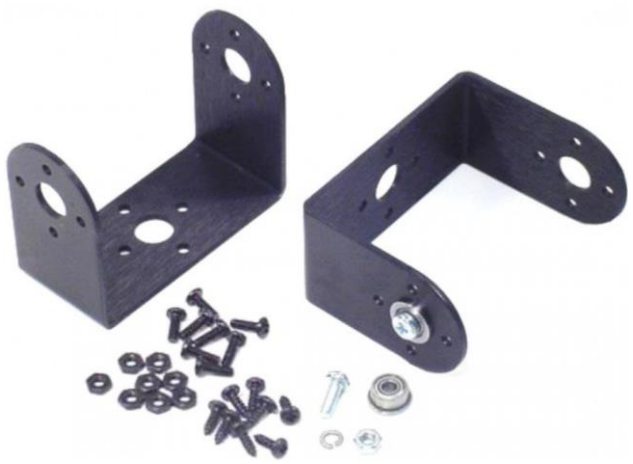


Figura 3.2: Soporte en C ASB-09 *(Fuente: Robotshop)*

Este bracket se ubica en la parte posterior del módulo y se une por un lado al eje del servomotor -transmitiendo el movimiento del servomotor- y por otro al módulo posterior -sirviendo de unión entre módulos consecutivos-.

De forma análoga al *bracket* anterior, de ahora en adelante se queda nombrado como "*BM-X*", donde en este caso la M hace referencia a la posición de la pieza -en este caso, posición Media-, y por último un número '*X*', que hace referencia al módulo al que pertenece.

Por último, la unión de los dos *brackets* anteriores dentro de un módulo, se lleva a cabo mediante soporte en "L" con referencia ASB-06 (véase la Figura 3.3). Esta pieza está diseñada para conectar un soporte "C" al extremo de un soporte servo multiusos. La finalidad de este *bracket* es realizar la unión entre el *bracket BA-X* y el *BP-X*. Se referirá a dicho *bracket* como "*BM-X*".

Figura 3.3: Soporte en L ASB-06 *(Fuente: Robotshop)*

3.4. DISEÑO DE LA ESTRUCTURA MECÁNICA

Tras la elección de los componentes mecánicos, se realiza el modelado y ensamblaje del prototipo mediante el software CAD *Solidworks.* Para ello, se importan los archivos CAD[11] de las piezas seleccionadas y un archivo de un servomotor estándar.

Cada módulo está compuesto por un servomotor y tres *brackets.* En primer lugar, se une el servomotor con el BA-X (pieza coloreada en azul en la Figura 3.4) mediante varias relaciones de posición de paralelismo de las caras y coincidencia para los agujeros destinados a los tornillos.

Tras limitar el movimiento de ambas piezas, se realiza el ensamblaje del *bracket* BM-X (pieza roja). La relación de posición más importante en el ensamblaje de ambos componentes es la coincidencia del eje del servomotor con el agujero en uno de los extremos del *brácket*, para poder permitir el movimiento circular de esta pieza.

En último lugar, se realiza la unión del *bracket* BP-X (pieza coloreada en verde en la Figura 3.4), mediante restricciones de paralelismo y concentricidad en los puntos destinados a la unión, mediante un tornillo, de los distintos *brackets.* El resultado del ensamblaje de un módulo completo se muestra en a Figura 3.4, donde se muestra una vista frontal y trasera del módulo. Además, para mayor detalle de este ensamblaje, se recomienda la visualización del Plano de Conjunto P01.

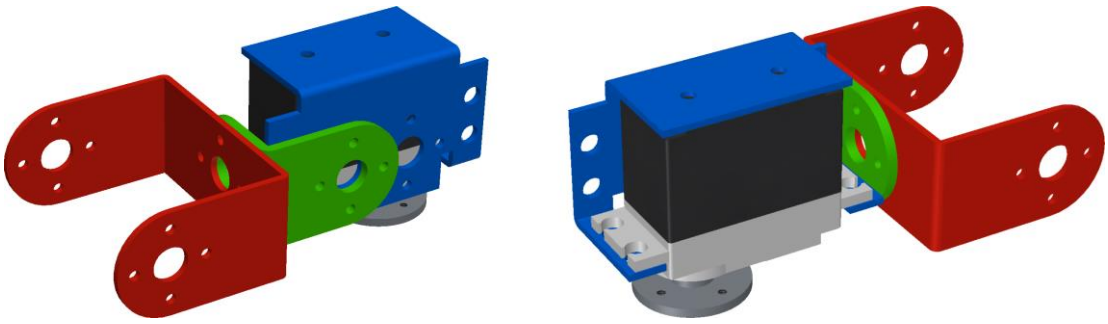


Figura 3.4: Vistas frontal y trasera de un módulo ensamblado (*Solidworks)*

Una vez realizado el ensamblaje de cada módulo se procede a realizar la unión de todos ellos. Como se quiere realizar el estudio de distintas configuraciones para obtener la configuración óptima para el movimiento, se han realizado dos ensamblajes distintos.

El primero de ellos consiste en un ensamblaje sencillo en que todos los módulos se unen en la misma posición, uno a continuación del otro. Para ello, se utiliza las relaciones de posición de paralelismo y concentricidad para unir los distintos módulos entre ellos. El resultado es el que se muestra en la Figura 3.5.

[11] Los archivos CAD de los componentes del ensamblado se han obtenido de la página oficial de *Lynxmotion* [47].

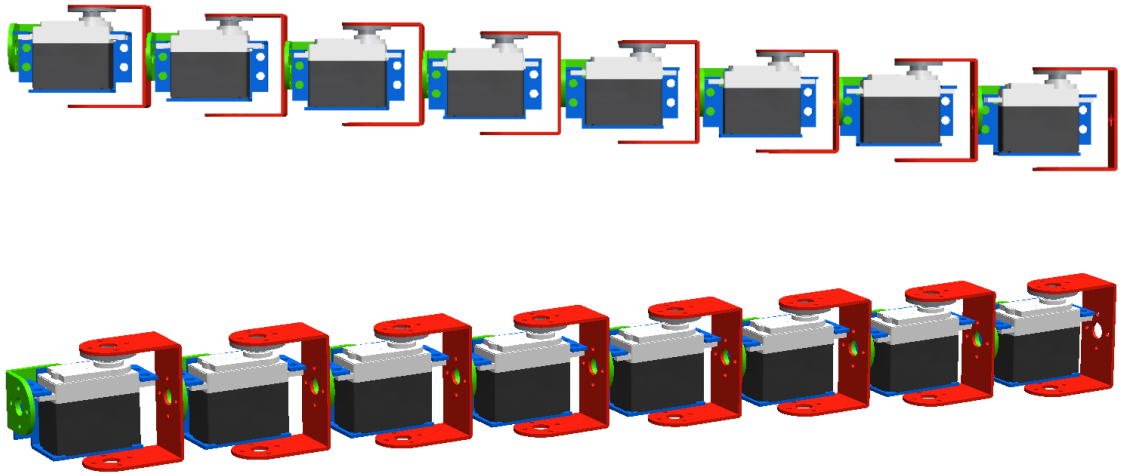


Figura 3.5: Ensamblaje en CAD de la configuración con igual orientación de todos los módulos *(Solidworks)*

Por otro lado, se realiza un ensamblaje más complejo consistente en alternar módulos orientados a 0 y 90°. El montaje resultante sería un módulo orientado a 0° unido a un módulo orientado a 90°, repitiéndose este patrón 4 veces a lo largo del prototipo. Su estructura se observa en la Figura 3.6.

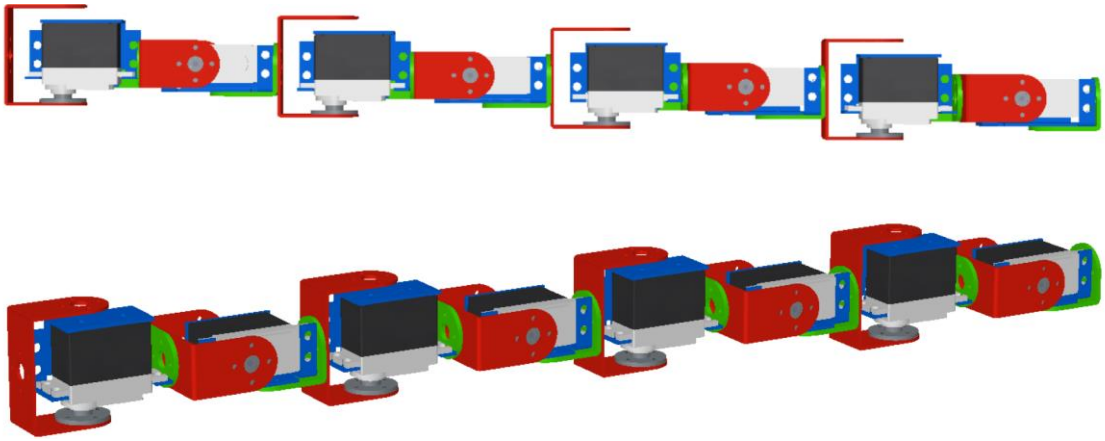


Figura 3.6: Ensamblaje en CAD de la configuración combinada o alternada *(Solidworks)*

La disposición de los módulos e identificación de componentes de ambas configuraciones, pueden observarse con mayor detalle en los planos de conjunto que se adjuntan al final del presente documento. Las configuraciones vertical y horizontal se encuentran en el Plano de Conjunto P02, mientras que la configuración combinada se puede encontrar el Plano de Conjunto P03.

3.5. NOMENCLATURA Y TERMINOLOGÍA

Para facilitar la comprensión de la terminología utilizada, se unifica la nomenclatura utilizada en el resto de la memoria.

En primer lugar, al conjunto de un servomotor y tres *brackets*, uno de cada tipo, se denomina módulo. Cada módulo se numera del 1 al 8 en función de la posición que ocupa, siendo el 1 el módulo que coincidiría con la cabeza del animal real, y 8 el último módulo. Al conjunto de los 8 módulos se le llama prototipo.

Además, a cada componente se le nombra por un conjunto de tres caracteres. El primer carácter es una letra que indica a qué tipo de componente es. Se representa con una "B", en caso de los *brackets*, o una S, en caso de servomotor.

El segundo carácter es exclusivo de los *brackets* y es una letra que indica a qué tipo de *bracket* se está refiriendo. Su identificación viene marcada por la posición en la que se encuentran dentro del módulo. De esta forma, existen tres posibilidades: "A" para la posición anterior, "M" para la posición media y "P" para la posición posterior.

Por último, el tercer carácter es un número que hace referencia al módulo que se está refiriendo.

Así pues, si se habla del "BM-3" se está haciendo referencia al *bracket* (B) ASB-04 (M) del módulo 3 del prototipo. De la misma forma, si se refiere al "S-7", se está aludiendo al servomotor (S) del módulo 7 del prototipo.

La nomenclatura de los componentes queda resumida en la Tabla 3.3 que se muestra a continuación.

Tabla 3.3: Resumen de la nomenclatura empleada para los componentes mecánicos

Primer carácter		**Segundo carácter**			**Tercer carácter**	
Identific.	**Referencia**	**Identific.**	**Referencia**	**Pieza Real**	**Identific.**	**Referencia**
B	Bracket	A	Anterior	ASB-04	1	Módulo 1
					2	Módulo 2
		M	Medio	ASB-09	3	Módulo 3
					4	Módulo 4
		P	Posterior	ASB-06	5	Módulo 5
					6	Módulo 6
S	Servomotor	Servomotor			7	Módulo 7
					8	Módulo 8

CAPÍTULO 4. SIMULACIÓN DEL MOVIMIENTO

La locomoción ápoda se basa en la generación de ondas sinusoidales para conseguir el movimiento. El avance se debe a la propagación de las ondas transversales a lo largo del cuerpo. La base técnica de dicho movimiento su desarrolla en la Sección 2.4.

Al tratarse de un movimiento sinusoidal es muy sensible a distintos parámetros, como los parámetros de cada onda (amplitud, frecuencia, etc) o el número de ondas distintas que se realizan. Cambios en estas variables generan grandes diferencias en el resultado del movimiento.

En el presente capítulo se realiza un estudio y simulación de diferentes configuraciones del prototipo con el fin de obtener una configuración óptima que permita el movimiento. Así pues, tras su lectura, se concluye cual es la configuración óptima en la que desarrollar los posteriores pasos.

Queda fuera del alcance de este capítulo el ajuste de los parámetros de la onda sinusoidal. La elección de estos parámetros se estudia en el CAPÍTULO 5, con la realización de un sistema de control del prototipo.

4.1. CONFIGURACIÓN PREVIA

4.1.1. Introducción

Con el fin de encontrar la configuración óptima para realizar el movimiento del prototipo, se analizan distintas configuraciones.

La diferencia entre las configuraciones estudiadas reside principalmente en la orientación de cada uno de los módulos del prototipo. Al presentar orientaciones distintas, la propagación de la onda sinusoidal genera diferentes comportamientos y puntos de apoyo -en los que basar el movimiento serpentino- en función del caso de estudio.

Además, al variar la orientación de los módulos, se permite la incorporación de distintas ondas independientes en el mismo movimiento.

Las configuraciones estudiadas se resumen en la Tabla 4.1. En cuanto a la unidad angular -grados-, se toma como 0° la situación en la que el plano que pasa en la dirección longitudinal del servomotor es paralelo al suelo.

Tabla 4.1: Resumen de las configuraciones empleadas en los distintos casos de estudio

Configuración	Ondas [ud.]	Módulos pares [°]	Módulos impares [°]
Horizontal	1	0	0
Vertical	1	90	90
Combinada	2	90	0

Por otro lado, las simulaciones de los distintos casos se realizan mediante el software *CoppeliaSim.* Debido a que la base del movimiento reside en la interacción de la onda sinusoidal con el suelo o los puntos de apoyo, la simulación debe de tener un comportamiento dinámico[12] y reactivo[13] -*'respondable'* en la terminología inglesa usada por el software *CoppeliaSim*- para poder simular el movimiento de forma adecuada.

A continuación, en la Sección 4.1.2 puede encontrarse la configuración del entorno de simulación para reproducir de forma precisa el movimiento ápodo.

[12] Se refiere a cualquier modelo que se vea afectado por fuerzas, colisiones entre distintos objetos y restricciones de movimientos [48].

[13] Se refiere a cualquier modelo que reaccione frente a otro objeto reactivo, cuando se produce una colisión [48].

4.1.2. Configuración del entorno de simulación

Con el fin de que la simulación sea lo más realista posible, se debe realizar una configuración del entorno de simulación, así como de las propiedades del prototipo diseñado. Dicha configuración es igual para cualquiera de las configuraciones descritas en la Tabla 4.1.

Para empezar a construir el entorno de simulación, se importa el diseño realizado en *Solidworks*. Los archivos *.STL* se importan con un escalado de 0.001, ya que el programa *CoppeliaSim* trabaja en metros, mientras que *Solidworks* en milímetros.

El siguiente paso consiste en el desarrollo del modelo dinámico del prototipo. Para ello, se crean objetos sencillos asociados a cada uno de los componentes importados. La forma de los objetos creados para el modelo dinámico se aproxima al componente que representa mediante formas geométricas puras.

Se distinguen las siguientes formas geométricas: plano, disco, cuboide, esfera y cilindro; los cuales, siguiendo la nomenclatura utilizada por *CoppeliaSim*, son denominadas: *plane, disc, cuboid, sphere* y *cylinder* respectivamente. Dichas piezas son las responsables de describir el comportamiento dinámico y reactivo del prototipo simulado.

Seguidamente, se unen las distintas piezas puras que no se quiere que interaccionen entre ellas. De esta forma, las posibles colisiones internas dentro de cada conjunto no son consideradas, sino solo se consideran las colisiones a nivel de grupo. Cada grupo corresponde a un módulo del prototipo.

De forma análoga, se agrupan los componentes importados de *Solidworks* de cada módulo para formar el modelo cinemático[14]. Por consiguiente, se dispone de 16 grupos: 8 que representan el modelo dinámico de cada módulo y 8 que representan el modelo cinemático.

A continuación, se muestra en la Figura 4.1 una comparación de un módulo y su grupo de objetos dinámicos puros asociados. En ella se observa que las piezas reales del modelo cinemático (imagen de la izquierda) se aproximan a prismas y cilindros para formar el modelo dinámico (imagen de la derecha).

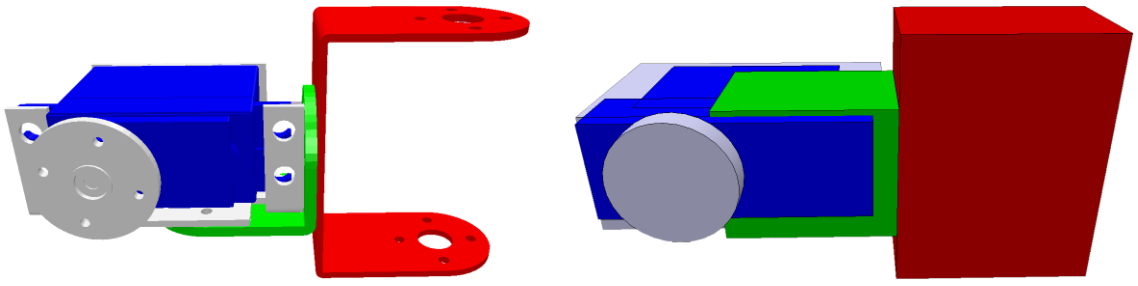


Figura 4.1: Configuración cinemática (izquierda) y la dinámica (derecha) de un módulo *(CoppeliaSim)*

[14] Se refiere a cualquier modelo que no se vea afectado por fuerzas externas. Su movimiento solo se da mediante forma manual o código.

Una vez definido el modelo dinámico, a continuación, se deshabilita la opción "*Local respondable mask*" -opción que contempla y reproduce la colisión entre los componentes-, puesto que los elementos que lo conforman son independientes.

Por otro lado, se modifica el material a uno con un elevado coeficiente de fricción, en concreto a la opción "*highFrictionMaterial*". Con dicha configuración los parámetros *"Friction (only Bullet V2.78)"*, *"Friction (after Bullet V2.78)"*[15] y la *"Friction"* de las propiedades de las ecuaciones diferenciales ordinarias -ODE-, se incrementan de **0.71** que se establece para un material por defecto, a un valor de **1.00**, tal y como se muestra en la siguiente Figura 4.2.

Physics Engines Properties - Material

Property	Value
Apply predefined settings:	highFrictionMaterial
▼ Bullet properties	
Friction (only Bullet V2.78)	1.0000e+00
Friction (after Bullet V2.78)	1.0000e+00
Restitution	0.0000e+00
Linear damping	0.0000e+00
Angular damping	0.0000e+00
Sticky contact (only Bullet V2.78)	☐ False
Auto-shrink convex mesh	☐ False
Custom collision margin	☐ False
Custom collision margin factor	1.0000e-01
▼ ODE properties	
Friction	1.0000e+00
Maximum contacts	64
Soft ERP	2.0000e-01
Soft CFM	0.0000e+00
Linear damping	0.0000e+00
Angular damping	0.0000e+00

Figura 4.2: Propiedades físicas del material empleado *(CoppeliaSim)*

De esta forma, la simulación actúa con cierta fricción, ya que en caso de ser lo contrario deslizaría más de lo deseable y no se podría representar el movimiento serpentino adecuadamente.

[15] Se refiere al valor de fricción utilizado en la simulación, donde la fricción combinada entre dos objetos tendrá un valor igual a valor1*valor2. Esto no se corresponde con el coeficiente de fricción real.

La unión de los diferentes módulos se realiza mediante articulaciones. El eje de la articulación se posiciona de forma que coincida con el eje del servomotor, para poder transmitir el movimiento.

Para el correcto funcionamiento de las articulaciones se debe configurar la articulación en modo "Par/Fuerza". Además, en las propiedades dinámicas de la articulación, se debe habilitar el motor y el control en bucle cerrado, puesto que se quiere controlar los servos por posición. (Véase Figura 4.3)

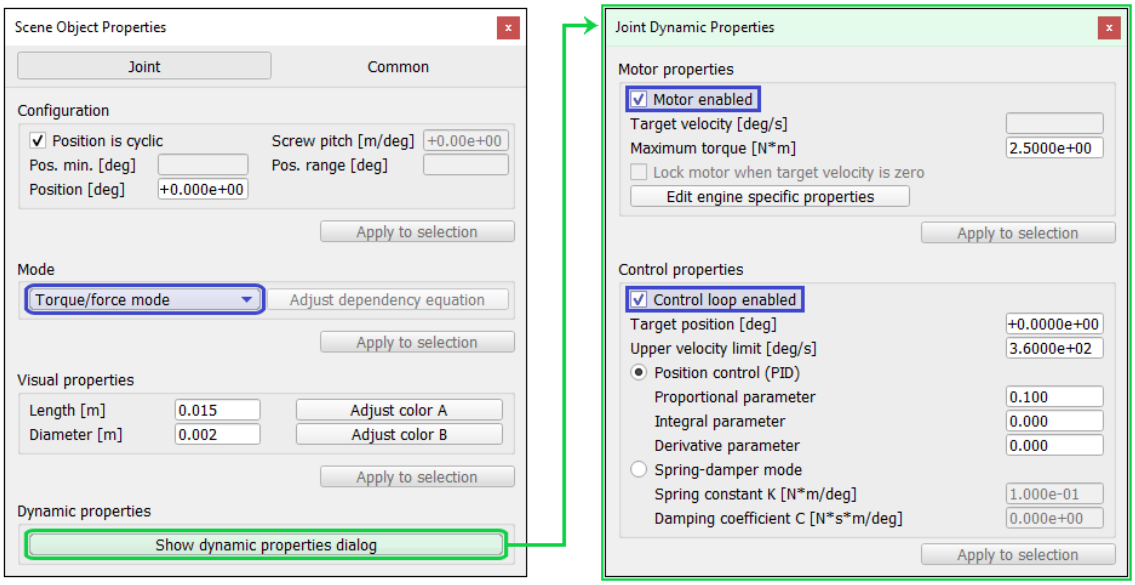


Figura 4.3: Propiedades generales (izquierda) y dinámicas (derecha) de la articulación *(CoppeliaSim)*

4.1.3. Relación de jerarquía del prototipo

La relación de jerarquía determina la relación principal-secundario entre las entidades del prototipo. Dicho de otra manera, decide que componentes se ven afectados y por cuales se ven afectados. La jerarquía desarrollada se muestra en la Figura 4.4 y se estructura así para que cada módulo arrastre al siguiente módulo al rededor del eje de la articulación.

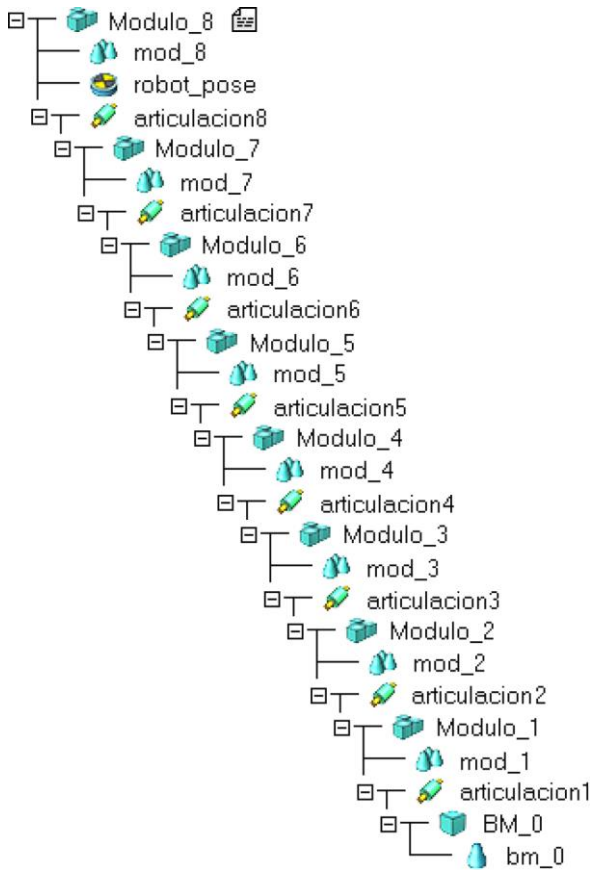


Figura 4.4: Relación jerárquica del prototipo *(CoppeliaSim)*

4.1.4. Escritura de código: scripts embebidos

Tal y como se menciona en la Sección 1.3, el software *CoppeliaSim* soporta hasta seis formas diferentes de programación de código. A lo largo del presente Trabajo Final de Máster, se utiliza la opción de los *scripts* -programados en *Lua*- debido a su sencillez y flexibilidad. Se utiliza un *main script* para controlar el bucle de simulación principal y un *child script* -más concretamente, un *non-threated script*- para controlar el modelo implementado.

El código implementado en cada *child script* varía en función del caso de estudio que se analice, por lo que se desarrolla su implementación en la sección correspondiente al estudio de cada caso de forma individual.

4.2. CONFIGURACIÓN HORIZONTAL

El primer caso de estudio analizado consiste en la configuración horizontal. En ella, todos los módulos tienen la misma orientación, 0°. Esta configuración genera una onda sinusoidal en el plano XZ, y presenta un único grado de libertad. El diseño del prototipo realizado es el mostrado en la Figura 4.5. La configuración del entorno de simulación, así como la jerarquía del prototipo, es la descrita en la Sección 4.1.

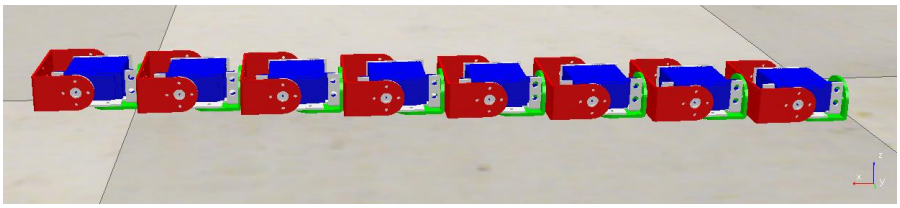


Figura 4.5: Caso de estudio 1: Configuración horizontal *(CoppeliaSim)*

La programación del caso de estudio bajo la configuración horizontal se realiza en un *non-threated child script* asociado al módulo ubicado en la primera posición de la jerarquía del prototipo -módulo 8- y estructurado en dos funciones: `sysCall_init()`, donde se inicializan los parámetros, y `sysCall_actuation()`, donde se programa el código de funcionamiento del prototipo.

En la función de inicialización se obtienen los identificadores de las articulaciones mediante la función `simGetObjectHandle()`[16]. Así, los cálculos posteriores que se realizan en el algoritmo de movimiento son referidos a cada uno de los módulos mediante su identificador.

Respecto a la función principal, en ella se realiza la programación del algoritmo de movimiento serpentino, basado en las ecuaciones desarrolladas en la Sección 2.4. El procedimiento consiste en la realización de dos bucles, el primero es el encargado de recorrer todas las articulaciones y calcular la posición angular y fijarla en el módulo correspondiente por medio de la función `sim.setJointTargetPosition()`[17].

El segundo bucle está inmerso en el bucle anterior y recorre de nuevo los módulos desde el inicio hasta el módulo que se encuentra la iteración del bucle anterior. Este bucle se encarga de calcular la suma de las posiciones cartesianas de los módulos de cada iteración.

[16] Función regular de API que devuelve el identificador de un objeto basado en su nombre [49]. El término API hace referencia a la interfaz de programación de aplicaciones.

[17] Función regular de API que establece la posición objetivo de una articulación si está en modo par/fuerza [50].

A modo de ejemplo, si el primer bucle se encuentra recorriendo el módulo tres, el segundo bucle calcula, en la primera iteración la posición del módulo uno; en la segunda la posición del módulo dos más la posición del módulo uno; y en la tercera y última calcula la posición del módulo tres más la suma de la posición del módulo uno y dos. Esta última posición es la que se guarda, obteniéndose así la posición absoluta de cada módulo respecto al punto inicial de referencia.

El pseudocódigo que sigue está función es el siguiente. Para mayor detalle, se adjunta el código completo en el Anexo B1.1.

- Inicialización variables,
- Entrada a un bucle (i) que se repite para cada articulación,
    - Inicialización de las variables de posición cartesianas a 0,
    - Entrada a un bucle (k) que se repite hasta la articulación i,
        - Cálculo de la posición absoluta de cada módulo,
    - Cambio de variable para mantener la posición del módulo anterior,
    - Cálculo de la posición angular del módulo,
    - Establecimiento de la posición angular en el módulo correspondiente,
- Aumento del paso de simulación.

El resultado de la presente simulación es un movimiento en una única dirección, la dirección de avance. Para comprobar el movimiento se fijan dos referencias con una separación de 1 m entre ellas y se observa el avance -en sentido de derecha a izquierda- del prototipo robótico en función del tiempo. El resultado se muestra en la Figura 4.6.

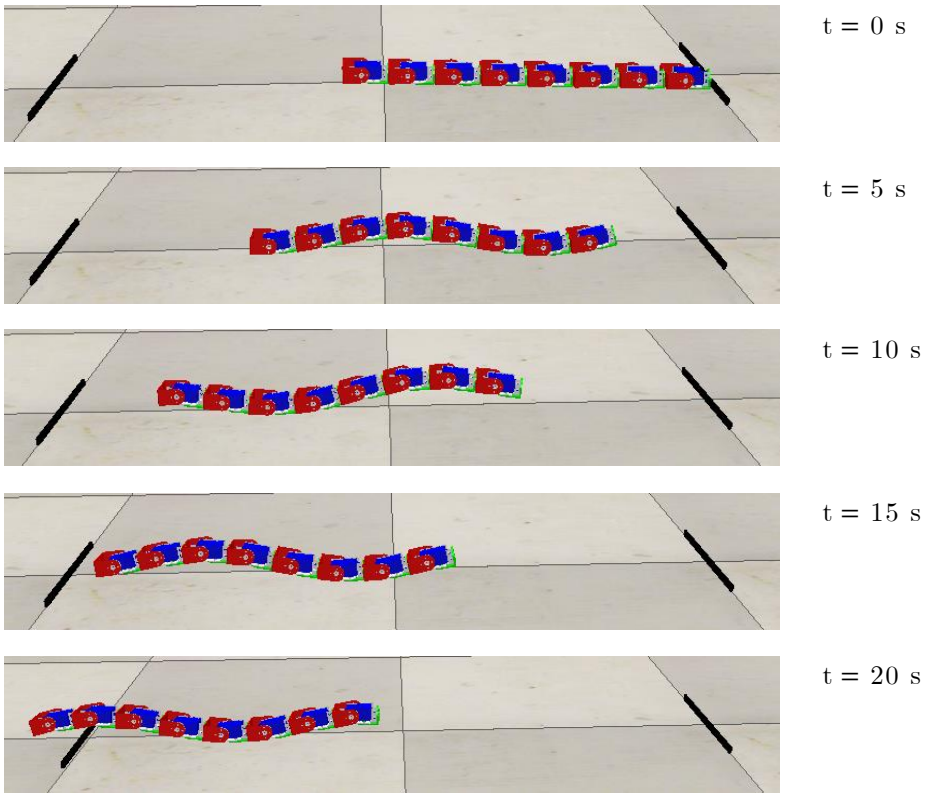


Figura 4.6: Simulación de avance para la configuración horizontal *(CoppeliaSim)*

4.3. CONFIGURACIÓN VERTICAL

El segundo caso de estudio consiste en la configuración vertical. En este caso todos los módulos están orientados 90° respecto a la superficie del suelo, generando una onda sinusoidal en el plano XY.

El diseño del prototipo utilizado para la simulación se muestra en la Figura 4.7, que, al igual que el caso anterior, presenta un único grado de libertad.

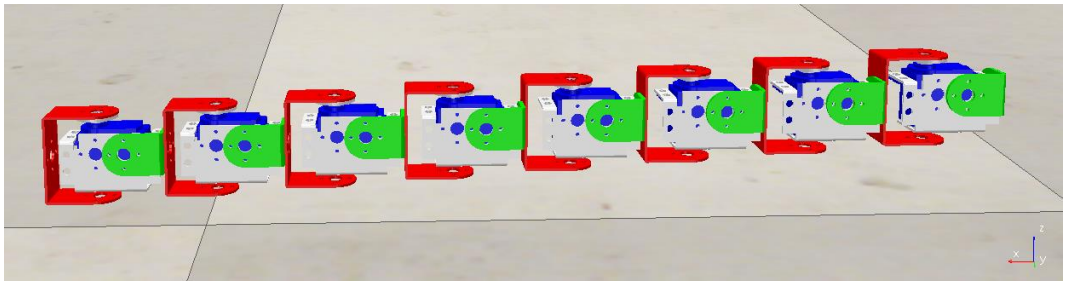


Figura 4.7: Caso de estudio 2: Configuración vertical *(CoppeliaSim)*

La configuración de esta opción es similar a la del caso anterior descrito en la Sección 4.2. La configuración del entorno de simulación y la programación del algoritmo es idéntica al caso horizontal. Sin embargo, al cambiar la orientación de los módulos, se generan puntos de apoyo y fricción distintos, por lo que el movimiento que se genera es completamente distinto al caso anterior.

El resultado de la simulación de este caso muestra un movimiento en una única dirección, pero en este caso la transversal. La onda sinusoidal creada en esta configuración es paralela al suelo, por lo que no se generan puntos de apoyo con los que poder realizar el avance.

Sin embargo, con esta configuración, al tener como dirección de movimiento la transversal, permite el giro del prototipo. Se muestra el resultado de la simulación en la Figura 4.8. Mencionar que el movimiento lateral que se produce es debido a una deriva por la fricción configurada, muy común en este tipo de programas de simulación.

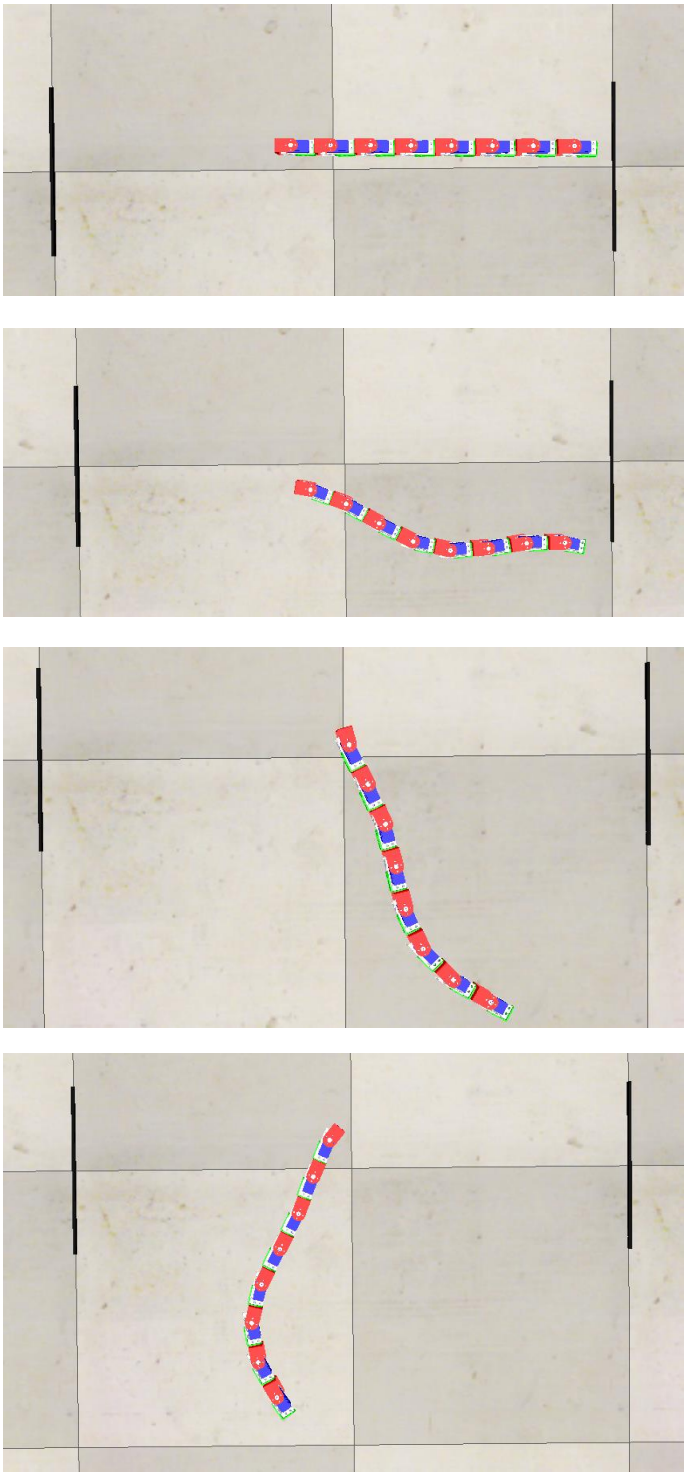


Figura 4.8: Simulación del giro para la configuración vertical *(CoppeliaSim)*

4.4. CONFIGURACIÓN COMBINADA

El último de los casos estudiados consiste en una combinación de los dos casos anteriores. Con el primer caso de estudio -orientación horizontal- se puede realizar el movimiento de avance, mientras que con el segundo caso de estudio – configuración vertical- se realiza el cambio de dirección. Por ello, se propone la realización de un diseño conjunto de forma que se pueda, con el mismo diseño, realizar el avance y el giro.

El diseño propuesto consiste en alternar un servomotor orientado a 0° seguido de otro a 90°. De esta forma, se le añade un grado de libertad al prototipo. El resultado del diseño global (mostrado en la Figura 4.9) son cuatro servomotores orientados a 0° y cuatro servomotores a 90°.

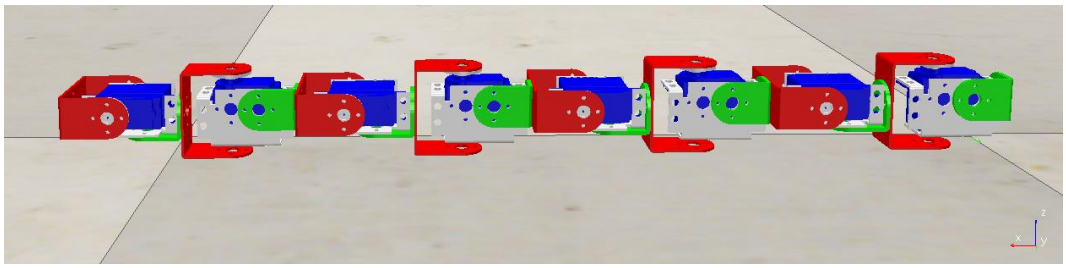


Figura 4.9: Caso de estudio 3: Configuración combinada *(CoppeliaSim)*

Cada grupo de cuatro servomotores es responsable de representar una onda sinusoidal distinta. Por ello, durante el movimiento se sobreponen dos ondas sinusoidales, una en el plano XY, encargada de los giros, y otra en el plano XZ, encargada del avance. De forma gráfica, puede verse el resultado de la acción de ambas ondas en la Figura 4.10. Cada onda sinusoidal por separado sigue el mismo razonamiento descrito en la Sección 2.4.

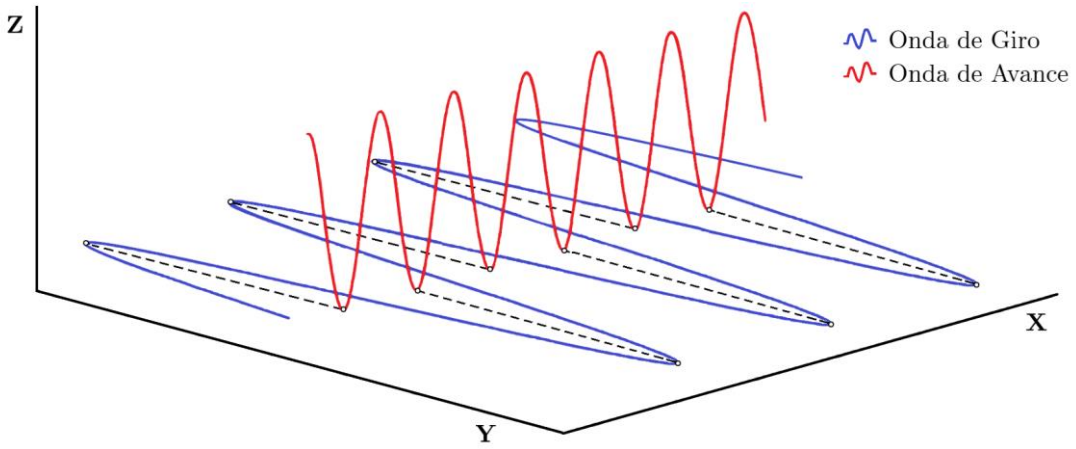


Figura 4.10: Curva serpentina en los planos XZ (avance) y XY (giro) para la configuración combinada

La programación del caso de estudio actual sigue el mismo esquema que los casos anteriores, con la diferencia de que se realiza todos cómputos por duplicado al tener dos ondas que programar. En la función de inicialización se obtienen los identificadores de las articulaciones, diferenciando en distintos vectores los módulos que controlan el movimiento de avance de los que controlan el cambio direccional.

En cuanto a la función principal, en ella se realiza la programación de los dos movimientos senoidales. En un primer lugar se inicializan por separado los parámetros referentes a la onda de avance y a la onda de giro. A continuación, se realizan dos bucles iterativos. El primero se encarga de recorrer todas las articulaciones, calculando las posiciones angulares y fijándolas por medio del comando `sim.getObjectOrientation`. El segundo bucle se encarga de calcular las posiciones cartesianas de cada módulo.

El funcionamiento de este algoritmo es el mismo que en la Sección 4.2, con la particularidad de que ahora en cada bucle se calcula un módulo horizontal y un módulo vertical, guardando los resultados en vectores distintos. Así, cada bucle realiza la mitad de iteraciones, pero el doble de trabajo, pues en todo momento se tienen las variables duplicadas y diferenciadas para una onda u otra. En el Anexo B1.2 se adjunta el código detallado de este caso de estudio.

Los resultados de la simulación muestran como el prototipo es capaz de avanzar hacia delante, tal y como se muestra en la Figura 4.11.

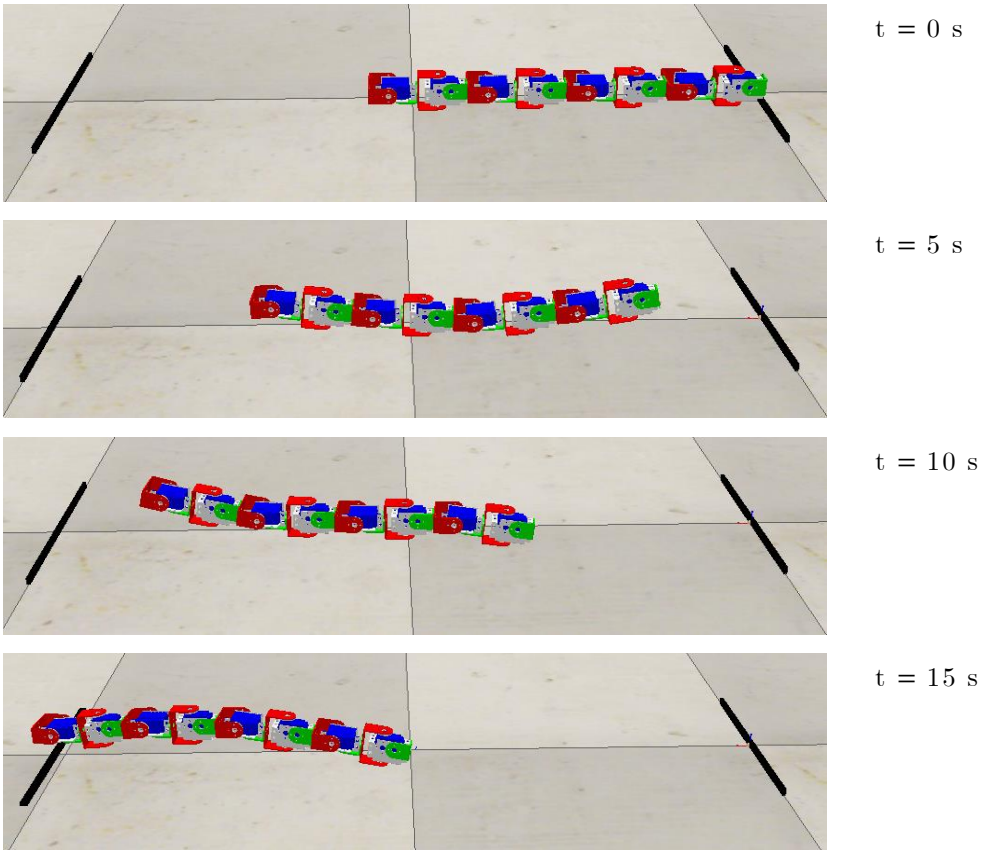


Figura 4.11: Simulación de avance para la configuración combinada *(CoppeliaSim)*

Igualmente, tal y como se observa en la Figura 4.12, el demostrador es capaz de llevar a cabo cambios de dirección satisfactoriamente.

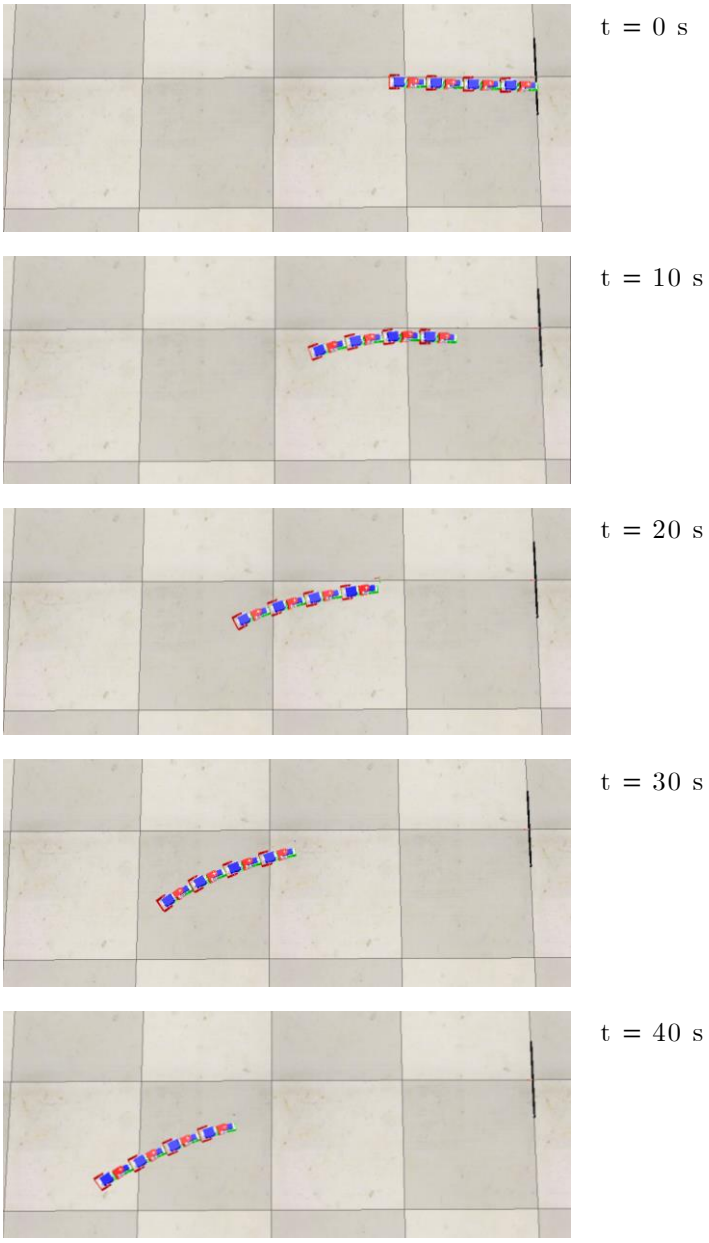


Figura 4.12: Simulación de giro para la configuración combinada *(CoppeliaSim)*

4.5. CONCLUSIONES

Del análisis de las tres configuraciones anteriores, se extraen las conclusiones que se exponen a continuación.

En primer lugar, en lo referente a la libertad del movimiento, tanto la primera configuración como la segunda solo cuentan con un grado de libertad, por lo que solo se permite el movimiento en una dirección. En el primero de los casos se observa que se consigue simular exitosamente el avance del prototipo, de una forma eficiente y similar al movimiento serpentino.

En el segundo de los casos, no se consigue ningún avance, sino una rotación sobre sí mismo. Este caso carece de interés por sí solo ya que no ofrece ninguna utilidad. Sin embargo, ofrece información suficiente para la implementación del giro del prototipo.

La información extraída de los dos casos anteriores concluye en la implementación del caso combinado. Esta opción permite el movimiento en dos direcciones. Esta configuración presenta 2 grados de libertad y se comprueba con la simulación que puede tanto avanzar como realizar cambios de dirección.

Por otro lado, en cuanto a la precisión del movimiento, las dos primeras opciones cuentan con ocho módulos para representar la onda sinusoidal. Como cuanto mayor es el número de módulos, mayor es la precisión y realismo del movimiento, se comprueba en las simulaciones el parecido del movimiento con el movimiento que realiza una serpiente real. Se obtiene así un movimiento serpentino con una precisión y realismo considerablemente buenos en comparación con el movimiento que imita.

En cuanto a la tercera opción, muestra una precisión menor y un comportamiento menos realista ya que, al realizar dos ondas sinusoidales simultáneamente, realmente cada onda solo es representada por cuatro módulos en vez de por los ocho módulos de los casos anteriores.

Sin embargo, como se desarrolla en el CAPÍTULO 3 cuando se realiza la elección del número de módulos, el número mínimo de módulos necesarios para representar una onda sinusoidal de forma correcta es 4. Por tanto, a pesar de presentar un comportamiento menos realista que las dos opciones anteriores, muestra un comportamiento suficientemente preciso.

Por último, parámetros como la velocidad del movimiento no son decisivos para la selección del diseño óptimo, pues pueden variarse en cualquiera de las tres opciones variando los parámetros internos de la onda sinusoidal. Este punto se estudia con mayor detalle en el CAPÍTULO 5.

A modo de resumen se realiza una tabla comparativa (Tabla 4.2) con los aspectos principales de cada caso de estudio.

Tabla 4.2: Tabla comparativa de los casos de estudio

Parámetro de estudio	**Opción Horizontal**	**Opción Vertical**	**Opción Combinada**
Dirección del movimiento	Longitudinal	Transversal	Ambos
Grados de libertad de cada módulo	1	1	1
Grados de libertad del prototipo	1	1	2
Número de ondas totales	1	1	2
Número de módulos por onda	8	8	4
Precisión del movimiento	Buena	Buena	Media

Por ello, se escoge como configuración óptima la opción combinada, ya que, a pesar de ser la que presenta el movimiento menos realista, este sigue siendo lo suficientemente preciso para tenerse en cuenta.

Además, este aspecto tiene poca importancia en comparación con la posibilidad de realizar el movimiento en dos direcciones, criterio fundamental en la elección de la configuración óptima. La posibilidad de realizar el movimiento en dos direcciones -avance y giro- de forma controlada proporciona numerosas ventajas en las aplicaciones que se mencionan en la Sección 2.1.3, que, al final, es el objetivo último del presente Trabajo Final de Máster.

Por último, mencionar que se han considerado otras configuraciones con orientaciones distintas a 0 o 90°, sin embargo, debido a la complejidad de su programación, así como el tiempo limitado que se dispone para la realización del presente Trabajo Final de Máster, quedan fuera del alcance de este proyecto.

CAPÍTULO 5.
DISEÑO DEL SISTEMA DE CONTROL

5.1. ARQUITECTURA DE CONTROL

El control, tanto a nivel simulación como sobre el prototipo real, es uno de los puntos principales en la realización y diseño de un robot. El control permite conocer la situación del robot, entendiendo "situación" como el intercambio de información e interacción del robot con el entorno.

Además, gracias al sistema de control, se puede conseguir que el robot cumpla ciertas especificaciones o movimientos a petición del usuario.

Hoy en día es posible controlar todos los componentes de un robot desde un ordenador. De esta forma, las ordenes se envían a través del ordenador, y son efectuadas por los sistemas electrónicos que conforman el robot.

A lo largo de la presente sección se describe el diseño e implementación de la arquitectura de control del robot ápodo realizado.

.

5.1.1. Parámetros de control

El sistema de control está formado por los sistemas electrónicos complejos, que controlan las acciones del robot, y por un ordenador, por medio del cual se introducen las órdenes. Este ordenador manda las ordenes al microcontrolador, que es el encargado de enviar cada orden al servomotor correspondiente para realizar el movimiento.

La finalidad del control realizado es conseguir que el prototipo realice el movimiento serpentino con una determinada velocidad lineal, v, y angular, w -el cálculo de ambas velocidades se aborda en la Sección 5.2.2 del presente capítulo-. Ambas velocidades determinadas por el usuario.

Por otro lado, como se describe en la Sección 4.4, el movimiento a controlar se compone de dos ondas sinusoidales, una vertical y otra horizontal. Cada onda, como se explica en la Sección 2.4, se caracteriza por tres parámetros: a, referente a la amplitud de la onda; b, relacionado con el número de ondas; y c, que define la curvatura de la trayectoria sinusoidal.

Como se cuenta con dos ondas distintas, cada una de las ondas cuenta con los tres parámetros anteriores, sumando un total de 6 parámetros a controlar. Para diferenciar cada parámetro a qué onda se refiere, se utiliza el subíndice 'h' para la onda horizontal y 'v' para la onda vertical. De esta forma, el parámetro a_h hace referencia a la amplitud de la onda horizontal y a_v a la amplitud de la onda vertical.

Asimismo, como se comprueba en la Sección 2.4, el parámetro c hace referencia a la curvatura de la trayectoria serpentina. Este parámetro provoca que el robot curve los dos extremos de su cuerpo, formando una circunferencia como se muestra en la Figura 5.1. Por ello, como esta posición no es de interés para el estudio que se está realizando, tanto c_h como c_v se fijan a 0 para cualquier configuración.

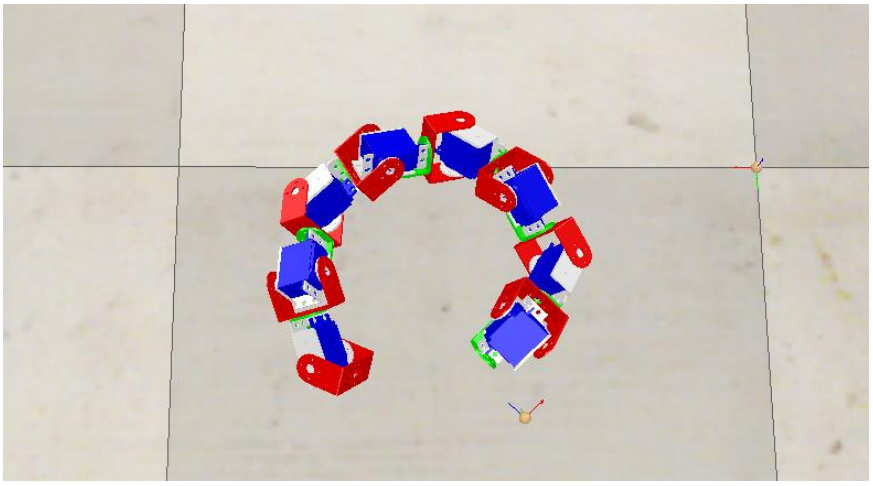


Figura 5.1: Curvatura del prototipo al modificar el parámetro c *(CoppeliaSim)*

5.1.2. Estructura de control

Para realizar el control de la velocidad angular y lineal del robot, se implementa un sistema de control por medio de la cual el usuario especifica ciertos parámetros y el sistema de control calcula el valor necesario del resto de variables para alcanzar las velocidades deseadas.

Como se menciona en el apartado anterior, las velocidades lineales y angulares serán determinadas por el usuario. Para conseguir dichas velocidades, se tienen 4 parámetros modificables: a_h, a_v, b_h y b_v.

De estas 4 variables, se define que los parámetros referentes a la amplitud de onda (a_h y a_v), sean determinados también por el usuario. Se define así debido a que el parámetro de amplitud tiene que ver con la separación del prototipo respecto el suelo y su elección por el usuario puede ser de interés para algunas aplicaciones.

De esta forma, el usuario determina las velocidades lineales y angulares y los parámetros de amplitud de la onda, y, por medio del sistema de control, se realizan los cálculos para determinar los parámetros b_h y b_v que consiguen las velocidades determinadas.

La estructura de la red de control queda como la mostrada en la siguiente Figura 5.2.

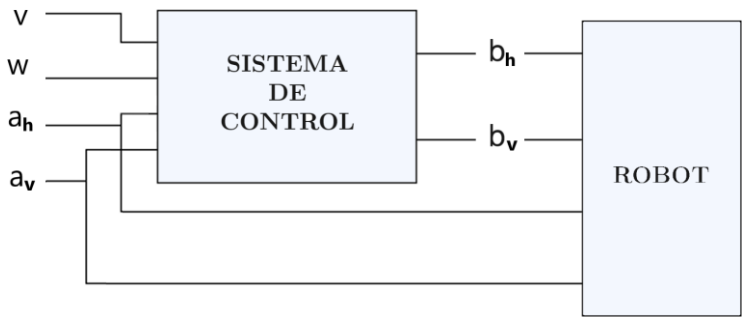


Figura 5.2: Estructura del sistema de control

Para la realización del sistema de control se desarrolla una red neuronal, ya que esta tecnología es capaz de aprender por medio de un entrenamiento inicial. De esta forma, no es necesario elaborar modelos para cada tipo de movimiento, sino que el propio prototipo aprende de forma autónoma las distintas posibilidades. Además, cuenta con una fácil inserción dentro de la tecnología existente.

5.2. IMPLEMENTACIÓN DE LA RED NEURONAL

5.2.1. Concepto de red neuronal

Las neuronas son células del sistema nervioso que reciben los estímulos provenientes del ambiente y los procesa convirtiéndolos en información que transmitir a otras células [51].

Las redes neuronales emulan el modo de funcionamiento en que las neuronas -de ahí su nombre- procesan la información. Se trata de modelos simples de neuronas -unidad básica de información- organizadas en capas.

La primera capa, la capa de entrada, representa la información entrante del ambiente. Esta información se transmite de neurona a neurona en una o varias capas ocultas, donde se procesa la información. Por último, hay una capa de salida, con una o varias neuronas, que representa los campos de destino.

Las neuronas están conectadas por medio de enlaces que tienen ponderaciones específicas. La información se propaga de capa en capa de neuronas en base a las ponderaciones asignadas a cada enlace. Dichas ponderaciones son aleatorias en un principio y, conforme la red aprende a través del entrenamiento, se van ajustando y precisando.

El entrenamiento consiste en introducirle una base de datos con resultados conocidos. La red genera una predicción para cada registro y lo compara con el resultado real, ajustando las ponderaciones en cada iteración. A cada iteración se le llama época, y es el número de veces que se ejecutan los algoritmos de *forwardpropagation*[18] y *backpropagation*[19].

Este proceso se realiza muchas veces hasta que converge en una red muy precisa en la replicación de resultados conocidos. Una vez entrenada, la red se puede aplicar a casos futuros en los que se desconoce el resultado [54].

[18] Se refiere al proceso de entrenamiento de la red neuronal en el cual los datos de entrada que se introducen a cada capa oculta de la red, se procesan según la función de activación, y se transmiten a la capa siguiente en dirección hacia delante [52].

[19] Se refiere al proceso de entrenamiento de la red neuronal en el cual los datos de entrada se transmiten en dirección hacia atrás para ajustar y corregir los pesos y alcanzar la función de pérdidas minimizada [53].

5.2.2. Adquisición de datos

Como se menciona en el apartado anterior, para entrenar una red neuronal es necesario tener una base de datos con resultados conocidos. Para conseguir la base de datos se realizan simulaciones en *CoppeliaSim* con la configuración seleccionada en la Sección 4.4.

Las simulaciones consisten en especificar los parámetros a_h, a_v, b_h y b_v de forma que se simula durante 10 segundos el movimiento serpentino del prototipo para cada una de las combinaciones probadas y se obtiene la posición (x_f e y_f) y orientación final (σ) tras el movimiento.

Para calcular la posición y orientación final se ha incorporado dos '*dummies*'[20] al entorno de simulación. Uno de ellos se sitúa en el inicio del movimiento, usándose como punto de referencia. El segundo *dummy* se sitúa sobre el prototipo, para saber dónde se encuentra en cada momento. De esta forma se puede saber la posición y orientación final del prototipo respecto al punto de inicio del movimiento.

Con la posición y la orientación finales se calculan la velocidad lineal y velocidad angular de la siguiente forma. El movimiento de la serpiente tiende a realizar un arco de circunferencia, A_{crc}, tal y como se muestra en azul en la Figura 5.3.

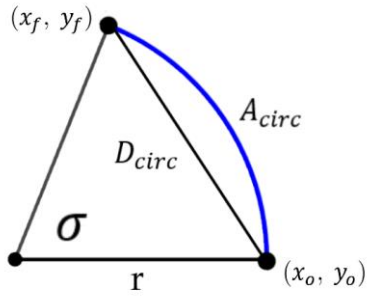


Figura 5.3: Geometría que describe la trayectoria serpentina

Sin embargo, con la posición final se obtiene la línea recta que une el punto inicial con el final, es decir, la cuerda de la circunferencia, D_{circ}, de la Figura 5.3. Dicha longitud se calcula con la posición final en los ejes X e Y del prototipo (x_f y y_f respectivamente), tal y como se muestra en la Ecuación 13.

$$D_{circ} = \sqrt{x_f^2 + y_f^2} \qquad \text{[Ec. 13]}$$

[20] Objeto ficticio simple (punto) con orientación [55].

Como se conoce por geometría la ecuación que relaciona la cuerda de una circunferencia con su radio (Ecuación 14), se calcula el radio de la trayectoria serpentina [56].

$$r = \frac{D_{circ}}{2 \cdot \sin\left(\frac{\sigma}{2}\right)} \qquad \text{[Ec. 14]}$$

El arco de la circunferencia es el radio de dicha circunferencia por el ángulo recorrido, que coincide con el σ obtenido de *CoppeliaSim* (Véase Ecuación 15).

$$A_{circ} = r \cdot \sigma \qquad \text{[Ec. 15]}$$

Una vez obtenido el arco de circunferencia, la velocidad lineal y angular del prototipo quedan definidas como se muestra en las Ecuaciones 16 y 17.

$$v = \frac{A_{circ}}{t} \qquad \text{[Ec. 16]}$$

$$w = \frac{\sigma}{t} \qquad \text{[Ec. 17]}$$

Por otro lado, para conseguir una base de datos lo suficientemente grande y precisa para servir de entrenamiento a la red neuronal, se automatiza el proceso mediante el software *Matlab R2019b*.

Matlab es el encargado de mandar y recoger la información. Para ello, se enlaza con *CoppeliaSim* mediante API Remota[21]. Con esta configuración, *Matlab* realiza un barrido de los distintos valores posibles para a_h, a_v, b_h y b_v envía dichos valores a *CoppeliaSim,* mediante la función API Remota `simxCallScriptFunction()`[22].

Una vez terminada la simulación de cada combinación, *CoppeliaSim* devuelve a *Matlab* el valor de la posición y orientación final, quien, por medio del desarrollo matemático realizado en esta sección, calcula la velocidad lineal y angular. Estos valores son almacenados en la base de datos junto a los valores de la combinación de parámetros que han generado dichas velocidades.

[21] La API remota forma parte del entorno de la API de CoppeliaSim. Permite la comunicación entre CoppeliaSim y una aplicación externa [57].

[22] Llama a una función de script desde un plugin, aplicación principal del cliente u otro script [58].

La base de datos resultante tiene un total de 2058 casos, con seis variables de estudio para cada uno de los casos. Los rangos estudiados para cada parámetro de entrada de la simulación se resumen en la Tabla 5.1.

Tabla 5.1: Rango de los parámetros del movimiento para la adquisición de la base de datos

Parámetro	Valor mínimo	Valor máximo	Variación
a_h	0.05	0.30	0.05
a_v	-0.15	0.15	0.05
b_h	0.00	30.00	5.00
b_v	0.00	30.00	5.00

Se adjunta en el Anexo B2.1 y Anexo B2.2 el código de *Matlab* y *CoppeliaSim*, respectivamente, realizado para la conexión de ambos programas y generación de la base de datos.

5.2.3. Realización de la red neuronal

Una vez obtenida la base de datos, se realiza la red neuronal. Esta red se va a realizar en Matlab, mediante la *toolbox 'Deep Learning Toolbox'*, que contiene las funciones necesarias para entrenar y programar una red neuronal.

Tras cargar la base de datos, es necesario realizar su normalización para que todos los datos estén comprendidos entre 0 y 1. Realizar la normalización es necesario debido a que, como se menciona en la Sección 5.2.1, la red neuronal multiplica la información por una determinada ponderación, y la compara con el resto de los datos. Para que la comparativa sea correcta es necesario que todas las variables se encuentren acotadas entre un mínimo y máximo conocido. Por ello, se acotan y normalizan todos los datos para estar comprendidos entre 0 y 1.

Para el entrenamiento de la red se utiliza el comando `train` de *Matlab*. Esta función entrena una red neuronal superficial, utilizando una única capa oculta [59].

Los argumentos de entrada de dicha función son: la red creada, la base de datos de entradas y la base de datos de las salidas. La sintaxis de la función se muestra en la Ecuación 18.

$$\texttt{[net,pr]=train(net,}\texttt{datos}_{\texttt{entrada}}\texttt{,}\texttt{datos}_{\texttt{salida}}\texttt{)} \qquad \text{[Ec. 18]}$$

La variable `net` contiene la red neuronal entrenada, mientras que la variable `pr` incluye la información y registro de entrenamiento.

La función de pérdida que utiliza el comando `train` es el mínimo error cuadrático. Este parámetro se guarda por defecto en la variable `net.performFcn` y sirve para comprobar la calidad de la red neuronal [60].

En cuanto al algoritmo de entrenamiento, el usado por defecto por la función es el algoritmo *Levenber-Marquardt*. Este algoritmo fue diseñado para acercarse a la velocidad de entrenamiento de segundo orden[23] sin tener que calcular la matriz hessiana.

La aproximación a dicha matriz hessiana es a la mostrada en la Ecuación 19. Para la aproximación, se tiene en cuenta que la función de rendimiento tiene la forma de una suma de cuadrados -función del error cuadrático mínimo-, por ello es posible realizar dicha aproximación.

$$H = J^T \cdot J \qquad \text{[Ec. 19]}$$

Y el gradiente por tanto queda como el mostrado en la Ecuación 20.

$$g = J^T \cdot e \qquad \text{[Ec. 20]}$$

Donde J es la matriz Jacobiana que contiene las primeras derivadas de los errores de la red neuronal respecto a los pesos y desviaciones, y e es un vector de errores de la red neuronal.

La matriz Jacobiana puede ser calculada mediante técnicas de *backpropagation*, mucho más sencillas que el cálculo de la matriz Hessiana. El algoritmo Levenberg-Marquardt usado para cada actualización es el mostrado en la Ecuación 21 [61].

$$LM_{epoch+1} = LM_{epoch} - [J^T \cdot J + \mu \cdot I]^{-1} \cdot J^T \cdot e \qquad \text{[Ec. 21]}$$

Cuando μ es 0, el algoritmo se convierte en el método de Newton[24], mientras que cuando μ adquiere un valor grande, este método se aproxima al método de descenso de gradiente[25] con un tamaño de paso pequeño.

El método de Newton es más rápido y preciso, por lo que el objetivo del método de entrenamiento es reducir el parámetro μ para ganar precisión y rapidez.

[23] Se refiere a los algoritmos conocidos como "de segundo orden" que hacen uso de la matriz de las segundas derivadas parciales, llamada matriz Hessiana, con el objetivo encontrar las mejores direcciones de variación de los parámetros

[24] El método de Newton es una resolución numérica que busca el cero de una función por medio de aproximaciones sucesivas respecto un valor inicial [62].

[25] El método de descenso de gradiente es un algoritmo de optimización que permite converger hacia el valor mínimo de una función mediante un proceso iterativo [63].

Previamente a utilizar dicha función, es necesario definir los parámetros básicos para crear la red que posteriormente se entrena.

En primer lugar, se fija el número de neuronas para la capa oculta. Este parámetro se ha ido variando hasta encontrar la solución óptima, fijándose finalmente en 800 neuronas.

Además, se fijan también el número de épocas. Este parámetro coincide con el número de repeticiones o iteraciones que realiza la red neuronal durante su entrenamiento antes de encontrar la solución óptima. Es importante que este parámetro sea suficientemente grande para que la red neuronal converja a una única solución.

La estructura de la red neuronal diseñada se puede observar en la Figura 5.4.

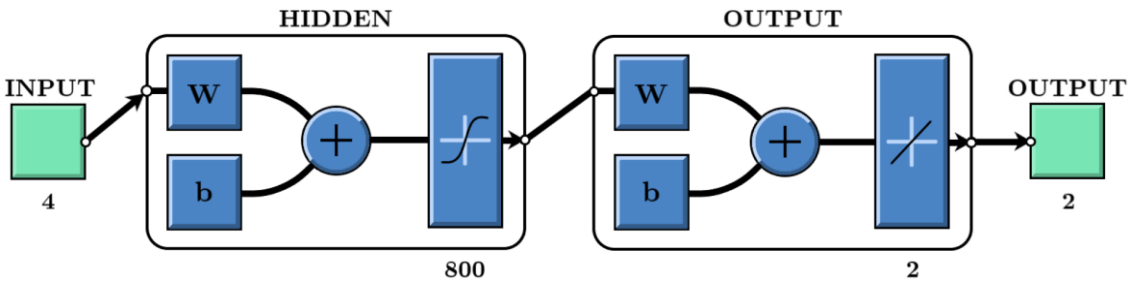


Figura 5.4: Estructura de la red neuronal implementada *(Matlab)*

En el Anexo B2.3 se adjunta el código de *Matlab* realizado para la programación de la red neuronal.

5.2.4. Resultados de la red neuronal

Tras el entrenamiento de la red, se observan los resultados descritos a continuación.

En primer lugar, para poder considerar que el entrenamiento de la red neuronal ha finalizado, es necesario que las predicciones converjan a un valor estable de ellas. Dicha convergencia se puede analizar con el error mínimo cuadrático de las predicciones.

En la Figura 5.5, se observa como el error converge a un valor estable cuando se supera la época 28000. Sin embargo, se deja un margen de 2000 épocas más para mayor seguridad y confianza de que se ha alcanzado la estabilidad, dándose así por finalizado el entrenamiento de la red neuronal. Además, cabe destacar que el error mínimo cuadrático estabiliza en un valor inferior a 10^{-3}, lo cual índica que la red neuronal realiza predicciones muy ajustadas a la realidad.

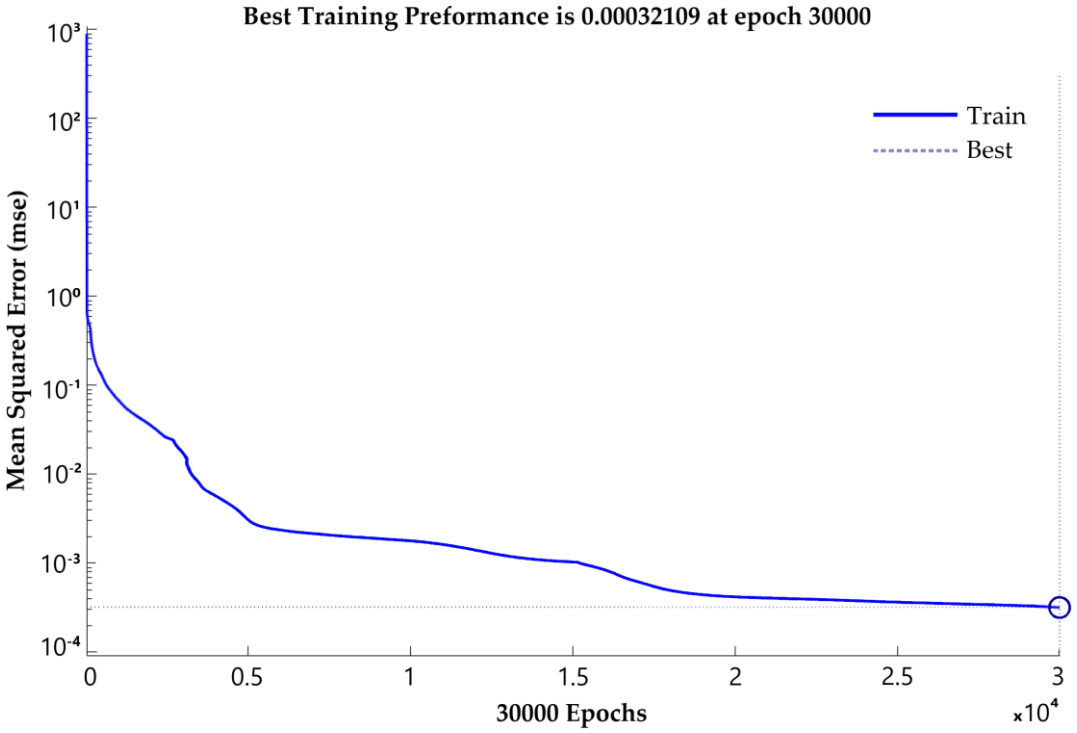


Figura 5.5: Error cuadrático medio en función del número de épocas *(Matlab)*

Además, en la Figura 5.5 se comprueba como la tasa de aprendizaje -*Training Performance*- tiene un valor de 0.00321. Esta tasa es un hiperparámetro[26] que controla cuánto cambia el modelo en respuesta al error estimado cada vez que se actualizan los pesos. Al ser un valor tan pequeño, facilita la precisión de la red neuronal, pues los "pasos" o saltos iterativos que da la red neuronal para buscar la solución más próxima al dato real son muy pequeños, y, por tanto, se comete menos error en las predicciones.

[26] En referencia al aprendizaje automático, se refiere a todo parámetro cuyo valor se utiliza para controlar el proceso de aprendizaje

Por otro lado, si se realiza un análisis de regresión del modelo, se puede comprobar como los datos predichos se ajustan al 99.987% a los datos reales. Los resultados de dicho análisis se observan en la Figura 5.6.

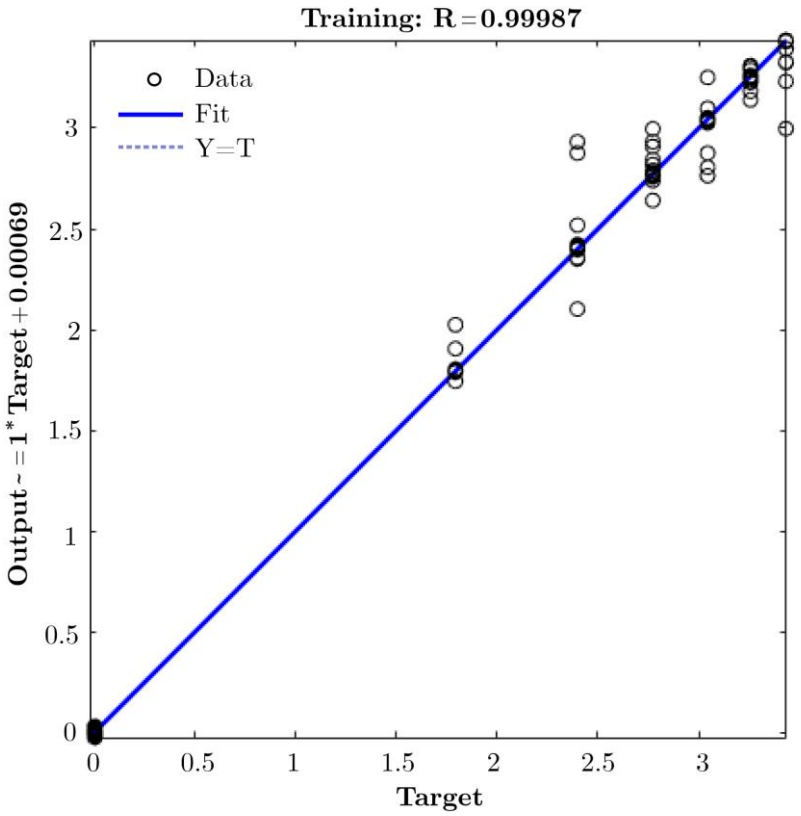


Figura 5.6: Análisis de regresión de la red neuronal *(Matlab)*

Además, en la Figura 5.6 se puede comprobar cómo gran parte de los datos se encuentran próximos a la recta de ajuste. Siendo unos pocos casos aislados los que quedan fuera del ajuste realizado.

Sin embargo, los puntos que no se ajustan con exactitud al modelo realizado, carecen de demasiada importancia puesto que, si se analiza los errores en las predicciones, se observa que la inmensa mayoría de los datos tienen un error del 0.02376 (véase Figura 5.7).

La totalidad de todos los datos -donde se incluyen los casos anómalos que no se ajustan a la recta de regresión- tienen un rango de error entre -0.002477 y 0.02376. Un error más que despreciable, en especial si se tiene en cuenta que los datos de las predicciones tienen valores entre 0 y 50 unidades.

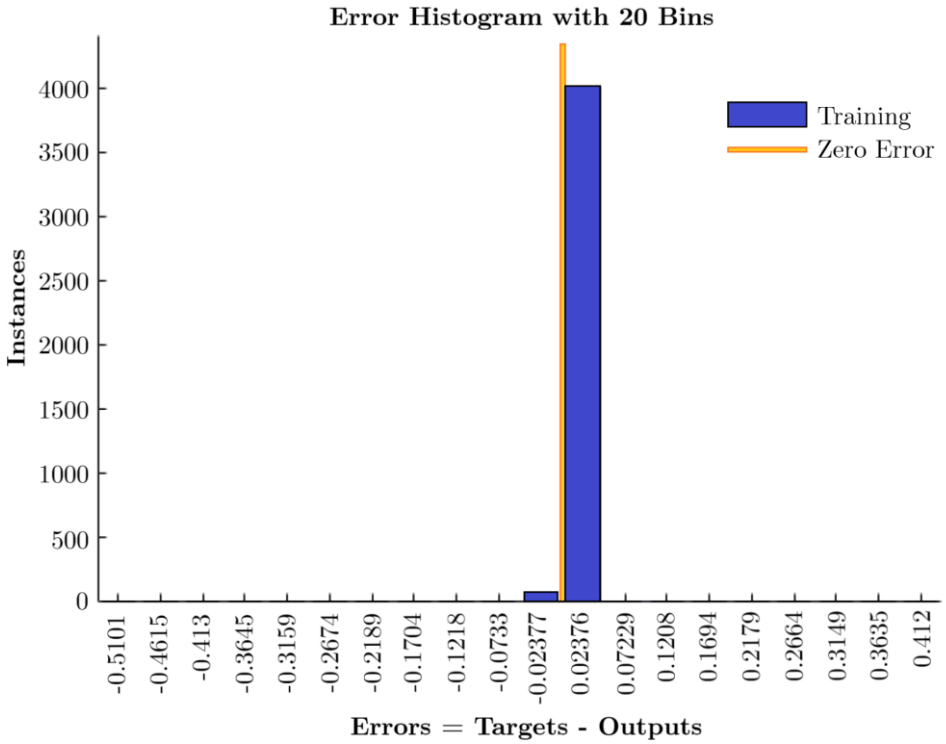


Figura 5.7: Histograma del error de las predicciones de la red neuronal *(Matlab)*

Por último, observando la evolución del gradiente (véase la Figura 5.8), puede comprobarse como este converge a un valor muy cercano a cero (0.000142). Lo mismo sucede con μ, que con un valor cercano a 10^{-7} garantiza una mayor precisión y rapidez del algoritmo de entrenamiento al transformarse en el método de Newton.

Con estos resultados, se justifica la capacidad y uso de la red neuronal realizada para la aplicación del control de velocidad lineal y angular del robot ápodo.

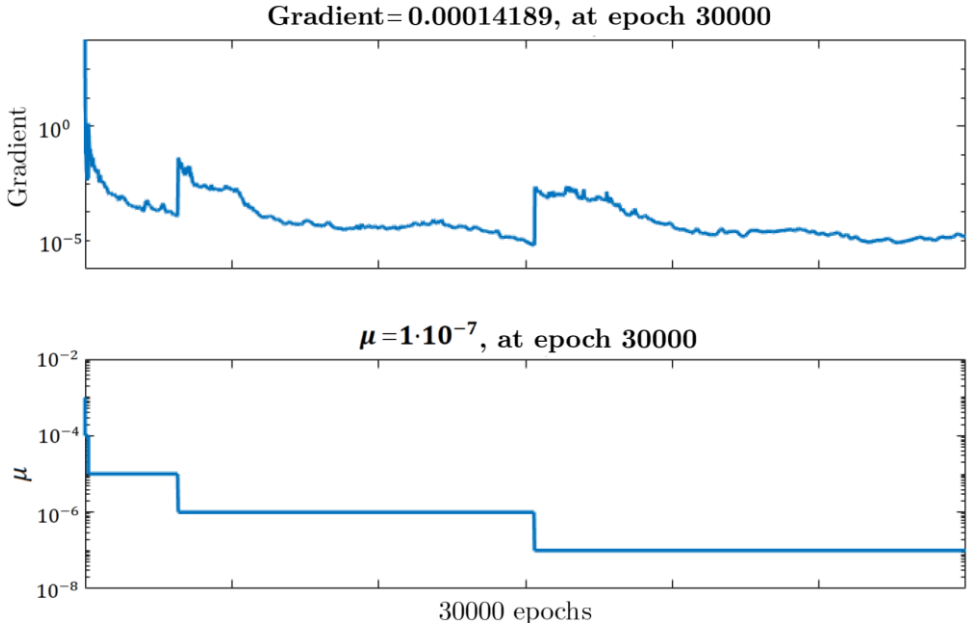


Figura 5.8: Evolución del gradiente y μ durante el entrenamiento de la red neuronal *(Matlab)*

5.3. INTERFAZ GRÁFICA DE CONTROL

Por último, para que el usuario pueda realizar el control de la velocidad realizada de forma sencilla, se desarrolla una interfaz gráfica. Esta interfaz ha sido programada en *Matlab* mediante el editor de diseño *GUIDE*. El uso de esta herramienta es debido a la sencillez que ofrece *Matlab* para la creación de interfaces y la facilidad de transferencia de la red neuronal, calculada en *Matlab* también.

El diseño de la GUI -*grafic user interface*- consiste en 4 deslizaderas o *sliders*, cada una para seleccionar el valor deseado de los parámetros a_v, a_h, v, y w. El valor numérico seleccionado se muestra también en la GUI, en la parte derecha del *slider*. De esta forma se conoce que parámetros se están fijando.

Dichos valores no son transmitidos a la red neuronal hasta que se pulsa el botón de 'calcular'. En ese momento, se mandan los valores fijados por el usuario en la interfaz gráfica a la red neuronal y esta devuelve las predicciones de los parámetros b_h y b_v.

El aspecto que muestra la interfaz es el que se observa en la Figura 5.9, donde en los campos que aparece "valor" se muestra los parámetros seleccionados por el usuario o devueltos por la red neurona.

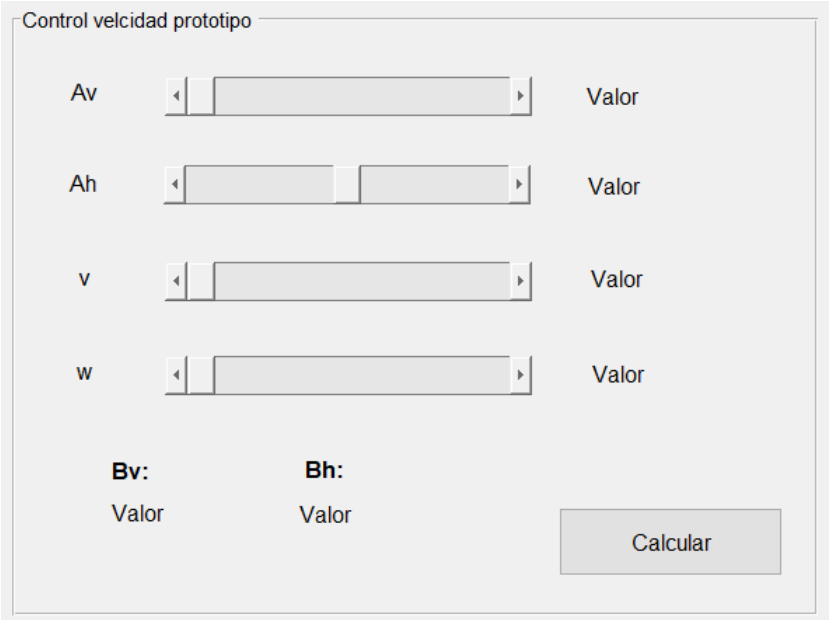


Figura 5.9: Interfaz gráfica de control para el usuario *(Matlab)*

Gracias a esta sencilla interfaz, cualquier usuario puede controlar y determinar de forma rápida y fácil las velocidades deseadas para el prototipo. Para una información más detallada de la interfaz gráfica realizada, puede observarse el código implementado en el Anexo B2.*4*.

CAPÍTULO 6.
MONTAJE FINAL Y VALIDACIÓN EXPERIMENTAL

En el presente Trabajo Final de Máster se realiza un estudio y análisis de los distintos movimientos ápodos.

Para realizar la simulación de dicho movimiento, se realizan diversos diseños de prototipos sobre los que se simula el movimiento ápodo. De esta forma se puede observar las ventajas y desventajas que ofrece cada configuración, concluyendo cual es la configuración óptima para implementar. Por último, se programa una red neuronal con el fin de poder realizar el control del prototipo de forma sencilla mediante las velocidades y amplitudes fijadas por el usuario.

Finalmente, se construye un prototipo a escala para realizar la validación del estudio llevado a cabo.

A lo largo de este capítulo se realiza la selección de los componentes electrónicos y la alimentación del prototipo. Justificando su elección y desarrollando sus especificaciones. Además, se explican las pautas seguidas para el ensamblaje, tanto mecánico como electrónico, y el código implementado para reproducir el movimiento serpentino sobre el prototipo.

Por último, se comparan diversos casos de movimientos sobre el prototipo real, comprobando su comportamiento y validando el estudio realizado sobre el movimiento ápodo.

6.1. SISTEMA DE ALIMENTACIÓN

Debido a la utilización de actuadores eléctricos, es necesario dotar al sistema de una fuente de alimentación que genere energía eléctrica. La elección de la fuente de alimentación se basa en seleccionar una batería con un coeficiente de descarga lo suficientemente grande como para poder alimentar a los 8 servomotores simultáneamente.

En la Tabla 3.2 se muestra la corriente que necesita el servomotor. Suponiendo el caso más desfavorable, el servomotor necesita una corriente de 150 mA sin carga. Como se tienen 8 servomotores que alimentar a la vez, la batería tiene que ser capaz de suministrar como mínimo una corriente de 1200mA.

Por esta razón se escoge la batería "Li-Po 8C 2S" con 7.4 V y una corriente de 4000mAh. Si se calcula su corriente de descarga, al tener una capacidad de 4000 mAh y una velocidad de descarga de 8 C -donde la unidad de C se mide en $1/h$ -, se obtiene una corriente de descarga de 32 A. Esta corriente es muy superior a la necesaria, por lo que la batería es adecuada para la aplicación desarrollada.

Las especificaciones de mayor interés se muestran en la Tabla 6.1, y la ficha técnica completa se adjunta en el Anexo A.2.

Tabla 6.1: Especificaciones batería Li-Po 7.4 V

Especificación	**Magnitud [ud]**
Voltaje nominal	7.4 V
Química	Li-Po
Capacidad	4 Ah
Velocidad de descarga	8 C
Conector	2.1 mm Barril
Dimensiones	68×49×17 mm
Peso	148.77 g
Número de celdas	2

Además, al tratarse de una batería Li-Po -polímero de litio- esta puede almacenar una gran cantidad de energía y ofrece unas prestaciones muy buenas. Sin embargo, es necesario la utilización de un cargador específico para Li-Po. En concreto, se utiliza el cargador. "ZHITING B3 20W 1.6A", que permite cargar baterías Li-Po de 2 y 3 celdas con seguridad.

Por último, se debe tener en cuenta que las baterías de tipo Li-Po deben usarse con gran precaución de no ser sobrecargadas o descargadas por debajo de un valor mínimo de corte de la célula (2.5V para la batería utilizada), puesto que se estropean con facilidad.

6.2. ELECTRÓNICA

Para el control de los servomotores se requiere un dispositivo capaz de enviar las órdenes programadas, así como transformar las posiciones angulares calculadas por el algoritmo, en las señales PWM utilizadas para controlar la posición de los servomotores. Por ello, se decide realizar una configuración de microcontrolador más una placa controladora de Servomotores.

El microcontrolador utilizado para la transmisión de órdenes es la placa de desarrollo ESP-01s. Una de las ventajas que tiene este controlador es la capacidad de programar con solo unos pocos pines disponibles, lo que reduce considerablemente su tamaño. Además, incluye un programador USB que facilita su uso y programación. Las especificaciones técnicas más relevantes se muestran en la Tabla 6.2.

Tabla 6.2: Especificaciones controlador ESP-01s

Especificación	**Magnitud [ud]**
Tensión de alimentación	3.3 V
Tensión de Entradas/Salidas	3.3 V
CPU	32 bits
Tipo	Módulo WiFi
Soporte de red	2.4 GHz
Protocolos soportados	TCP/IP
Dimensiones	25×14×1 mm

Por otro lado, para transformar las órdenes programadas a las señales PWM que reciben los servomotores, se utiliza la placa controladora de Servomotores PCA9685. Dicha placa tiene 16 salidas PWM disponibles, de esta forma, se pueden controlar los 8 servomotores utilizados en el prototipo dejando opción a posibles ampliaciones futuras. Las especificaciones técnicas de este componente se indican en la Tabla 6.3.

Tabla 6.3: Especificaciones controladora de servomotores PCA9685

Especificación	**Magnitud [ud]**
Tensión de alimentación	5 V
Alimentación motores (máx)	6 V
Frecuencia salida	40-1000 Hz
Canales	16
Resolución por canal	12 bits
Interfaz	I2C
Dimensiones	62.3×25.3 mm

Sin embargo, tal y como se observa en la Tabla 6.2 y Tabla 6.3, las tensiones de alimentación del microcontrolador y la placa controladora de servos son de 3.3 V y 5.0 V respectivamente, mientras que la batería utilizada proporciona un voltaje de 7.4 V.

Por ello, es necesario el uso de un regulador de voltaje para ajustar la alimentación de ambas placas. Para ello, se utiliza el adaptador LM2596, que regula la tensión de entrada, en un rango admisible de 3.2 a 40 V a una tensión de salida de 1.25 a 35 V. Las especificaciones técnicas se exponen con más detalle en la Tabla 6.4.

Tabla 6.4: Especificaciones convertidor de voltaje LM2596

Especificación	**Magnitud [ud]**
Voltaje de Entrada	4.5 a 40 V DC
Voltaje de Salida	1.23 a 37 V DC
Corriente de Salida	2.5 A (máx 3 A)
Potencia de Salida	25 W
Eficiencia del conversor	92%
Frecuencia de trabajo	150 KHz
Dimensiones	43×21×13 mm

Además, para una mayor información, se adjuntan del Anexo A.3 al Anexo A.5 las fichas de especificaciones completas de los componentes electrónicos seleccionados.

6.3. ENSAMBLAJE FINAL

Tras la selección de los componentes mecánicos y electrónicos, y la elección de la configuración óptima llevada a cabo en el CAPÍTULO 4, se desarrolla el montaje final del prototipo. Los componentes seleccionados para la construcción del prototipo real se resumen a continuación en la Tabla 6.5.

Tabla 6.5: Componentes necesarios para el montaje del prototipo

Componente	Uds./Módulo	Uds./Prototipo
Bracket ASB-04 (BA)	1	8
Bracket ASB-09 (BM)	1	8
Bracket ASB-06 (BP)	1	8
Servomotor Hitec HS-422 (S)	1	8
Batería Li-Po 7.4 V	-	1
Placa de desarrollo ESP-01s	-	1
Controladora PCA9685	-	1
Regulador Voltaje LM2596	-	2

Para realizar el ensamblaje se realiza el primer lugar el montaje de la parte mecánica. Posteriormente, cuando esta realizado el montaje mecánico al completo, se procede a realizar las conexiones de la electrónica del prototipo.

6.3.1. Ensamblaje mecánico

El diseño de la parte mecánica, como se explica en la Sección 3.3, consiste en el montaje de ocho módulos prácticamente idénticos, alternados 90° entre ellos. Cada módulo está compuesto por un servomotor y tres *brackets* que lo envuelven y sirven de unión entre las distintas piezas.

Previo a realizar el montaje, es conveniente posicionar todos los servomotores a 90°, para así permitir el movimiento de los módulos en ambos sentidos en un rango de 0° a 180°.

Una vez posicionado el Servomotor, se sitúa en el interior del *bracket* BA. La base del servomotor y uno de sus laterales quedan totalmente cubiertos por las caras del *bracket* BA, mientras que el eje del servomotor queda libre para poder realizar el movimiento.

A continuación, se agrega el *bracket* BM al montaje. Este *bracket* tiene forma de "L" y hace de unión entre los otros dos *brackets*. La parte interior de la "L" coincide con la esquina del *bracket* BA más alejada del eje del Servomotor. Por el otro lado de la "L", se une a la cara longitudinal del *bracket* BP, haciendo coincidir sus centros. Por último, uno de los dos lados del *bracket* BP se une con el eje del servomotor del módulo siguiente.

La diferencia entre las configuraciones de los módulos reside en la orientación del BM. Así los módulos pares presentarán la esquina de la "L" en el lado más próximo a la cara superior del servomotor, y los módulos impares en el lado opuesto.

En la Figura 6.1 se puede observar el montaje de uno de los módulos.

Figura 6.1: Vista delantera (izquierda) y trasera (derecha) de un módulo completo

Una vez realizado el montaje de todos los módulos, se procede a realizar su ensamblaje. La única condición que debe cumplir es que el eje de los servomotores de los módulos verticales quede posicionado en la parte superior, y el de los módulos horizontales en el lado izquierdo -siguiendo el sentido del movimiento-.

El montaje final de la parte mecánica del prototipo se muestra en la Figura 6.2.

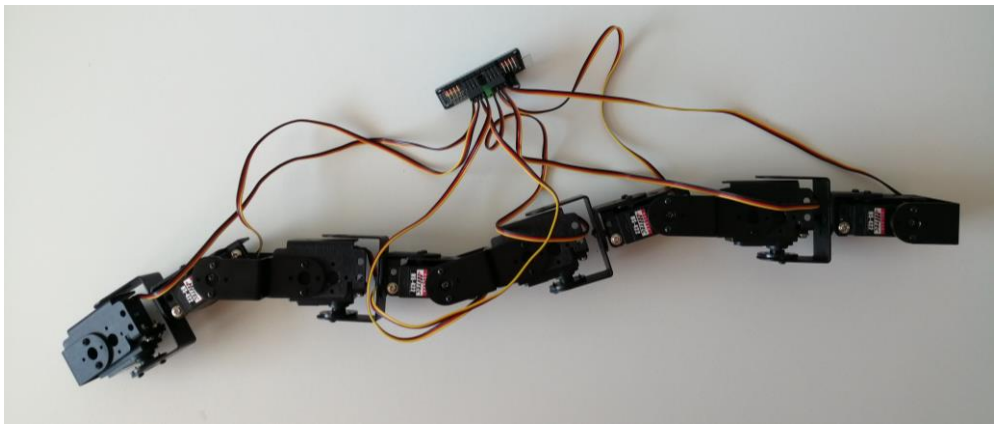


Figura 6.2: Montaje mecánico del prototipo completo bajo la configuración combinada

Con esta configuración, el prototipo robótico de serpiente posee una sección transversal de 5.22×5.22 cm. Con una longitud total de cada módulo igual a 6.96 cm, la longitud total de la serpiente asciende a 55.69 cm.

6.3.2. Ensamblaje electrónico

Los componentes electrónicos son los que dotan al robot de "inteligencia". El esquema seguido para montar la electrónica del sistema se muestra en la Figura 6.3, y se pueden distinguir tres etapas de montaje.

En primer lugar, se conectan los servomotores de cada módulo a la controladora PCA9685. Estos se conectan de forma ordenada a los pines 4 a 11, que generan una onda PWM para alimentar los servomotores.

La placa controladora se conecta al microcontrolador ESP-01s, encargado de ejecutar las órdenes grabadas en su memoria. La conexión se realiza por medio de I^2C bus serie[27], conectando los pines GPIO0 y GPIO2 del ESP-01s con los pines SDA y SCL de la controladora PCA9685.

Por último, se conectan ambas placas a la batería Li-Po 7.4 V. Para conseguir la alimentación adecuada del sistema es necesario regular las tensiones de alimentación por medio de dos reguladores de voltaje LM2596.

Uno de ellos regula el voltaje a 3.3 V para conectarlo al pin VCC de la ESP-01s. El otro regulador, tiene un voltaje de salida de 5 V y se conecta a los pines VCC y V+ de la controladora de Servomotores PCA9685. El primero para alimentar la placa y el segundo para mandar la alimentación a los servomotores. Los pines GND de ambas placas se conectan a cualquier terminal que actúe como masa de la batería.

Para realizar las conexiones a la batería Li-Po se utiliza una *protoboard*, ya que la batería utilizada solo tiene un terminal de alimentación y un terminal de masa, y se debe conectar dos componentes distintos a cada uno.

[27] Se refiere a un bus con múltiples maestros. Varios chips se pueden conectar al mismo bus y todos ellos pueden actuar como maestro.

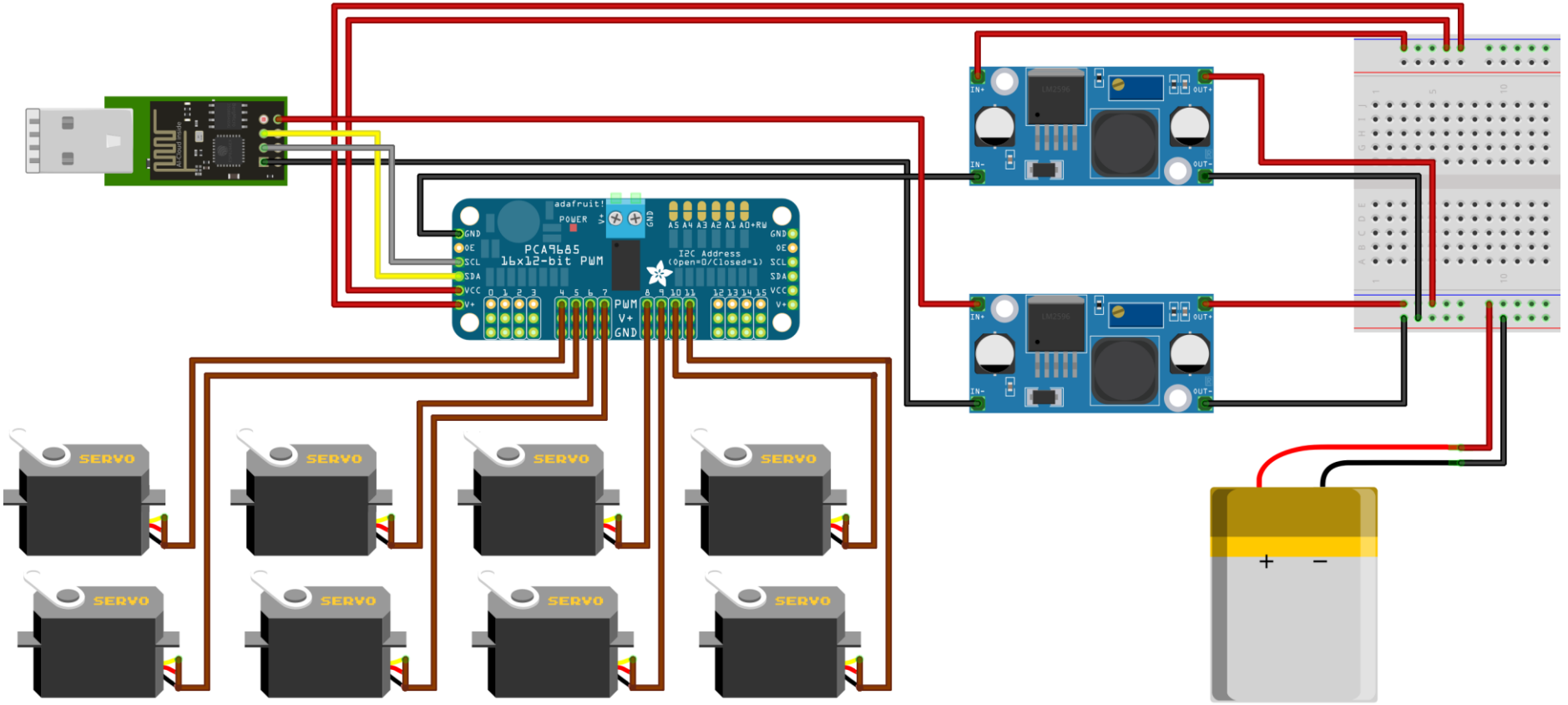


Figura 6.3: Diagrama de conexión de la electrónica del prototipo *(Fritzing, elaboración propia)*[28]

[28] Cabe destacar que todas las conexiones de los servos a la PCA9685, están compuestas por 3 cables cada una, correspondientes a la señal PWM, la alimentación de los propios servomotores (V+), y el terminal de tierra (GND)

El conexionado de todos los pines se muestra en la Tabla 6.6 a la Tabla 6.9.

Tabla 6.6: Conexionado PCA9685

Controladora Servos PCA9685	**PIN**
Servo Módulo 1	4
Servo Módulo 2	5
Servo Módulo 3	6
Servo Módulo 4	7
Servo Módulo 5	8
Servo Módulo 6	9
Servo Módulo 7	10
Servo Módulo 8	11
GPIO0 (ESP-01s)	SCL
GPIO2 (ESP-01s)	SDA
Alimentación para chip PCA9685	VCC
Alimentación para Servomotores	V+
Masa	GND

Tabla 6.7: Conexionado ESP-01s

Controladora Servos ESP-01s	**PIN**
SDA (PCA9685)	GPIO0
SCL (PCA9685)	GPIO2
OUT LM2596	VCC
Masa	GND

Tabla 6.8: Conexionado LM2596 regulada a 3.3V

Regulador voltaje LM2596	**PIN**
VCC (ESP-01s)	OUT
Alimentación Batería Li-Po 7.4V	IN

Tabla 6.9: Conexionado LM2596 regulada a 5 V

Regulador voltaje LM2596	**PIN**
VCC (PCA9685)	OUT
V+ (PCA9685)	OUT
Alimentación Batería Li-Po 7.4V	IN

6.4. PROGRAMACIÓN EN ARDUINO

Para la programación del prototipo robótico se utiliza el entorno de programación Arduino IDE.

Para poder programar la electrónica seleccionada, se debe realizar una configuración previa de Arduino. Por un lado, debe descargar un gestor adicional de tarjetas ESP8266 para poder programar el microcontrolador ESP-01s. Además, para poder programar y utilizar las funciones asociadas a la controladora PCA9685 se debe de descargar la librería "*Adafruit PWM Servo Driver*".

Una vez realizada la configuración, se procede a realizar el código para el control de posicionamiento de los Servomotores.

En primer lugar, se define una estructura del tipo `RobotServo_t` con los valores necesarios para controlar el servomotor. Dicha estructura está compuesta por el pin al que está conectado, el *offset* que presenta el servomotor y las posiciones mínima y máxima que puede alcanzar el servomotor –medidas en ángulos-.

A continuación, en el `setup()` se inicia el puerto serie -115200 baudios- y se inicializan los puertos de la controladora PCA9685 utilizados para la comunicación con la ESP-01s. Además, se definen los parámetros necesarios para el desarrollo del programa.

En la función principal `void loop()`, se implementa el código que calcula la posición que debe tener cada uno de los servomotores para realizar el movimiento serpentino.

Por último, se realiza la función `writeServo`, encargada de posicionar los servomotores en la posición calculada en la función principal.

Una vez alcanzadas las posiciones deseadas, se desactivan los servomotores con la función `detachServo`.

Dicho código se sube por medio del puerto serie al microcontrolador ESP-01s, quien se encarga de mandar las órdenes correspondientes a cada uno de los componentes que conforman el prototipo.

6.4.1. Validación del código en *CoppeliaSim*

Debido a la fragilidad de los componentes electrónicos del prototipo, se decide emular primero el movimiento del robot en *CoppeliaSim*. Esto implica que el microcontrolador ESP-01s, en vez de mandar las órdenes al resto de componentes, las manda a *CoppeliaSim* mediante el puerto serie.

Así, se puede ver el resultado del programa realizado en Arduino y lo que ejecutaría el robot real si se le mandase el código programado. De esta forma se evita posibles fallos que dañen el prototipo.

Para poder indicar a *CoppeliaSim* la posición de los servos del robot se define un protocolo de comunicación. Dicho protocolo consiste en mandar la letra '*S*' junto al pin al que está conectado. A continuación, se utiliza el separador ':' para indicar el ángulo de posición al que se desea mover el servomotor. Por último, se añade el separador ';' para finalizar el mensaje.

Por tanto, el resultado del mensaje enviado sería: "`S4:90;`", donde se indica que el servomotor conectado al pin 4 debe posicionarse a 90°. Este mensaje debe de enviarse para cada servomotor.

Para poder utilizar el robot emulado en *CoppeliaSim* desde Arduino, se define la constante `ROBOT_EMULATION`. Gracias a esta variable, se puede diferenciar en cada momento si se quiere implementar el programa en el prototipo real o en el robot emulado.

La diferencia que existe en el código en función de si se implementa el programa en el prototipo real o se emula en *CoppeliaSim* reside principalmente en la función `writeServo`. Dicha función enviará los datos de forma distinta en función de quién los vaya a recibir. Si se quiere realizar la emulación se manda mediante el protocolo descrito, mientras que si se quiere implementar en el robot real se fija directamente el valor del ángulo en el Servomotor correspondiente mediante la función `setPWM` de la librería "*Adafruit PWM Servo Driver*".

Cabe destacar que la variable `robotServo_t` se define de forma distinta si se está haciendo referencia al robot real o al emulado en *CoppeliaSim*, ya que, en este último, los pines representan directamente el número de articulación y no tiene offset de calibración.

Por último, se debe programar en *CoppeliaSim* la recepción de datos del puerto serie mediante la función `sim.serialOpen`, para abrir un puerto serie dentro del programa. Además, se crea en la función `sysCall_actuation()`, que lee y decodifica el mensaje recibido por el puerto serie. Una vez obtenidos los datos, se fija el valor en las articulaciones correspondientes emulando el movimiento programado en *Arduino*.

El código realizado en *Arduino,* tanto para la emulación en *CoppeliaSim* como para la validación en el prototipo real, y el código de recepción y lectura de *CoppeliaSim,* se adjuntan en el Anexo B.*3*.

6.5. ENSAYO FINAL DEL PROTOTIPO

Tras el montaje, se procede a realizar la validación del estudio realizado sobre el prototipo real. A lo largo de esta sección se presentan diversos casos de estudio que se han planteado sobre el prototipo real. Se estudia el comportamiento que muestra el robot, y se extraen las conclusiones obtenidas del ensayo.

6.5.1. Procedimiento de ensayo

El procedimiento seguido para realizar el ensayo de validación de la implementación de un movimiento ápodo es el descrito a continuación.

En primer lugar, se realiza la calibración de los Servomotores. Todos los servos son susceptibles de tener un "*offset*" de medida, por ello, antes que nada, es necesario determinar el *offset* de cada uno para poder corregir los ángulos en base a ello.

Con dicha finalidad, se programa que todos los servomotores fijen su posición a 90° y se observa las desviaciones de cada uno respecto a la horizontal. Así, por medio de pruebas, se realizan los ajustes necesarios hasta encontrar los *offset* de cada servomotor y poder dejar todo el prototipo completamente horizontal, como se muestra en la Figura 6.4.

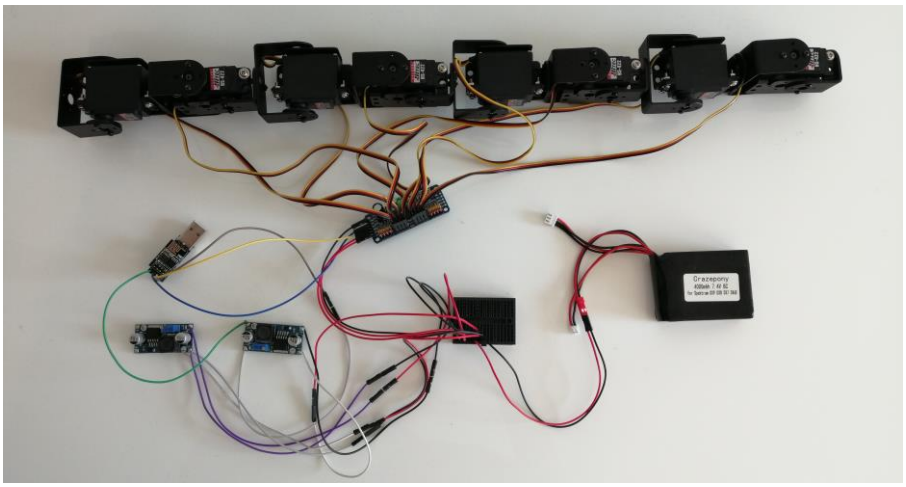


Figura 6.4: Prototipo completo en configuración de 90°

Además, se ha realizado una comprobación del rango de funcionamiento y orientación de cada servomotor. Dicha comprobación se lleva a cabo por medio de la implementación de un código que fija la posición de cada uno de los servomotores de forma individual, 20° en sentido horario y 20° en sentido antihorario respecto a la posición horizontal -coincidente con 90°-. De esta forma, se comprueba que los servomotores están correctamente montados y pueden girar en ambos sentidos.

Una vez comprobado el montaje, se decide que movimiento se quiere implementar y validar, es decir, si se quiere que vaya en una sola dirección o que realice giros, así como la velocidad de ambos movimientos.

Tras decidir el caso de estudio a implementar, se varían los parámetros v, w, a_h y a_v en la red neuronal, para poder obtener los parámetros b_h y b_v que definen por completo el movimiento serpentino.

Seguidamente, para evitar posibles fallos de coordinación del movimiento, se realiza la emulación en *CoppeliaSim* del movimiento que va a implementarse sobre el prototipo real.

Tras comprobar que la emulación realiza el movimiento definido de forma correcta, se realiza la validación sobre el prototipo real. El microcontrolador ESP-01s manda las órdenes a la controladora PCA9685 para que realice el posicionamiento de los servomotores con el fin de realizar el movimiento serpentino.

Por último, se extraen las conclusiones del movimiento realizado por el prototipo real. Para ello, se fijan los puntos inicial y final de desplazamiento, los cuales permiten recorrer una distancia rectilínea de 20 cm, y se cronometra el tiempo que tarda el prototipo en desplazarse de un punto a otro. También se calcula la posición final que alcanza el prototipo tras la simulación y, siguiendo el mismo razonamiento que el descrito en la Sección 5.2.2, se comprueba la velocidad lineal y angular que presenta el movimiento real.

6.5.2. Resultados

Para la validación del prototipo real, se realiza la simulación de cuatro casos de estudio -resumidos en la Tabla 6.10-. Con ellos, se pretende validar el estudio y diseño del control de movimiento de un robot ápodo desarrollado a lo largo del presente Trabajo Final de Máster.

Tabla 6.10: Resumen de casos a estudiar sobre el prototipo real

Caso de estudio	**Movimiento**	a_v	a_h	v **[m/s]**	w **[rad/s]**
Caso 1	Rectilíneo	0.2	0.0	0.003	0.00
Caso 2	Rectilíneo	0.4	0.0	0.015	0.00
Caso 3	Giro derecha	0.4	-0.1	0.008	0.04
Caso 4	Giro izquierda	0.4	0.1	0.012	0.06

Para las simulaciones se ha hecho uso de una superficie aterciopelada -ya que la apariencia de esta tela varía en función de la dirección de sus fibras- con el fin de que el prototipo deje unas marcas de paso sobre ella, cuya perpendicular coincide con la línea de dirección del movimiento.

Dicha decisión conlleva dos ventajas principales. En primer lugar, se puede comprobar el avance o paso en cada secuencia con la diferencia de longitud que hay entre dos marcas consecutivas. En segundo lugar, gracias a la marca inicial y final, se puede calcular las posiciones iniciales y finales en los ejes X e Y, así como el ángulo girado por el prototipo.

CASO 1: Avance en una sola dirección con velocidad lineal 0.003 m/s y velocidad angular 0 rad/s

Para realizar el avance en una sola dirección, los módulos horizontales no realizan movimiento, por ello, tanto a_h como b_h son 0. Además, al tratarse de un movimiento en línea recta, la velocidad angular w es 0 rad/s también.

En cuanto a la velocidad lineal, y el parámetro a_v, ambas variables se fijan a 0.003 m/s y 0.2 respectivamente. Para la primera prueba se escogen velocidades bajas para no realizar movimientos bruscos que dañen el prototipo.

Una vez definidos todos los parámetros por el usuario, se le introducen a la red neuronal, que devuelve el valor del parámetro b_v. El resultado proporcionado por la red neuronal se observa en la Figura 6.5.

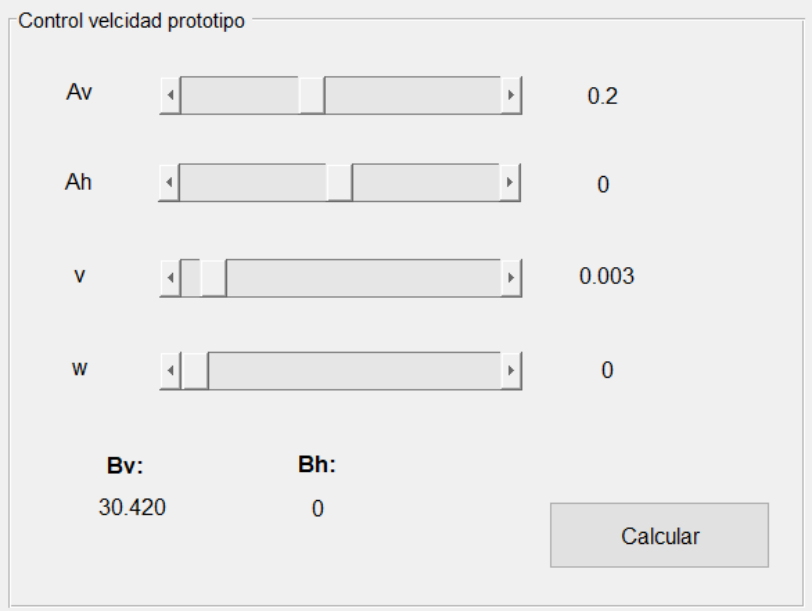


Figura 6.5: Resultados de la red neuronal para la validación del caso 1 *(Matlab)*

Con los valores de los parámetros $b_v = 30.420$ y $b_h = 0$, se realiza la emulación en *CoppeliaSim*, observándose como la simulación realiza el movimiento en línea recta de forma correcta.

Igualmente, se realiza la validación del movimiento sobre el prototipo real, obteniéndose el movimiento mostrado en la Figura 6.6.

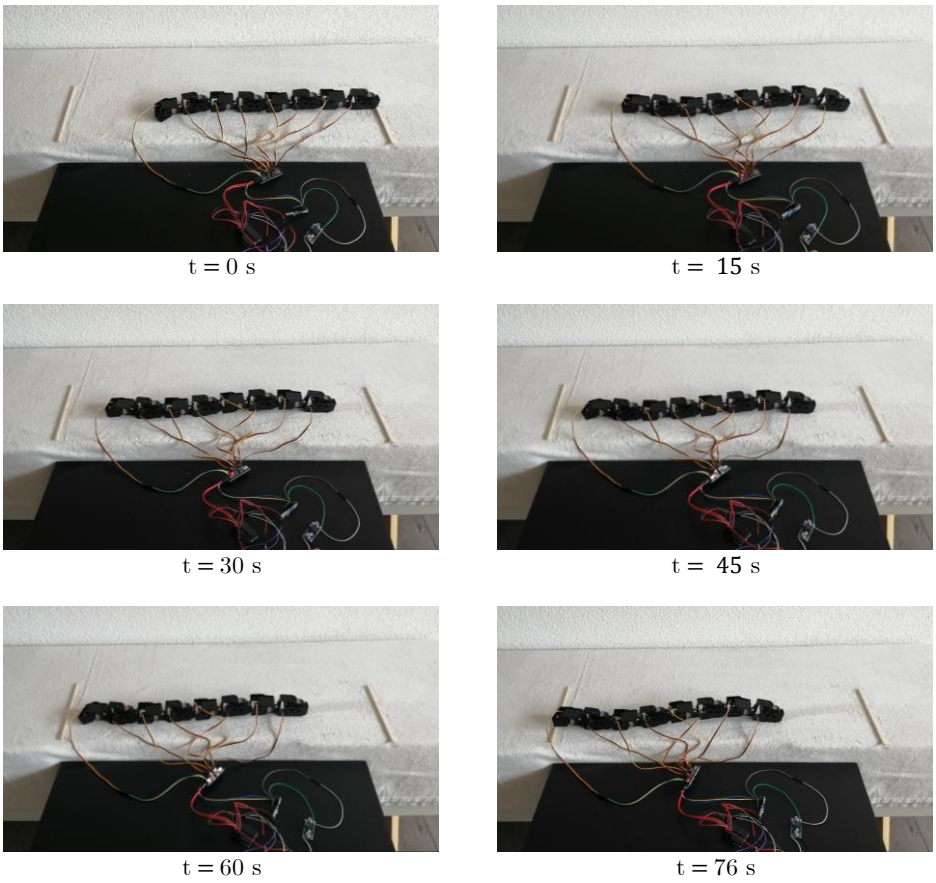


Figura 6.6: Validación en el prototipo real del caso 1

El tiempo que tarda en realizar el desplazamiento entre ambas marcas es de 76 segundos. Por tanto, la velocidad lineal que presenta el prototipo en la realidad es la calculada en la Ecuación 22.

$$v = \frac{A_{circ}}{t} = \frac{0.2\ m}{76\ s} = 0.0026\ m/s \qquad \text{[Ec. 22]}$$

Este resultado es ligeramente inferior a la velocidad lineal que se había fijado al inicio del proceso de validación para el presente caso de estudio, pero es lo suficientemente parecida para darse por validado el ensayo realizado.

CASO 2: Avance en una sola dirección con velocidad lineal 0.015 m/s y velocidad angular 0 rad/s

Tras analizar el primer caso, se prueba un segundo movimiento rectilíneo, en esta ocasión más rápido. Para ello, los parámetros fijados para la simulación son: $a_v = 0.4$, $a_h = 0$, $v = 0.015$ m/s y $w = 0\ rad/s$. Dichos parámetros se introducen a la red neuronal para obtener los valores $b_v = 59.887$ y $b_h = 0$, como se muestra en la Figura 6.7.

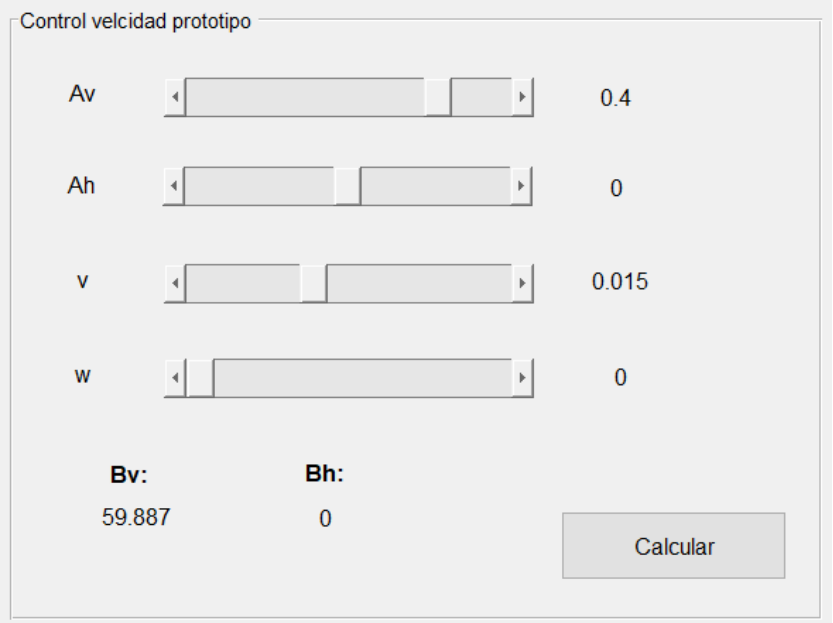


Figura 6.7: Resultados de la red neuronal para la validación del caso 2 *(Matlab)*

Tras conocer todos los parámetros que definen el movimiento serpentino, se realiza la emulación del movimiento, comprobándose que es correcta.

El comportamiento que muestra el prototipo real ante la situación estudiada se muestra en la Figura 6.8.

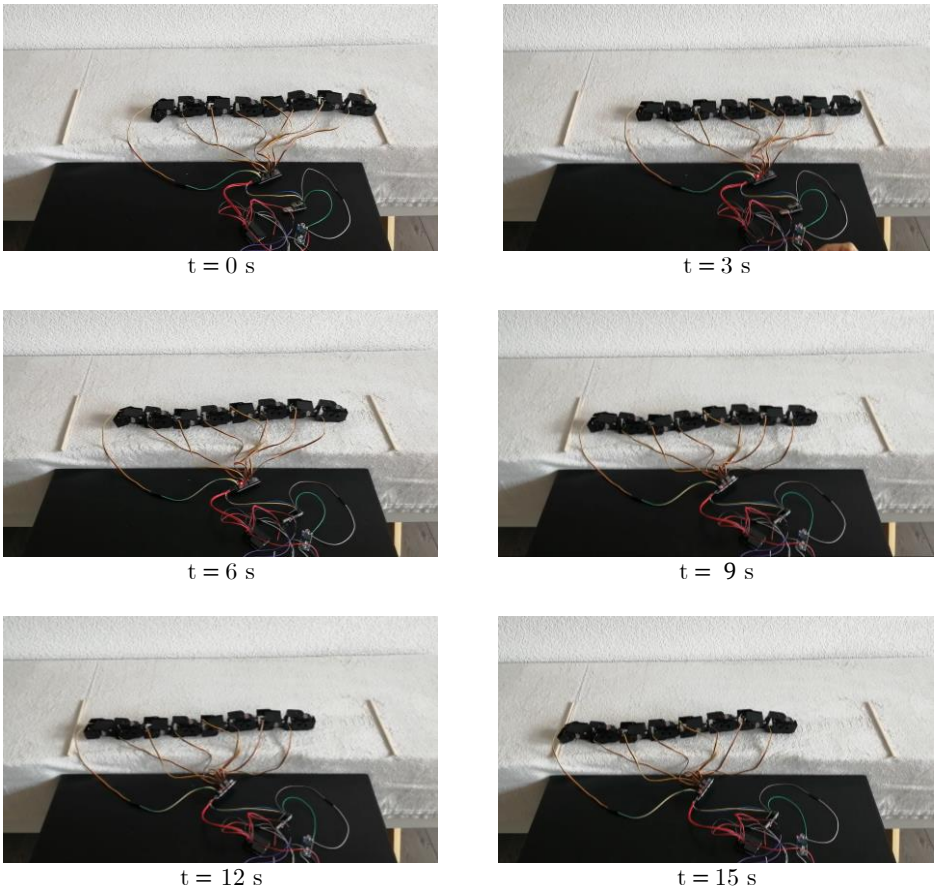


Figura 6.8: Validación en el prototipo real del caso 2

El tiempo que tarda en realizar el mismo desplazamiento que en el caso anterior, ahora es de 15 segundos, la quinta parte de lo que tardaba en el caso anterior. La velocidad a la que recorre este espacio se calcula en la Ecuación 23 y se comprueba con éxito como es muy similar a la velocidad que se había programado.

$$v = \frac{A_{circ}}{t} = \frac{0.2}{15} = 0.013\ m/s \qquad \text{[Ec. 23]}$$

CASO 3: Avance con giro a la derecha, con velocidad lineal 0.008 m/s y velocidad angular 0.04 rad/s

Una vez comprobado el avance en línea recta, se va a validar el giro del prototipo. En primer lugar, se va a estudiar el avance con giro a la derecha. Para ello, los parámetros seleccionados son: $a_v = 0.4$, $a_h = -0.1$, $v = 0.008$ m/s y $w = 0.04$ rad/s. Y los devueltos por la red neuronal: $b_v = 60.509$ y $b_h = 20.112$, tal y como se muestra en la Figura 6.9.

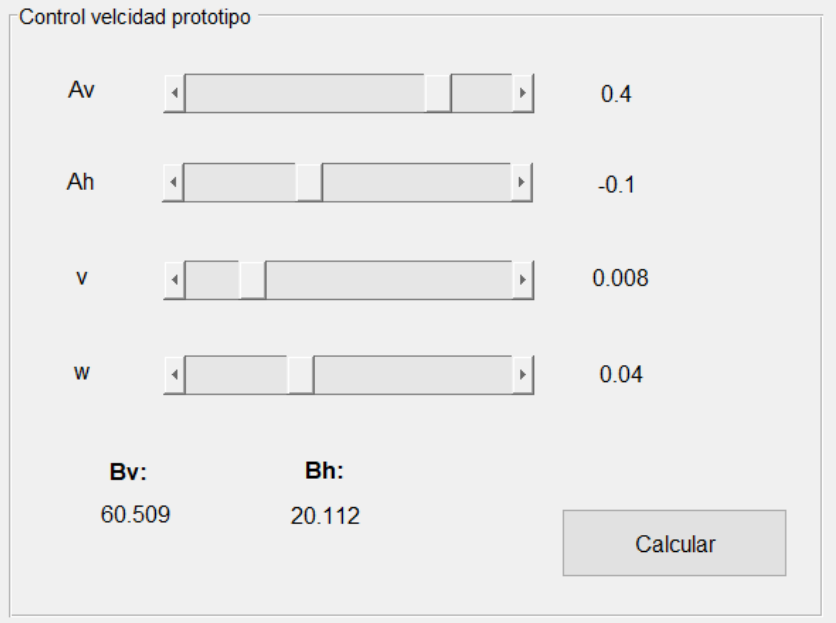


Figura 6.9: Resultados de la red neuronal para la validación del caso 3 *(Matlab)*

La emulación en *CoppeliaSim* muestra el comportamiento deseado, y la respuesta de la implementación en el prototipo real es la mostrada en la Figura 6.10.

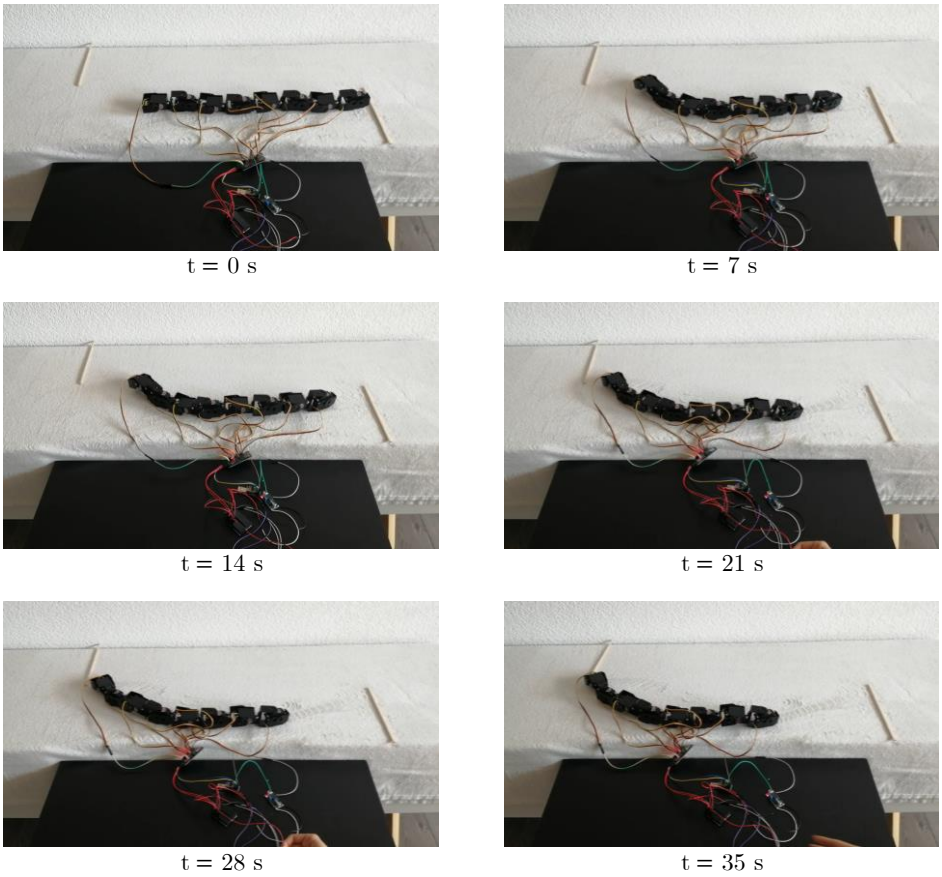


Figura 6.10: Validación en el prototipo real del caso 3

El tiempo de desplazamiento es de 35 s y la posición final que se alcanza es (-0.2, 0.05) m -siguiendo el convenio de signos en el que el sentido positivo del eje X corresponde con un desplazamiento hacia la derecha, y el sentido positivo del eje Y con un desplazamiento hacia arriba-. Además, basándose en las marcas de paso sobre el terciopelo iniciales y finales del primer módulo del prototipo, se ha estimado un ángulo de giro (σ) de $70°$.

Por tanto, la velocidad lineal y angular que presenta el prototipo -siguiendo el desarrollo explicado en la Sección 5.2.2- es la calculada en las Ecuaciones 24 y 25.

$$v = \frac{A_{circ}}{t} = 0.0065\ m/s \qquad \text{[Ec. 24]}$$

$$w = \frac{\sigma}{t} = 0.0359\ rad/s \qquad \text{[Ec. 25]}$$

Ambas son ligeramente inferiores a las fijadas inicialmente -en especial la velocidad lineal- pero entran dentro de un rango aceptable para validar el movimiento de giro del prototipo.

CASO 4: Avance con giro a la izquierda, con velocidad lineal 0.012 m/s y velocidad angular 0.06 rad/s

Por último, se va a analizar el giro a la izquierda del prototipo. Los parámetros seleccionados son: $a_v = 0.4$, $a_h = 0.1$, $v = 0.012$ m/s y $w = 0.06$ rad/s; para los que la red neuronal devuelve: $b_v = 68.452$ y $b_h = 31.184$, tal y como se muestra en la siguiente Figura 6.11.

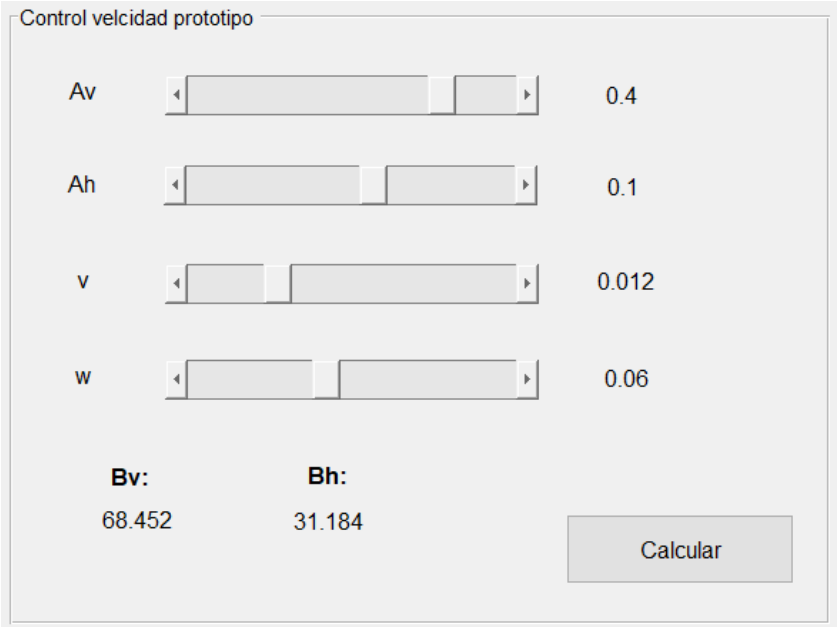


Figura 6.11: Resultados de la red neuronal para la validación del caso 4 *(Matlab)*

Se proponen dichos valores para comprobar un giro más rápido que el giro a la derecha analizado anteriormente.

Tras la correcta comprobación en *CoppeliaSim* del movimiento, se obtiene el movimiento del prototipo mostrado en la Figura 6.12.

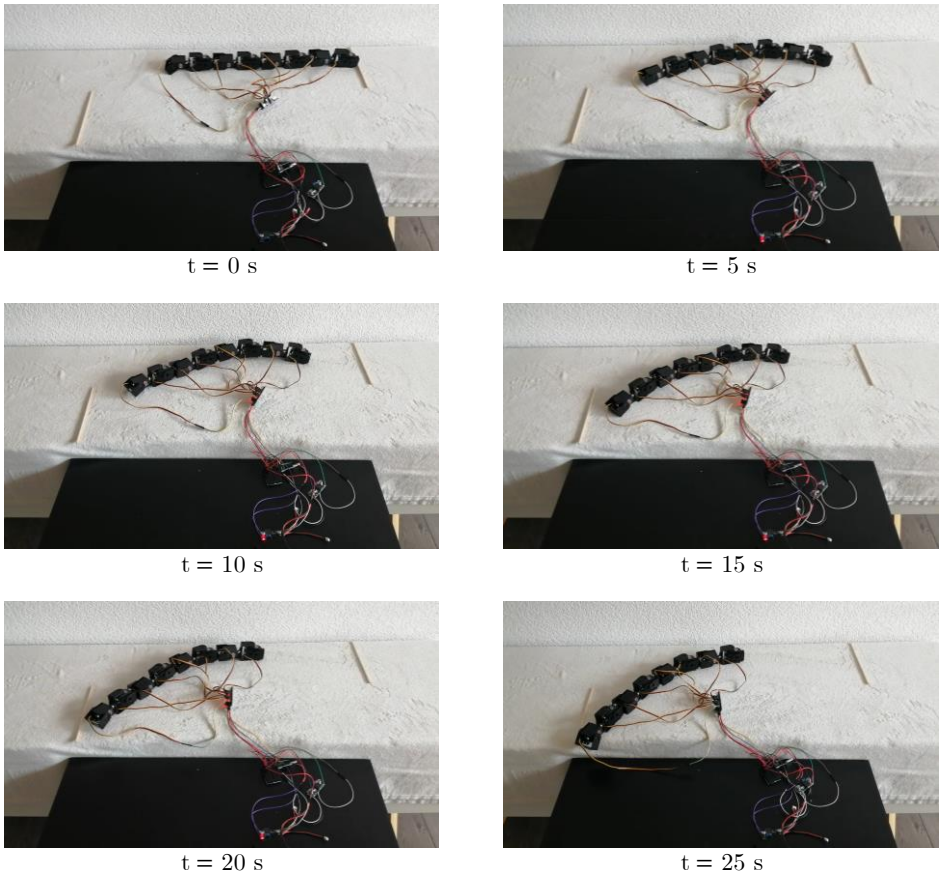


Figura 6.12: Validación en el prototipo real del caso 4

Se observa como el giro realizado en este caso es más rápido. Se consigue llegar a la posición (-0.2, -0.15) m en 25 s. Puede observarse un incremento de 0.1 m en 10 s menos con respecto al caso de velocidad reducida. Además, el giro es más pronunciado, girando un ángulo de 80°.

Las velocidades lineales y angulares del presente caso de estudio se muestran en las Ecuaciones 26 y 27.

$$v = \frac{A_{circ}}{t} = 0.0109\, m/s \qquad \text{[Ec. 26]}$$

$$w = \frac{\sigma}{t} = 0.0559\, rad/s \qquad \text{[Ec. 27]}$$

Se comprueba como ambas velocidades son similares a las que se pretendía conseguir en este caso de estudio

6.5.3. Conclusiones del ensayo

Tras el estudio de los cuatro casos descritos en el apartado anterior, se verifica y valida el correcto funcionamiento del movimiento ápodo estudiado a lo largo del presente Trabajo Final de Máster.

El prototipo representa el movimiento serpentino en dos direcciones de forma correcta, por medio de las trayectorias sinusoidales programadas.

Además, de las simulaciones se extraen dos conclusiones principales. En primer lugar, se comprueba que cuanto mayor son los valores de los parámetros a_h, a_v, b_h y b_v, mayor es la velocidad lineal y angular que presenta el prototipo.

Por otro lado, se observa como las velocidades lineales y angulares que presenta el prototipo real son ligeramente inferiores a las que se programan. Esto es debido a que el parámetro de fricción definido en la configuración de *CoppeliaSim*, no es exactamente igual que la fricción real -ya que este parámetro varía en función del medio en el que se realiza el movimiento-. Por ello, se dan ligeras diferencias en las velocidades reales respecto a las programadas en función de la superficie por la que se realiza el desplazamiento.

Sin embargo, las diferencias entre las velocidades reales y programadas son muy pequeñas y entran dentro de un rango aceptable para considerar correcto el movimiento programado. Esto indica que, tanto la red neuronal como el algoritmo del movimiento realizado presentan un buen funcionamiento y una precisión correcta.

Con todo ello, queda validado en el prototipo real, el estudio del movimiento de un robot ápodo en dos direcciones realizado a lo largo del presente Trabajo Final de Máster.

CAPÍTULO 7. CONCLUSIONES

Tras la realización del presente trabajo, son muchas las conclusiones que podemos extraer, a nivel tanto teórico como experimental. Como conclusión principal, se ha alcanzado con éxito el diseño y control de movimientos de un robot ápodo, comprobándose así la viabilidad de la locomoción ápoda aplicada al campo de la robótica.

Además, se ha señalado como su uso puede beneficiar a la sociedad en misiones de rescate o exploración, gracias a que el movimiento es independiente del terreno por el que se realiza. Se demuestra así que el uso de estos robots es perfecto cuando se desconoce el terreno por el que va a desplazarse.

Con respecto a las simulaciones realizadas, se ha podido configurar un *Software* de simulación con características realistas que imitan el comportamiento y fricción real que presenta el cuerpo de la serpiente.

Gracias a ello, se ha validado la creación de un modelo matemático para la generación de trayectorias sinusoidales. Comprobando en el software la similitud del movimiento generado por los algoritmos de movimiento con el de una serpiente real. Además, se ha comprobado el alcance del movimiento analizando el avance en línea recta y con cambios de dirección.

Asimismo, se ha concluido que la configuración óptima consiste en alternar 90° la orientación de los módulos consecutivos. Esta configuración es la única que permite el movimiento en dos dimensiones y garantiza la similitud al movimiento serpentino.

Por último, se ha podido comprobar la relación de dependencia existente entre los distintos parámetros que definen el movimiento serpentino. Se concluye así que los parámetros que tienen más peso en el movimiento ápodo son los relativos a la amplitud de la onda y al número de ondas. Además, se ha demostrado que cuanto mayor son dichos parámetros, más rápido es el movimiento descrito por el prototipo.

Por otro lado, en cuanto al control del prototipo, se ha conseguido controlar el movimiento serpentino a partir de las velocidades lineales y angulares por medio de una red neuronal.

Además, se han analizado distintos parámetros de entrenamiento de la red. Con ello, se ha concluido el número óptimo de neuronas por capa y de épocas para conseguir un menor error y un mayor rendimiento de las predicciones proporcionadas por la red neuronal.

Finalmente, se ha realizado una interfaz gráfica para facilitar el uso a nivel usuario.

En último lugar y para concluir, se ha conseguido diseñar y ensamblar un prototipo a escala. En él se ha probado con éxito el algoritmo realizado, así como el control del movimiento ápodo. Además, se ha podido comprobar cómo afecta el parámetro de fricción en los cálculos del movimiento realizado y como depende dicho parámetro del medio por el que se está realizando el movimiento.

En cuanto a líneas futuras, se plantea la posibilidad de realizar un cambio en el diseño mecánico con el fin de seleccionar unos *brackets* con un soporte para los componentes electrónicos del prototipo. De esta forma, al integrar la parte electrónica al diseño, se dota al prototipo de una mayor robustez y eficiencia, ya que no tendría que depender del arrastre de dichos elementos para realizar el movimiento.

Asimismo, se plantea la opción de una mayor automatización del prototipo por medio de sensores. De ese modo, el prototipo podría obtener datos del entorno y actuar en base a ellos con una mayor autonomía.

Con todo esto, se valida con éxito el estudio y diseño del control de movimientos de un robot ápodo realizado a lo largo del presente Trabajo Final de Máster.

CAPÍTULO 8. BIBLIOGRAFÍA

[1] Sutori.com. (2021). *Evolución de la Robótica.* Sutori.com [online]. Available at: https://www.sutori.com/story/evolucion-de-la-roboticas

[2] Bejerano, P. G. (2019). *«Automa cavaliere»: el robot que diseñó Leonardo da Vinci.* EL PAIS [online]. Available at: https://elpais.com/tecnologia/2019/04/25/actualidad/1556188737_786641.html

[3] García, B. (2018). *George Devol: vida y muerte del creador de la robótica industrial.* Blogthinkbig.com [online]. Available at: https://blogthinkbig.com/george-devol-el-creador-de-la-robotica-industrial

[4] Sánchez, C. (2017). *Shakey, el primer robot inteligente de la historia y el abuelo del coche autónomo.* ElDiario.es [online]. Available at: https://www.eldiario.es/shakey-robot-inteligencia-artificial-coche-autonomo.html

[5] R. (2020). *Breve historia visual de la inteligencia artificial.* www.nationalgeographic.com.es [online]. Available at: https://www.nationalgeographic.com.es/ciencia/historia-visual-inteligencia-artificial

[6] Next U. (2017). *La historia detrás de siri y su creador.* NextU LATAM [online]. Available at: https://www.nextu.com/blog/la-historia-detras-de-siri-y-su-creador-el-hombre-que-nacio-desarrollador-de-aplicaciones/

[7] Brownlee, J. (2020). *What is Deep Learning?* Machine Learning Mastery [Online]. Available At: https://machinelearningmastery.com/what-is-deep-learning/

[8] Wikipedia.org. (2021). *Aprendizaje profundo.* Wikipedia [online]. Available at: https://es.wikipedia.org/wiki/Aprendizaje_profundo

[9] Boston Dynamics. (2021). *Spot.* Boston Dynamics [Online]. Available At: https://www.bostondynamics.com/spot

[10] icirugiarobotica.com. (2019). *Cirugía Robótica Da Vinci.* iCirugiaRobotica [online]. Available at: http://www.icirugiarobotica.com/cirugia/robot-da-vinci/

[11] Coppeliarobotics.com. (2021). *Robot simulator CoppeliaSim.* Coppeliarobotics [Online]. Available At: https://www.coppeliarobotics.com/

[12] MATLAB (2021). *El lenguaje del cálculo técnico.* Mathworks [online]. Available at: https://es.mathworks.com/products/matlab.html

[13] Arduino.cc. (2021). *Arduino software.* Arduino [Online]. Available At: https://www.arduino.cc/en/software/

[14] Llena, I. (2019). *Clasificación de los robots según su función.* Esneca [online]. Available at: https://www.esneca.com/blog/clasificacion-robots-segun-funcion/

[15] Oxford University Press (OUP). (2021). *Robot.* Lexico.com [online]. Available at: https://www.lexico.com/es/definicion/robot

[16] Bonell Sanchez, M. (2011*). Diseño y contrucción de un robot humanoide.* upcommons.upc.edu, p. 4–7 [online]. Available at: https://upcommons.upc.edu/2099.1/12840/pfc1.pdf

[17] Instituto Politécnico Nacional. (2007). *Robots Móviles: Evolución y estado del arte.* redalyc.org, p. 13 [online]. Available at: https://www.redalyc.org/4026-40448003.pdf

[18] Gómez Cánovas, J. A. (2011). *Robótica móvil.* repositorio.upct.es, p.21-23 [online]. Available at: https://repositorio.upct.es

[19] González Gómez, J. (2008). *Robótica Modular y locomoción: Aplicación a Robots Ápodos.* researchgate.net, p. 3–5, [online]. Available at: https://www.researchgate.net/publication/41223679_Robotica_modular_y_locomocion_aplicacion_a_robots_apodos

[20] cs.rochester.edu. (2021). *Snakebot.* cs.rochester.edu [online]. Available at: https://www.cs.rochester.edu/u/nelson/courses/csc_robocon/robot_manual/snakebot/snakebot.html

[21] Gonzalez Gomez, J. G. G. (2008). *Robótica modular y locomoción: aplicación a robots ápodos.* iearobotics.com, p-19 [online]. Available at: http://www.iearobotics.com/downloads/Tesis-Juan-Gonzalez-Gomez.pdf

[22] González Gómez, J. (2002). *Principales líneas de investigación en robots tipo ápodos.* iearobotics.com, p-23-24 [online]. Available at: http://www.iearobotics.com/personal/juan/doctorado/Robots_apodos/estudio_apodos.pdf

[23] Universidad El Bosque. (2021). *Diseño e Implementación de un Sitema electrónico para Robot Modular tipo Serpiente.* robotics.umng.edu.co, p-1 [online]. Available at: http://robotics.umng.edu.co/publications/2017-Unibosque-Diseño_e_Implementación_de_un_Sistema_Electrónico_Para_Robot_Modular_Tipo_Serpiente.pdf

[24] R. Zug, G. (2021). *Locomotion.* Enciclopedia Britanica [Online]. Available At: https://www.britannica.com/topic/locomotion

[25] Pardos, A. C. & ProQuest. (2018). *La locomoción.* Alianza Editorial.

[26] National Geograpohic. (2021). *Serpientes: características y fotos.* www.nationalgeographic.com.es [online]. Available at: https://www.nationalgeographic.com.es/animales/serpientes

[27] Editorial. (2020). *Cómo se desplazan los animales.* Botanical [online]. Available at: https://www.botanical-online.com/animales/animales-terrestres-desplazamiento

[28] A. (2021). *Cómo se desplazan las serpientes.* Deserpientes [online]. Available at: https://deserpientes.net/como-se-desplazan-las-serpientes/

[29] Ambientech. (2020). *¿Qué es un movimiento peristáltico?* Ambientech [online]. Available at: https://ambientech.org/movimientos-peristalticos

[30] Centro de Automática y Robótica UPM-CSIC. (2020). *Análisis cinemático de patrones de movimiento para un robot tipo gusano.* oa.upm.es, p. 3–4 [online]. Available at: http://oa.upm.es/8024/2/INVE_MEM_2010_80462.pdf

[31] Greelane.com. (2020). *Hechos fascinantes, comportamientos y rasgos de las orugas.* greelane.com [online]. Available at: https://www.greelane.com/es/ciencia-tecnologia-matematicas/animales-y-naturaleza/fascinating-facts-about-caterpillars-1968169

[32] Bioscripts.net. (2021). *Características de los anélidos.* bioscripts.net [online]. Available at: https://www.bioscripts.net/zoowiki/temas/10D.html

[33] González Gómez, J. (2008). *Robótica modular y locomoción: Aplicación a robots ápodos.* iearobotics.com, p. 20–22 [online]. Available at: http://www.iearobotics.com/downloads/Tesis-Juan-Gonzalez-Gomez.pdf

[34] Tokyo Institute of Technology. (2013). Shigeo Hirose - *Relentless passion for the creation of robots.* Titech.Ac.Jp [Online]. Available At: https://www.titech.ac.jp/public-relations/research/shigeo-hirose

[35] CMU Biorobotics. (2021). *Modular Snake Robots.* biorobotics.ri.cmu.ed [online]. Available at: http://biorobotics.ri.cmu.edu/projects/modsnake/

[36] Rodríguez Rubio, F. (1996). *Control Adaptativo y Robusto.* researchgate.net [online]. Available at: https://www.researchgate.net/publication/Control-Adaptativo_y_Robusto/el_control_adaptativo_es_un_controlador_no_lineal,a_la_dinamica_del_bucle_ordinario_de_realimentacion

[37] EPFL. (2021). *AmphiBot.* Epfl.Ch [Online]. Available At: https://www.epfl.ch/labs/biorob/research/amphibious/amphibot/

[38] Miller, G. (1999). *S5 Snake Robot Prototype.* snakerobots.com [online]. Available at: http://snakerobots.com/S5.html

[39] University of Michigan. (2007). *The OmniTread OT-4 Serpentine Robot.* deepblue.lib.umich.edu [online]. Available at: https://deepblue.lib.umich.edu/bitstream/handle/2027.4256171/20196_ftp.pdf-sequence=1

[40] Laboratory of Intelligent Systems - EPFL. (2021). *Swarm-bots.* epfl.ch [online]. Available at: https://lis2.epfl.ch/CompletedResearchProjects/SwarmBots/

[41] San Millán Rodríguez, A. (2012). *Diseño, construcción y control de una serpiente robótica.* researchgate.net, p. 2–4 [online]. Available at: https://www.researchgate.net/publication/Diseno_construccion_y_control_de_una_serpiente_robotica

[42] Garzón Oviedo, M. A. (2011). *Estrategias Bio-Inspiradas para Locomoción de Robots Ápodos.* oa.upm.es, p. 100–103 [online]. Available at: http://oa.upm.es/16243/1/TFMMario.pdf

[43] Liu, J. (2004). *Path planning of a snake-like robot based on serpenoid curve and genetic algorithms.* researchgate.net, p.1-3 [online]. Available at: https://www.researchgate.net/publication/Path_planning_of_a_snake-like_robot_based_on_serpenoid_curve_and_genetic_algorithms

[44] Saito, M. (2021). *Modeling, analysis, and synthesis of serpentine locomotion with a multilink robotic snake.* projects.ics.forth.gr, p. 9–11 [online]. Available at: https://projects.ics.forth.gr/bioloch/internal/papers/snake_robot.pdf

[45] Brunete, A. (2021). *Actuadores hidráulicos.* bookdown.org [online]. Available at: https://bookdown.org/intro-automatica/actuadores-hidraulicos.html

[46] Platea.pntic.mec.es. (2021). *Sistemas de accionamiento.* platea.pntic.mec.es [online]. Available at: http://platea.pntic.mec.es /robotica/actuadores.htm

[47] Lynxmotion. (2021). *Lynxmotion - 3D Models.* Lynxmotion.com [online]. Available at: http://www.lynxmotion.com/s-5-ses-3d-models.aspx

[48] Coppeliarobotics. (2021c). *Shape dynamics properties.* coppeliarobotics.com [online]. Available at: https://www.coppeliarobotics.com/helpFiles/shapeDynamicsProperties.htm

[49] Coppeliarobotics. (2021e). *simGetObjectHandle.* coppeliarobotics.com [online]. Available at: https://www.coppeliarobotics.com/helpFiles/en/regular-Api/simGetObjectHandle.htm

[50] Coppeliarobotics. (2021f). *simSetJointTargetPosition.* coppeliarobotics.com [online]. Available at: https://www.coppeliarobotics.com/helpFiles/regular-Api/simSetJointTargetPosition.htm

[51] Significados.com. (2019). *Significado de Neurona*. Significados.com [online]. Available at: https://www.significados.com/neurona/

[52] Luhaniwal, V. (2020). *Forward propagation in neural networks.* Towardsdatascience.Com [Online]. Available At: https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250

[53] Vaughan, J. (2015). *Backpropagation.* Searchenterpriseai.Techtarget.com [Online]. Available At: https://searchenterpriseai.techtarget.com/definition/backpropagation-algorithm

[54] IBM. (2021). *El modelo de redes neuronales*. ibm.com [online]. Available at: https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=networks-neural-model

[55] Coppeliarobotics. (2021a). *Dummies.* coppeliarobotics.com [online]. Available at: https://www.coppeliarobotics.com/helpFiles/en/dummies.htm

[56] Serra, B. R. (2021). *Cuerda de la circunferencia*. universoformulas.com [online]. Available at: https://www.universoformulas.com/cuerda-circunferencia/

[57] Coppeliarobotics. (2021b). *Remote API.* coppeliarobotics.com [online]. Available at: https://www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm

[58] Coppeliarobotics. (2021d). *simCallScriptFunction.* coppeliarobotics.com [online]. Available at: https://www.coppeliarobotics.com/helpFiles/regular-Api/simCallScriptFunction.htm

[59] MathWorks España. (2021d). *Train shallow neural network - MATLAB train.* es.mathworks.com [online]. Available at: https://es.mathworks.com/help/deeplearning/ref/network.train.html

[60] MathWorks España. (2021b). *Mean squared normalized error performance function - MATLAB mse.* es.mathworks.com [online]. Available at: https://es.mathworks.com/help/deeplearning/ref/mse.html

[61] MathWorks España. (2021a). *Levenberg-Marquardt backpropagation - MATLAB trainlm*. es.mathworks.com [online]. Available at: https://es.mathworks.com/help/deeplearning/ref/trainlm.html

[62] Geogebra. (2017). *Método de Newton-Raphson*. GeoGebra.org [online]. Available at: https://www.geogebra.org/m/XCrwWHzy

[63] Rodríguez, D. (2018). *Implementación del método descenso del gradiente.* Analyticslane.com [online]. Available at: https://www.analyticslane.com/2018-12-21/implementacion-del-metodo-descenso-del-gradiente-en-python/

ANEXOS

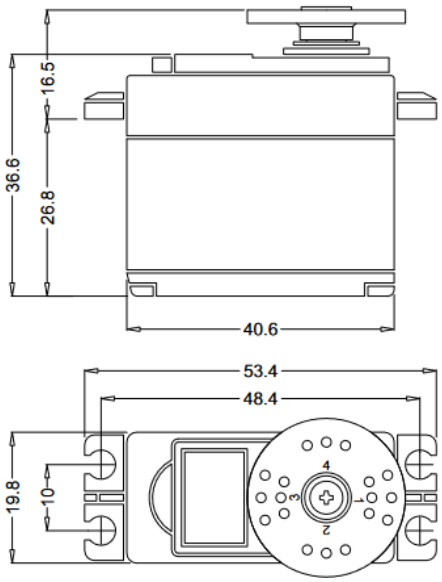
Índice de Anexos

ANEXO A: DOCUMENTACIÓN TÉCNICA

ANEXO A1: SERVOMOTOR HITEC HS-422

SERVO MOTOR HITEC HS-422

- Construcción durable y confiable
- Tren de engranajes de resina duradera
- Casquillos Oilite compatibles con eje de salida
- Circuitos de alto rendimiento



Especificación	Magnitud [ud]	
Sistema de control	+Pulse width control 1500 μs NEUTRAL	
Rango Voltaje de Operación	4.8 a 6.0 V	
Rango Temperatura de Operación	-20 a 60 °C	
Voltaje para Test	4.8 V	6.0 V
Velocidad de Operación (sin carga)	0.21 s/60°	0.16 s/60°
Par de bloqueo	3.3 kg/cm	4.1 kg/cm
Ángulo de Operación	90°/pulso lateral viajando 400 μs	
Dirección	Sentido horario/pulso viajando de 1500 a 1900 μs	
Consumo de corriente	8 mA/con carga y 150 mA/sin carga	
Ancho de banda	8 μs	
Longitud cable conector	300 mm	
Dimensiones	40.6x19.8x36.6 mm	
Peso	45.5 g	

ANEXO A2: BATERIA LI-PO DE 7.4 V

BATERIA LI-PO DE 7.4 V

- Batería de polímero de litio
- Proporciona alta calidad y potencia confiable para su transmisor.
- Conector balanceador

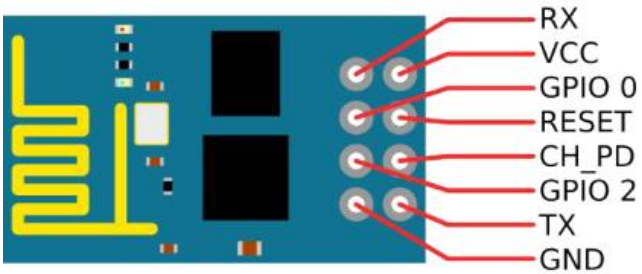
- JST: conector de alimentación para corriente continua
- JR: conector de descarga
- Conector de balance: cable balanceador para realizar la carga de la batería de forma equilibrada

Especificación	Magnitud [ud]
Capacidad	4000 mA
Velocidad de descarga	8 C
Tensión de salida	7.4
Número de celdas	2
Configuración	2S1P
Calibre del cable	22AWG
Dimensiones	68 × 49 × 17 mm
Peso	7 g

ANEXO A3: PLACA DE DESARROLLO ESP-01S

ESP-01s CON PROGRAMADOR USB

- Módulo pequeño
- Wifi integrado
- Incluye programador USB para facilitar la programación
- Soportado por Arduino IDE instalando paquetes y librerías adicionales



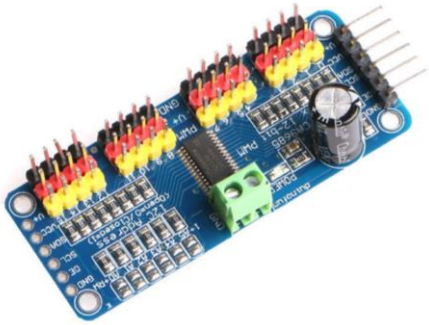
- RX: Serial de recepción
- VCC: 3.3 VDC
- GPIO 0: Conexión a VCC
- RESET: Conexión a VCC
- CH_PD: Conexión a VCC
- GPIO 2: Conexión a VCC
- TX: Serial de transmisión
- GND: Tierra

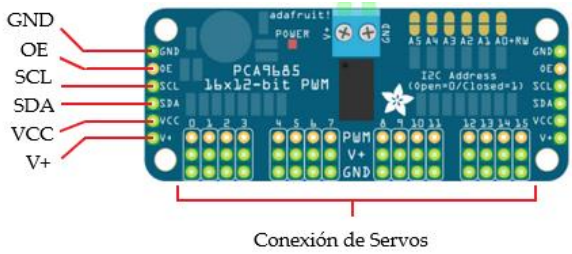
Especificación	Magnitud [ud]
Chip	ESP8266
Protocolo	802.11b/g/n, TCP/IP integrado
CPU	32 bits
WIFI	2.4 GHz, seguridad WPA/WPA2
Potencia de salida	802.11b mode + 19.5 dB
Corriente	$<$10 μA
Encendido/transferencia paquetes	$<$2 ms
Consumo de energía en espera	$<$1.0 mW
Rango Temperatura de Operación	-40 a 125 °C
Voltaje de alimentación	3.3 V
Comunicación serial	TX/RX
Dimensiones	25 × 14 × 1 mm
Peso	7 g

ANEXO A4: CONTROLADORA PCA9685 DE SERVOMOTORES

CONTROLADORA SERVOS PCA9685

- Permite el control de 16 canales con precisión
- Alimentación independiente de la señal de potencia a controlar
- Salidas PWM ajustadas a la misma frecuencia
- Interfaz de bus I2C compatible con Fast-mode Plus de 1 MHz





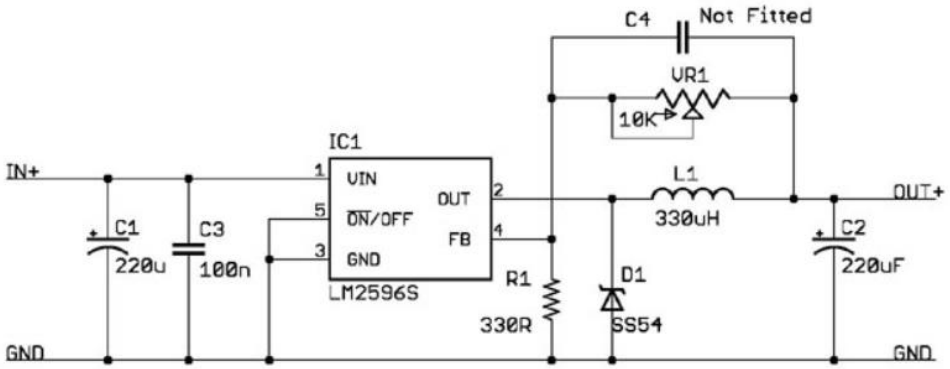
- GND: Tierra
- OE: Salida habilitada (*Output Enable*)
- SCL: BUS I2C
- SDA: BUS 12C
- VCC: Alimentación controladora
- V+: Alimentación par servo motores

Especificación	Magnitud [ud]
Canales	16
Rango Voltaje de Operación	2.3 a 5.5 V
Voltaje Alimentación	5.0 V
Alimentación motores máxima	6.0 V
Frecuencia de salida	40 a 1000 Hz
Resolución por canal	12 bits
Interfaz	I2C/Serie
Rango Temperatura de Operación	-40 a 85 °C
Paquetes ofrecidos	TSSOP28, HVQFN28
Dimensiones	62.3 × 25.3 × 11 mm
Peso	12 g

ANEXO A5: REGULADOR DE VOLTAJE LM2596

REGULADOR VOLTAJE LM2596

- Regulador de voltaje compuesto por circuitos integrados monolíticos
- Incluye compensación de frecuencia interna y un oscilador de frecuencia fija
- Requiere sólo cuatro componentes externos
- Excelentes especificaciones de regulación de línea y carga



Especificación	Magnitud [ud]
Tipo de regulador	StepDown(Entrada no aislada a la salida)
Voltaje de entrada	4 a 40 VDC
Voltaje de salida	1.23 a 35.00 VDC
Corriente de salida	2 A (máx. 3 A)
Eficiencia	>92%
Frecuencia de conmutación	150 kHz
Tensión de caída	2 VDC mínimo
Protección	Limitación de la corriente de cortocircuito
Regulación de la carga	±0.50%
Voltaje de regulación	±2.50%
Temperatura de Operación	-40 a 85°
Dimensiones	43.6 × 21 × 14 mm

ANEXO B: CÓDIGOS DE PROGRAMACIÓN

ANEXO B1: PROGRAMACIÓN DEL MOVIMIENTO DE LOCOMOCIÓN

A continuación, pueden encontrarse los códigos implementados en *CoppeliaSim* para la generación del movimiento serpentino. En el Anexo B1.1 puede visualizarse el código correspondiente a la configuración tanto horizontal como vertical, mientras que en el Anexo B1.*2* se adjunta el código referente a la configuración combinada.

Anexo B1.1: Código en CoppeliaSim de la configuración Vertical/ Horizontal

En el Anexo B1.1 puede visualizarse el código correspondiente a la configuración tanto horizontal como vertical, -ya que la diferencia entre ambas configuraciones radica en el diseño mecánico, no en la programación del movimiento-.

La estructura del código se basa en dos funciones. La primera, `sysCall_init()`, inicializa las articulaciones y extrae su identificador para referirse individualmente a cada una. La segunda función -la principal- es sobre la que se programa el movimiento serpentino.

```
function sysCall_init()
    -- Inicialización variables:
    art={}
    t0=0

    --Extraer identificador de cada articulación:
    for i=1,8,1 do
        art[i]=sim.getObjectHandle('articulacion'..i)
    end
end


function sysCall_actuation()
    -- Inicialización variables movimiento:
    paso=1e-4
    a=0.4
    b=30
    c=0
    s=0
    X=0
    Y=0
    Xant=0
    Yant=0
    pos={}
    n=8

    --Código generación del movimiento:
    for i=1,n do
        xtemp=0
        ytemp=0

        for k=1,i do
            xtemp=xtemp+(1/n)*math.cos(a*math.cos(k+t0*b/n)+c*(k+t0)/n)
            ytemp=ytemp+(1/n)*math.sin(a*math.cos(k+t0*b/n)+c*(k+t0)/n)
        end
```

```
            Xant=X
            Yant=Y
            X=xtemp
            Y=ytemp

            pos=math.atan((Y-Yant)/(X-Xant))
            sim.setJointTargetPosition(art[i],pos)
        end

        t0=t0+0.1 --Incremento del periodo temporal
    end
```

Anexo B1.2: Código en CoppeliaSim de la configuración Combinada

En el Anexo B1.*2* se adjunta el código referente al movimiento serpentino cuando la configuración es combinada. Para esta opción es necesario implementar el movimiento de las dos ondas de forma independiente. Así se consigue el movimiento en dos direcciones.

La programación para dicha configuración sigue la misma estructura que para la vertical/horizontal, con la diferencia de que se realizan todas las operaciones por duplicado.

```
function sysCall_init()
    -- Inicialización variables:
    art={}
    art_dir={}
    t0=0
    n=4
    n_skip=2

   --Extraer identificador de cada articulación:
    for i=1,n,1 do
        art[i] = sim.getObjectHandle('articulacion'..(i*n_skip-1))
        art_dir[i] = sim.getObjectHandle('articulacion'..(i*n_skip))
    end
end


function sysCall_actuation()
    --Definir parámetros para el avance:
    paso=1e-4
    a=0.15
    b=10
    c=0
    s=0
    X=0
    Y=0
    Xant=0
    Yant=0
    pos={}

    --Definir parámetros para controlar la dirección:
    a_dir=0.1
    b_dir=20
    c_dir=0
    s_dir=0
    X_dir=0
    Y_dir=0
    Xant_dir=0
    Yant_dir=0
    pos_dir={}
```

```
    --Código generación del movimiento:
    for i=1,n do
       xtemp=0
       ytemp=0
       xtemp_dir=0
       ytemp_dir=0

       for k=1,i do
           xtemp = xtemp+(1/n)*math.cos(a*math.cos((k+t0)*b/n)+
                   c*(k+t0)/n)
           ytemp = ytemp+(1/n)*math.sin(a*math.cos((k+t0)*b/n)+
                   c*(k+t0)/n)
           xtemp_dir = xtemp_dir+(1/n)*math.cos(a_dir*math.cos((k+t0)*
                       b_dir/n)+c_dir*(k+t0)/n)
           ytemp_dir = ytemp_dir+(1/n)*math.sin(a_dir*math.cos((k+t0)*
                       b_dir/n)+c*(k+t0)/n)
       end

       Xant=X
       Yant=Y
       X=xtemp
       Y=ytemp

       Xant_dir=X_dir
       Yant_dir=Y_dir
       X_dir=xtemp_dir
       Y_dir=ytemp_dir

       pos=math.atan((Y-Yant)/(X-Xant))
       sim.setJointTargetPosition(art[i],pos)

       pos_dir=math.atan((Y_dir-Yant_dir)/(X_dir-Xant_dir))
       sim.setJointTargetPosition(art_dir[i],pos_dir)
    end

    t0=t0+0.1 --mas el paso
end
```

ANEXO B2: PROGRAMACIÓN DEL SISTEMA DE CONTROL

El presente anexo puede dividirse a su vez, en dos partes bien diferenciadas.

La primera de ellas contiene el código de adquisición de datos realizado para el entrenamiento de la red neuronal. En primer lugar, se adjunta el código realizado en *Matlab R2019b* (véase el Anexo B2.1), y posteriormente se adjunta el código programado en CoppeliaSim para la recepción de ordenes de Matlab (véase el Anexo B2.2).

Posterior a la adquisición de datos, pueden encontrarse adjuntos los códigos de programación realizados para el desarrollo y entrenamiento de la red neuronal en *Matlab R2019b* (véase el Anexo B2.3), así como el código implementado para realizar la interfaz gráfica que facilita el uso de la red neuronal (véase el Anexo B2.*4*).

Anexo B2.1: Código en Matlab para la adquisición de datos:

La estructura de este script consiste en una primera parte encargada de realizar la conexión con *CopppelaSim* por medio de API's remotas. Seguidamente, se define las ordenes que se quieren enviar a CoppeliaSim. Por último, se analizan los datos extraídos de las simulaciones de CoppeliaSim.

```
%Establecimiento conexión con CoppeliaSim:
sim=remApi('remoteApi');
sim.simxFinish(-1);
clientID=sim.simxStart('127.0.0.1',1999,true,true,5000,5);

%Inicialización variables finales:
datos_entrada=[];
datos_salida=[];
pos=[];

%Bucle de toma de datos:
for ah=0.05:0.05:0.3
    for av=-0.15:0.05:0.15
        for bh=0:5:30
            for bv=0:5:30
                cv=0;
                ch=0;
                kstep=0.1;
                inputFloats=[ah av cv kstep]
                inputInts=[bh bv]
                inputStrings=[]
                inputBuffer=[]

                %Llamada a función para poner el prototipo en el origen:
                [res2, retInts2, ret_float2, retStrings2, retBuffer2] =
                 sim.simxCallScriptFunction(clientID,'tomadatos',sim.sim_scrip
                 ttype_childscript,'inicializacion',inputInts,inputFloats,inpu
                 tStrings,inputBuffer,sim.simx_opmode_blocking)

                tiempo=10;
                tic

                %Llamada a función para realizar el movimiento serpentino:
                while (toc<tiempo)
                    [res, retInts, ret_float, retStrings, retBuffer] =
                     sim.simxCallScriptFunction(clientID,'tomadatos',sim.sim_s
                     cripttype_childscript,'tomadato',inputInts,inputFloats,in
                     putStrings,inputBuffer,sim.simx_opmode_blocking)
                end
```

```
                if(res==sim.simx_return_ok)
                    %Cálculo de las velocidades lineales y angulares:
                    datos_entrada=[datos_entrada; inputFloats inputInts];

                    %los datos que se le introducen
                    theta=ret_float(1,3);
                    pos=[pos; ret_float];
                    L=sqrt(ret_float(1,1)^2+ret_float(1,2)^2);
                    R=L/(2*sin(theta/2));
                    S=theta*R;
                    T=tiempo;
                    v=S/T;
                    w=theta/T;
                    datos_salida=[datos_salida; v w]; %datos de interés
                    sim.delete(); % call the destructor!
                end
            end
        end
    end
end
```

Anexo B2.2: Código en CoppeliaSim para la adquisición de datos procedentes de Matlab

El presente código se encarga de recibir los parámetros enviados desde *Matlab* y realizar el movimiento serpentino en base a ello.
Cabe destacar, que en este código se incluyen la programación de los *dummies* para el cálculo de la posición y orientación del prototipo.

```
simRemoteApi.start(1999)

function tomadato(ints,floats,inStrings,inBuffer)
    c=0
    s=0
    X=0
    Y=0
    Xant=0
    Yant=0
    pos={}
    s_dir=0
    X_dir=0
    Y_dir=0
    Xant_dir=0
    Yant_dir=0
    pos_dir={}

    --Código movimiento serpentino
    for i=1,n do
        xtemp=0
        ytemp=0
        xtemp_dir=0
        ytemp_dir=0

        for k=1,i do
            xtemp = xtemp+(1/n)*math.cos(floats[1]*math.cos((k+t0)*ints[1]/n)+
                    c*(k+t0)/n)
            ytemp = ytemp+(1/n)*math.sin(floats[1]*math.cos((k+t0)*ints[1]/n)+
                    c*(k+t0)/n)
            xtemp_dir = xtemp_dir+(1/n)*math.cos(floats[2]*math.cos((k+t0)*
                        ints[2]/n)+ floats[2]*(k+t0)/n)
            ytemp_dir = ytemp_dir+(1/n)*math.sin(floats[2]*math.cos((k+t0)*
                        ints[2]/n)+c*(k+t0)/n)
        end
```

```
            Xant=X
            Yant=Y
            X=xtemp
            Y=ytemp

            Xant_dir=X_dir
            Yant_dir=Y_dir
            X_dir=xtemp_dir
            Y_dir=ytemp_dir

            pos=math.atan((Y-Yant)/(X-Xant))
            sim.setJointTargetPosition(art[i],pos)

            pos_dir=math.atan((Y_dir-Yant_dir)/(X_dir-Xant_dir))
            sim.setJointTargetPosition(art_dir[i],pos_dir)
        end

        t0=t0+0.1 --más el paso
        pose=updateRobotPose()
        orientation=sim.getObjectOrientation(ref_point,-1)

        --Obtengo la posicion y la orientación final
        pos_final=sim.getObjectPosition(robot_pose, -1)
        identificador=sim.getObjectHandle('robot_pose')
        orientacion_final=sim.getObjectOrientation(identificador,-1)

        --Datos a devolver:
        ret_int={1, 1, 2}
        ret_buffer={}
        ret_string=''
        ret_float={pos_final[1], pos_final[2], orientacion_final[3]}
    end


    function inicializacion(ints,floats,inStrings,inBuffer)
        --Posicionar en el Inicio cada Modulo
        sim.setObjectPose(id_8,-1,pose_m8)
        sim.setObjectPose(id_7,-1,pose_m7)
        sim.setObjectPose(id_6,-1,pose_m6)
        sim.setObjectPose(id_5,-1,pose_m5)
        sim.setObjectPose(id_4,-1,pose_m4)
        sim.setObjectPose(id_3,-1,pose_m3)
        sim.setObjectPose(id_2,-1,pose_m2)
        sim.setObjectPose(id_1,-1,pose_m1)
        sim.setObjectPose(id_0,-1,pose_m0)

        --Datos a devolver (valores aleatorios, necesario para evitar el error)
        ret_int2={1,1,1}
        ret_float2={1.1,1.2,1.3}
        ret_buffer2={}
        ret_string2=''

        return ret_int2, ret_float2, ret_buffer2, ret_string2
    end


    function sysCall_init()
        -- Incialización:
        art={}
        art_dir={}
        robot_pose=sim.getObjectHandle('robot_pose')
        inicial_pose=sim.getObjectHandle('robot_pose')
        ref_point=sim.getObjectHandle('ref_point')
        incial_pose=updateRobotPose()
        pose=updateRobotPose()
```

```
    --Cálculo de la posición De cada módulo:
    --Identificador
    id_8 = sim.getObjectHandle('tomadatos')
    id_7 = sim.getObjectHandle('tuerc_7')
    id_6 = sim.getObjectHandle('tuerc_6')
    id_5 = sim.getObjectHandle('tuerc_5')
    id_4 = sim.getObjectHandle('tuerc_4')
    id_3 = sim.getObjectHandle('tuerc_3')
    id_2 = sim.getObjectHandle('tuerc_2')
    id_1 = sim.getObjectHandle('tuerc_1')
    id_0 = sim.getObjectHandle('brk1_1')

    --Posicion
    pose_m8 = sim.getObjectPose(id_8,-1)
    pose_m7 = sim.getObjectPose(id_7,-1)
    pose_m6 = sim.getObjectPose(id_6,-1)
    pose_m5 = sim.getObjectPose(id_5,-1)
    pose_m4 = sim.getObjectPose(id_4,-1)
    pose_m3 = sim.getObjectPose(id_3,-1)
    pose_m2 = sim.getObjectPose(id_2,-1)
    pose_m1 = sim.getObjectPose(id_1,-1)
    pose_m0 = sim.getObjectPose(id_0,-1)

    t0=0
    n=4
    n_skip=2

    for i=1,n,1 do
        art[i]=sim.getObjectHandle('articulacion'..(i*n_skip-1))
        art_dir[i]=sim.getObjectHandle('articulacion'..(i*n_skip))
    end
end



function sysCall_sensing()
    robot_pose=sim.getObjectHandle('robot_pose')
    pose=updateRobotPose()
end



function updateRobotPose()
    local pose
    position=sim.getObjectPosition(robot_pose,-1)
    orientation=sim.getObjectOrientation(robot_pose,-1)
    pose={position[1],position[2],orientation[3]}
    return pose
end
```

Anexo B2.3: Código en Matlab para la red neuronal.

```
%Adquisición de Datos:
load('datos_entrada');
load('datos_salida')
x=[datos_entrada(:,1) datos_entrada(:,2) datos_salida(:,:)];
y=[datos_entrada(:, 5:6)];

%Normalización de datos:
y2=log(1+y);
x2=[];
for i=1:4
    x2(:,i)=(x(:,i)-min(x(:,i)))/(max(x(:,i))-min(x(:,i)));
end

xt=x2';
yt=y2';


%Definición de los parámetros de la red:
hiddenLayerSize=800;
net = fitnet(hiddenLayerSize);
net.divideParam.trainRatio=80/100;
net.divideParam.valRatio=20/100;
net.divideParam.testRatio=0/100;
net.trainParam.epochs=30000;
net.trainParam.max_fail=10;

load('net.mat')
net.divideParam.trainRatio=0/100;
net.divideParam.valRatio=20/100;

%Entrenamiento de la red:
[net, tr]=train(net, xt, yt);

%Determinación del Error:
yTrain=exp(net(xt(:,tr.trainInd)))-1;
yTrainTrue=exp(yt(tr.trainInd))-1;
sqrt(mean((yTrain-yTrainTrue).^2));
yVal = exp(net(xt(:,tr.valInd)))-1;
yValTrue=exp(yt(tr.valInd))-1;
sqrt(mean((yVal-yValTrue).^2));

save('net');
```

Anexo B2.4: Código en Matlab para la implementación de la GUI

```
function varargout = interfaz_grafica(varargin)
% INTERGAZ_GRAFICA MATLAB code for intergaz_grafica.fig
% INTERGAZ_GRAFICA, by itself, creates a new INTERGAZ_GRAFICA or raises
  the existing singleton*.
% H = INTERGAZ_GRAFICA returns the handle to a new INTERGAZ_GRAFICA or
  the handle to the existing singleton*.
% INTERGAZ_GRAFICA('CALLBACK',hObject,eventData,handles,...) calls the
  local function named CALLBACK in INTERGAZ_GRAFICA.M with the given
  input arguments.
% INTERGAZ_GRAFICA('Property','Value',...) creates a new INTERGAZ_GRAFICA
  or raises the existing singleton*. Starting from the left, property
  value pairs are applied to the GUI before intergaz_grafica_OpeningFcn
  gets called. An unrecognized property name or invalid value makes
  property application stop. All inputs are passed to
  intergaz_grafica_OpeningFcn via varargin.
%*See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
  instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help intergaz_grafica
% Last Modified by GUIDE v2.5 01-Sep-2021 16:32:16


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

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


% Executes just before intergaz_grafica is made visible.
function interfaz_grafica_OpeningFcn(hObject, eventdata, handles, varargin)
    % This function has no output args, see OutputFcn.
    % hObject    handle to figure
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    % varargin   command line arguments to intergaz_grafica (see VARARGIN)

    % Choose default command line output for intergaz_grafica
    handles.output = hObject;

    % Update handles structure
    guidata(hObject, handles);

    % UIWAIT makes intergaz_grafica wait for user response (see UIRESUME)
    % uiwait(handles.figure1);
```

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

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


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
    global Ah Av v w
    % hObject    handle to pushbutton1 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    load('net.mat')
    x_fut = [Ah; Av; v; w];
    x_fut_norm = (x_fut(:,1)-min(x_fut(:,1)))/(max(x_fut(:,1))-
                  min(x_fut(:,1)));
    y_fut_norm = sim (net,x_fut_norm);
    y_fut = exp(y_fut_norm)-1;
    Bh = y_fut(1,1);
    Bv = y_fut(2,1);
    set(handles.text12, 'String', Bh)
    set(handles.text13, 'String', Bv)


% --- Executes on slider movement.
function slider6_Callback(hObject, eventdata, handles)
global Ah
    % hObject    handle to slider6 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'Value') returns position of slider
    %        get(hObject,'Min') and get(hObject,'Max') determine range of slider
    get(hObject,'Value');
    Ah=hObject.Value;
    set(handles.text7, 'String', Ah)


% --- Executes during object creation, after setting all properties.
function slider6_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to slider6 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

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


% --- Executes on slider movement.
function slider7_Callback(hObject, eventdata, handles)
    global Av
    % hObject    handle to slider7 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'Value') returns position of slider
    %        get(hObject,'Min') and get(hObject,'Max') determine range of slider
    get(hObject,'Value');
    Av = hObject.Value;
    set(handles.text8, 'String', Av)
```

```
% --- Executes during object creation, after setting all properties.
function slider7_CreateFcn(hObject, eventdata, handles)
    % hObject   handle to slider7 (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles   empty - handles not created until after all CreateFcns called

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


% --- Executes on slider movement.
function slider8_Callback(hObject, eventdata, handles)
    global v
    % hObject    handle to slider8 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'Value') returns position of slider
    %        get(hObject,'Min') and get(hObject,'Max') determine range of slider
    get(hObject,'Value');
    v=hObject.Value;
    set(handles.text9, 'String', v)


% --- Executes during object creation, after setting all properties.
function slider8_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to slider8 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

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


% --- Executes on slider movement.
function slider9_Callback(hObject, eventdata, handles)
    global w
    % hObject    handle to slider9 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'Value') returns position of slider
    %        get(hObject,'Min') and get(hObject,'Max') determine range of slider
        get(hObject,'Value');
        w=hObject.Value;
        set(handles.text10, 'String', w)


% --- Executes during object creation, after setting all properties.
function slider9_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to slider9 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

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

```
% --- Executes on slider movement.
function slider10_Callback(hObject, eventdata, handles)
    % hObject    handle to slider10 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'Value') returns position of slider
    %        get(hObject,'Min') and get(hObject,'Max') determine range of slider
    get(hObject,'Value')
    Kstep = hObject.Value;
    set(handles.text11, 'String', Kstep)


% --- Executes during object creation, after setting all properties.
function slider10_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to slider10 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

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

ANEXO B3: PROGRAMACIÓN DEL FUNCIONAMIENTO FINAL

En el presente anexo se adjunta el código realizado en Arduino IDE para la programación del prototipo.

El código está programado para poder enviar las ordenes al prototipo real, así como para realizar la emulación del prototipo por puerto serie en *CoppeliaSim*. Para elegir una de las dos opciones solo se debe de definir, o no, `ROBOT_EMULATION`.

Así, si está definido dicha variable, el microcontrolador enviará las ordenes por puerto serie a *CoppeliaSim* para emular el movimiento. De forma contraria, si la variable `ROBOT_EMULATION` no está definida, el microcontrolador mandará las ordenes al prototipo real, que realizará el movimiento serpentino.

Además, se adjunta también en el Anexo B3.2 el código realizado en *CoppeliaSim* para emular el movimiento. El código consiste en la recepción de los datos enviados por puerto serie para posteriormente decodificarlos y poder utilizarlos para la emulación del movimiento.

Anexo B3.1: Código en Arduino IDE para el movimiento serpentino

```
#define PCA9685
#ifdef PCA9685
    #include <Adafruit_PWMServoDriver.h>
    Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
    #define MIN_PWM 130
    #define MAX_PWM 570
    #include <Servo.h>
    Servo servos[12]; //Max number of digital signals
#else
    #include <Servo.h>
    Servo servos[12]; //Max number of digital signals
#endif


//#define ROBOT_EMULATION

typedef struct{
    uint8_t pin;
    int offset;
    int min_pos;
    int max_pose;
}   RobotServo_t;


void writeServo(const RobotServo_t  &servo, int angle){
    #ifndef ROBOT_EMULATION
        #ifdef PCA9685
            int pulse_width;
            pulse_width = map(angle+servo.offset, 0, 180,MIN_PWM, MAX_PWM);
            pwm.setPWM(servo.pin,0,pulse_width);
            Serial.print(pulse_width);
            Serial.print("\n");
        #else
            servos[servo.pin].attach(servo.pin);
          servos[servo.pin].write(angle);
        #endif
```

```
    #else
        angle = constrain(angle+servo.offset,servo.min_pos,servo.max_pose);
        Serial.print("S");
        Serial.print(servo.pin);
        Serial.print(":");
        Serial.print((int)angle, DEC);
        Serial.print(";");

    #endif
    }


void detachServo(RobotServo_t &servo){
    #ifndef ROBOT_EMULATION
        servos[servo.pin].detach();
    #endif
}

#define JOINTS 8
#ifdef ROBOT_EMULATION
    RobotServo_t robotServos[JOINTS] = {{1, 0, -180, 180},   {2, 0, -180, 180},
                                        {3, 0, -180, 180},   {4, 0, -180, 180},
                                        {5, 0, -180, 180},   {6, 0, -180, 180},
                                        {7, 0, -180, 180},   {8, 0, -180, 180},};
#else
    RobotServo_t robotServos[JOINTS] = {{0, -30, 0, 180},    {5, -15, 0, 180},
                                        {6, -30, 0, 180},    {7, -15, 0, 180},
                                        {8, 20, 0, 180},     {9, -30, 0, 180},
                                        {10, -10, 0, 180},   {11, 10, 0, 180},};
#endif

int t0 = 0;
float n = 4.0;
int n_skip = 2;
float paso = 0.0004;
int i = 0;
int k = 0;
int indice_av = 1;
int indice_dir = 1;

float xtemp = 0.0;
float ytemp = 0.0;
float xtemp_dir = 0.0;
float ytemp_dir = 0.0;
float joint = 0;
float decision = 0;

//Parametros para el control del avance
float a = 0.6;
float b = 80.0;
float c = 0.0;
float s = 0.0;
float X = 0.0;
float Y = 0.0;
float Xant = 0.0;
float Yant = 0.0;
double pos_av[4] = {};
int cont = 0;

//Parametros para el control de la dirección
float a_dir = 0.0;
float b_dir = 0.0;
float c_dir = 0.0;
float s_dir = 0.0;
float X_dir = 0.0;
float Y_dir = 0.0;
float Xant_dir = 0.0;
float Yant_dir = 0.0;
```

```
double pos_dir[4] = {};
int tiempo = 0;
float tperiodo = 0.0;

double pos[JOINTS] = {90.0,90.0,90.0,90.0,90.0,90.0,90.0,90.0};
double  q1[JOINTS] = {90.0,90.0,90.0,90.0,90.0,90.0,90.0,90.0};



void setup(){
    tperiodo = 0.0;
    Serial.begin(115200);
    #ifndef ROBOT_EMULATION
        #ifdef PCA9685
            //Initializes the PCA9685 servo driver
            Wire.pins(0, 2);
            Wire.begin(0, 2);
            pwm.begin();
            pwm.setPWMFreq(50);
        #endif
    #endif
for(int i=0;i<JOINTS; i++) writeServo(robotServos[i], (int)q1[i]);
}



void loop(){
    X = 0.0;
    Y = 0.0;
    X_dir = 0.0;
    Y_dir = 0.0;

    for (i = 0; i < n; i++) {
        xtemp = 0.0;
        ytemp = 0.0;
        xtemp_dir = 0.0;
        ytemp_dir = 0.0;

        for (k = 1; k <= (i + 1); k++) {
            xtemp = xtemp+(1/n)*cos(a*cos((k+tperiodo)*b/n)+
                    c*(k+tperiodo)/n);
            ytemp = ytemp+(1/n)*sin(a*cos((k+tperiodo)*b/n)+
                    c*(k+tperiodo)/n);

            xtemp_dir = xtemp_dir+(1/n)*cos(a_dir*cos((k+tperiodo)*b_dir/n)+
                        c_dir*k+tperiodo)/n);

            ytemp_dir = ytemp_dir+(1/n)*sin(a_dir*cos((k+tperiodo)*b_dir/n)+
                        c*(k+tperiodo)/n);

        }

    Xant = X;
    Yant = Y;
    X = xtemp;
    Y = ytemp;

    Xant_dir = X_dir;
    Yant_dir = Y_dir;
    X_dir = xtemp_dir;
    Y_dir = ytemp_dir;

    pos_av[i]=atan2((Y-Yant),(X-Xant));
    pos_av[i]=pos_av[i]*(-180/3.14159);
    pos_dir[i] = atan2((Y_dir - Yant_dir), (X_dir - Xant_dir));
    pos_dir[i]=pos_dir[i]*(-180/3.14159);
    }
```

```
tperiodo += 0.1;

indice_av = 0;
indice_dir = 0;

    for (i = 0; i < JOINTS; i++){
        if (cont == 0) {
            pos[i] = 90+pos_av[indice_av];
            indice_av = indice_av + 1;
            cont = cont + 1;
        }
    else if (cont == 1) {
        pos[i] =90.0 + pos_dir[indice_dir];
        indice_dir = indice_dir + 1;
        cont = cont - 1;
        }
    }

for (int i = 0; i < JOINTS; i++) writeServo(robotServos[i], (int)pos[i]);
     delay(100);
}
```

Anexo B3.2: Código en CoppeliaSim para emulación del movimiento

```
function parseServoData(serial_data)
    local q={math.rad(90), math.rad(90), math.rad(90), math.rad(90),
    math.rad(90), math.rad(90), math.rad(90), math.rad(90)}

    local bytes, pin, angle
    while (string.len(serial_data)>0) do
        for i=1,8,1 do
            colon=string.find(serial_data, ':')
            semicolon=string.find(serial_data, ';')
            pin=tonumber(string.sub(serial_data, 2, colon-1))
            angle=tonumber(string.sub(serial_data, colon+1, semicolon-1))
            q[pin]=math.rad(angle)
            serial_data=string.sub(serial_data, semicolon+1)
        end
    end
    return q
end


function sysCall_init()
    -- Inicialización
    serial=sim.serialOpen("//./COM3", 115200)
    joints={}
    for i=1,8,1 do
        joints[i]=sim.getObjectHandle('articulacion'..i)
    end
    print(serial)
end


function sysCall_actuation()
    -- Código de actuación
    local bytes, serial_data
    bytes=sim.serialCheck(serial)
    while (bytes>0) do
        serial_data=sim.serialRead(serial,bytes,false,';')
        q=parseServoData(serial_data)
        print(serial_data)
        for i=1,#q,1 do
            sim.setJointTargetPosition(joints[i], q[i])
        end
        bytes=sim.serialCheck(serial)
    end
end


function sysCall_cleanup()
    -- Clean-up
    sim.serialClose(serial)
end
```

PRESUPUESTO

Índice del Presupuesto

1. INTRODUCCIÓN

En lo que sigue se presenta con todo nivel de detalle, el presupuesto correspondiente al del presente trabajo final de Máster. En caso de querer llevarse a cabo la ejecución del prototipo desarrollado, únicamente sería necesario considerar las unidades de obra UO.006 y UO.007, dejando a un lado toda las fases de documentación, diseño mecánico y electrónico, y validación del demostrador.

Del mismo modo, en caso de querer tomar como base el diseño del presente trabajo para desarrollar un robot apodo con un mayor número de módulos, únicamente sería necesario modificar la UO.006 en el Cuadro de Mediciones, incorporando un rendimiento de valor igual al número de módulos a incluir.

En las unidades de obra en las que se describen las fases de desarrollo del prototipo desde el ordenador -desarrolladas en el Cuadro de Precios Descompuestos-, se ha pretendido ejemplificar un "mes tipo" de trabajo, con el objetivo de luego, en el Cuadro de Mediciones, mostrar el tiempo real dedicado a partir de esta unidad de tiempo. Asimismo, en las unidades de obra en las que se lleva a cabo la selección de componentes mecánicos y electrónicos, se ha tratado de considerar -de forma simbólica- el asesoramiento ofrecido por parte de los distintos profesionales en las pertinentes materias.

Con respecto al software utilizado, se han introducido los costes anuales de las licencias de los distintos programas con modalidad de pago, con el objetivo de proporcionar un carácter más verídico al trabajo. No obstante, para el presente Trabajo Final de Máster, se han empleado licencias educacionales que no han supuesto coste alguno para su desarrollo.

En lo referente al Cuadro de Precios Descompuestos, en el presente presupuesto se añade un 2% en concepto de Costes Directos Complementarios, correspondientes a cada unidad de obra. Igualmente, en lo que respecta al Cuadro de Mediciones, se incorpora un 1,25% en concepto de Costes Indirectos. Por último, sobre el Presupuesto de Ejecución Material, se asume un 13% de Gastos generales y un 6% de Beneficio industrial.

2. CUADRO DE PRECIOS BÁSICOS

Código	Ud.	Descripción	Importe (€)
1		**Mano de Obra**	
MO.001	h	Ingeniera Técnica en Tecnologías Industriales	25,000
MO.002	h	Ingeniero Senior en Tecnologías Industriales	45,000
MO.003	h	Mecánico de Taller	10,000
MO.004	h	Ingeniera Técnica especializada en Robótica y Automática Industrial	30,000
MO.005	h	Técnico en Electrónica	15,000

3. CUADRO DE PRECIOS UNITARIOS

Código	Ud.	Descripción	Importe (€)
1		**Software**	
SW.001	año	Solidworks 2020 Professional License	13.200,000
SW.002	año	Matlab R2019b License	800,000
SW.003	año	Deep Learning Toolbox (version 13) para Matlab	0,000
SW.004	año	CoppeliaSim Professional License	1.600,000
SW.005	año	Open-source Arduino Software (IDE)	0,000
SW.006	año	Gestor de Tarjetas ESP8266 para Arduino	0,000
SW.007	año	Adafruit PWM Servo Driver Library para Arduino	0,000
2		**Herramientas para Montaje y Validación Experimental**	
HM.001	ud.	Set de destornilladores, STANLEY modelo STHT0-62113 (42 piezas)	20,990
HM.002	ud.	Fijador de roscas de media resistencia, Loctite modelo 243 (10 ml)	27,090
HM.003	ud.	Soldador de Estaño 60W 220V, modelo SREMTCH	16,990
HM.004	ud.	Bobina de estaño 0.6mm para soldadura electrónica, sin Plomo y con núcleo de Resina Sn99-0.7Cu	13,990
HM.005	ud.	Set de puntas para soldador de estaño 900M-T-I de cobre puro sin plomo reemplazable (10 piezas)	13,790
HM.006	ud.	Cargador Portátil para Li-Po 2-3S / 7.4-11.1V - modelo ZHITING B3 20W 1.6A P	16,990
HM.007	ud.	Multímetro Digital, modelo AoKoZo 21D	19,990

Código	Ud.	Descripción	Importe (€)
3		**Componentes Comerciales**	
CC.001	ud.	Servo Motor Hitec modelo HS-422	12,740
CC.002	ud.	Soporte en C (ref. ASB-09)	5,460
CC.003	ud.	Soporte en L (ref. ASB-06)	2,540
CC.004	ud.	Soporte de Servo Multiusos (ref. ASB-04)	5,060
CC.005	ud.	Perno M2 x 5 - D7985-H M2 x 5 (A2-70INOX)	0,160
CC.006	ud.	Tuerca M2 - D125-1A M2 (A4INOX)	0,120
CC.007	ud.	Arandela M2 - D934 M2 (A4-70INOX)	0,090
CC.008	ud.	Perno M2 x 5 - D7985-H M2 x 5 (4.8StZINN)	0,332
CC.009	ud.	Tuerca M2 - D934 M2 (CL8StZINC)	0,260
CC.010	ud.	Arandela M2 - D125-1A M2 (140HVStZINC)	0,450
CC.011	ud.	Tornillo M2,2 x 5 - D7981 CH 2,20 x 5 (StZINC)	0,870
CC.012	ud.	Tornillo M2,9 x 6,5 - D7981 CH 2,90 x 6,50 (StZINN)	1,160
CC.013	ud.	Batería Li-Po 5C 7,4V 2000mAh	13,190
CC.014	ud.	Batería Li-Po 8C 7,4V 4000mAh 2S	25,990
CC.015	ud.	Convertidor Regulador de Tensión 3.2-40V a 1.25-35V, modelo Yizhet 5 Piezas LM2596 DC a DC	10,690
CC.016	ud.	Microcontrolador ESP-01s con Programador USB	5,500
CC.017	ud.	Controladora de servos PCA9685	5,000
CC.018	ud.	Pack de cables protoboard hembra-hembra (20 cm), marca DuPont (40 uds.)	2,000
CC.019	ud.	Pack de cables protoboard macho-hembra (20 cm), marca DuPont (40 uds.)	2,000
CC.020	ud.	Pack de cables protoboard macho-macho (20 cm), marca DuPont (40 uds.)	2,000
CC.021	ud.	Protoboard de ensayos para circuitos electrónicos (12 uds.)	5,000

4. CUADRO DE PRECIOS DESCOMPUESTOS

UO.001 [mes] Documentación previa sobre locomoción ápoda

Documentación previa al diseño del prototipo demostrador, que incorpora toda la fase de análisis de la robótica móvil y las distintas tipologías de robots ápodos, así como el estudio de modelos matemáticos para la generación de trayectorias serpentinas.

Código	Ud.	Descripción	Rendim.	Precio unitario	Importe (€)
1.1		**Mano de Obra**			
MO.001	h	Ingeniera Técnica en Tecnologías Industriales	112,000	25,000	2.800,000
MO.002	h	Ingeniero Senior en Tecnologías Industriales	15,000	45,000	675,000
				Subtotal mano de obra:	**3.475,000**
1.2		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	3.475,000	69,500
				Costes directos:	**3.544,500**

UO.002 [mes] Diseño mecánico del prototipo

Diseño mecánico del prototipo, consistente en la selección de los actuadores y los brackets necesarios para la formación de un módulo

Código	Ud.	Descripción	Rendim.	Precio unitario	Importe (€)
2.1		**Mano de Obra**			
MO.001	h	Ingeniera Técnica en Tecnologías Industriales	112,000	25,000	2.800,000
MO.002	h	Ingeniero Senior en Tecnologías Industriales	15,000	45,000	675,000
MO.003	h	Mecánico de taller	4,000	10,000	40,000
				Subtotal mano de obra:	**3.515,000**
2.2		**Software**			
SW.001	año	Solidworks 2020 Professional License	0,083	13.200,000	1.100,000
SW.002	año	Matlab R2019b License	0,083	800,000	66,667
				Subtotal software:	**1.166,667**
2.3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	4.681,667	93,633
				Costes directos:	**4.775,300**

UO.003 [mes] Estudio de la configuración física optima

Simulación de las tres alternativas para la selección de la mejor configuración de locomoción del prototipo de serpiente ápoda, empleando el software específico para simulación dinámica de CoppeliaSim, a partir de un ensamblaje extraído de Solidworks

Código	Ud.	Descripción	Rendim.	Precio unitario	Importe (€)
3.1		**Mano de Obra**			
MO.004	h	Ingeniera Técnica especializada en Robótica y Automática Industrial	112,000	30,000	3.360,000
MO.002	h	Ingeniero Senior en Tecnologías Industriales	15,000	45,000	675,000
				Subtotal mano de obra:	**4.035,000**
3.2		**Software**			
SW.001	año	Solidworks 2020 Professional License	0,083	13.200,000	1.100,000
SW.002	año	Matlab R2019b License	0,083	800,000	66,667
SW.004	año	CoppeliaSim Professional License	0,083	1.600,000	133,333
				Subtotal software:	**1.300,000**
3.3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	5.335,000	106,700
				Costes directos:	**5.441,700**

UO.004 [mes] Desarrollo del sistema de control

Desarrollo de un sistema de control basado en una red neuronal, que predice los parámetros necesarios a partir de las especificaciones y requerimientos del usuario. Implementado en Matlab, incorporando librerías específicas de *Deep Learning*.

Código	Ud.	Descripción	Rendim.	Precio unitario	Importe (€)
4.1		**Mano de Obra**			
MO.004	h	Ingeniera Técnica especializada en Robótica y Automática Industrial	112,000	30,000	3.360,000
MO.002	h	Ingeniero Senior en Tecnologías Industriales	15,000	45,000	675,000
				Subtotal mano de obra:	**4.035,000**
4.2		**Software**			
SW.002	año	Matlab R2019b License	0,083	800,000	66,667
SW.003	año	Deep Learning Toolbox (version 13) para Matlab	0,083	0,000	0,000
				Subtotal software:	**66,667**
4.3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	4.101,667	82,033
				Costes directos:	**4.183,700**

UO.005 [mes] Diseño electrónico del prototipo

Diseño y selección de los componentes encargados del control y transmisión de órdenes del prototipo, principalmente compuesto por el microcontrolador, la placa controladora, y la batería. Cálculos preliminares con Matlab de potencia, coeficiente de descarga y otros parámetros. Asesoramiento técnico por parte de la empresa proveedora de componentes.

Código	Ud.	Descripción	Rendim.	Precio unitario	Importe (€)
5.1		**Mano de Obra**			
MO.003	h	Ingeniera Técnica especializada en Robótica y Automática Industrial	112,000	30,000	3.360,000
MO.002	h	Ingeniero Senior en Tecnologías Industriales	15,000	45,000	675,000
MO.005	h	Técnico en Electrónica	4,000	15,000	60,000
				Subtotal mano de obra:	**4.095,000**
5.2		**Software**			
SW.002	año	Matlab R2019b License	0,083	800,000	66,667
				Subtotal software:	**66,667**
5.3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	4.161,667	83,233
				Costes directos:	**4.244,900**

UO.006 [Ud.] Adquisición de herramientas para montaje y validación experimental

Adquisición de todas y cada una de las herramientas necesarias tanto para el montaje de los módulos individuales como del prototipo completo. Igualmente, queda incluida toda la instrumentación necesaria para llevar a cabo todas las operaciones necesarias durante el testing, incluidas las operaciones de soldadura.

Código	Ud.	Descripción	Rendim.	Precio unitario	Importe (€)
6.1		**Herramientas para montaje mecánico**			
HM.001	ud.	Set de destornilladores, STANLEY modelo STHT0-62113 (42 piezas)	1,000	20,990	20,990
HM.002	ud.	Fijador de roscas de media resistencia, Loctite modelo 243 (10 ml)	1,000	27,090	27,090
		Subtotal herramientas para montaje:			**48,080**
6.2		**Material para Soldadura**			
HM.003	ud.	Soldador de Estaño 60W 220V, modelo SREMTCH	1,000	12,740	12,740
HM.004	ud.	Bobina de estaño 0.6mm para soldadura electrónica, sin Plomo y con núcleo de Resina Sn99-0.7Cu	1,000	5,060	5,060
HM.005	ud.	Set de puntas para soldador de estaño 900M-T-I de cobre puro sin plomo reemplazable (10 piezas)	1,000	5,460	5,460
		Subtotal material para soldadura:			**23,260**
6.2		**Instrumentación para validación experimental**			
HM.006	ud.	Cargador Portátil para batería Li-Po 2-3S / 7.4-11.1V, modelo ZHITING B3 20W 1.6A P	1,000	12,740	12,740
HM.007	ud.	Multímetro Digital, modelo AoKoZo 21D	1,000	5,060	5,060
		Subtotal instrumentación para validación experimental:			**17,800**
6.3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	89,140	1,783
				Costes directos:	**90,923**

UO.007 [Ud.] Montaje de un módulo individual

Montaje de la unidad básica que compone el prototipo -un módulo-, consistente en un servo protegido por los *brackets* seleccionados, y unidos mediante tornillería específica.

Código	Ud.	Descripción	Rendim.	Precio unitario	Importe (€)
7.1		**Mano de Obra**			
MO.001	h	Ingeniera Técnica en Tecnologías Industriales	1,000	25,000	25,000
MO.003	h	Mecánico de Taller	5,000	10,000	50,000
				Subtotal mano de obra:	**75,000**
7.2		**Componentes principales**			
CC.001	ud.	Servo Motor Hitec modelo HS-422	1,000	12,740	12,740
CC.002	ud.	Soporte en C (ref. ASB-09)	1,000	5,460	5,460
CC.003	ud.	Soporte en L (ref. ASB-06)	1,000	2,540	2,540
CC.004	ud.	Soporte de Servo Multiusos (ref. ASB-04)	1,000	5,060	5,060
				Subtotal componentes principales:	**25,800**
7.3		**Tornillería *bracket* Multiusos - Servo**			
CC.005	ud.	Perno M2x5 - D7985-H M2x5 (A2-70INOX)	4,000	0,160	0,640
CC.006	ud.	Tuerca M2 - D125-1A M2 (A4INOX)	4,000	0,120	0,480
CC.007	ud.	Arandela M2 - D934 M2 (A4-70INOX)	4,000	0,090	0,360
				Subtotal tornillería *bracket* Multiusos - Servo:	**1,480**
7.4		**Tornillería *brackets* L - Multiusos**			
CC.008	ud.	Perno M2x5 - D7985-H M2x5 (4.8StZINN)	4,000	0,332	1,328
CC.009	ud.	Tuerca M2 - D934 M2 (CL8StZINC)	4,000	0,260	1,040
CC.010	ud.	Arandela M2 - D125-1A M2 (140HVStZINC)	4,000	0,450	1,800
				Subtotal tornillería *brackets* L - Multiusos:	**4,168**
7.5		**Tornillería *bracket* C - Servo**			
CC.011	ud.	Tornillo M2,2x5 - D7981 CH 2,20x5 (StZINC)	4,000	0,870	3,480
CC.012	ud.	Tornillo M2,9x6,5 - D7981 CH 2,90x6,50 (StZINN)	1,000	1,160	1,160
				Subtotal tornillería *bracket* C - Servo:	**4,640**
7.6		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	36,088	0,722
				Costes directos:	**36,810**

UO.008 [Ud.] Montaje de la serpiente completa

Montaje de la serpiente completa a partir del ensamblaje de todos y cada uno de los módulos individuales. Incorporación de la electrónica y del sistema de alimentación.

Código	Ud.	Descripción	Rendim.	Precio unitario	Importe (€)
8.1		**Mano de Obra**			
MO.001	h	Ingeniera Técnica en Tecnologías Industriales	1,000	25,000	25,000
MO.003	h	Mecánico de Taller	5,000	10,000	50,000
MO.005	h	Técnico en Electrónica	5,000	15,000	75,000
				Subtotal mano de obra:	**150,000**
8.2		**Tornillería de unión entre módulos**			
CC.008	ud.	Perno M2x5 - D7985-H M2x5 (4.8StZINN)	4,000	0,332	1,328
CC.009	ud.	Tuerca M2 - D934 M2 (CL8StZINC)	4,000	0,260	1,040
CC.010	ud.	Arandela M2 - D125-1A M2 (140HVStZINC)	4,000	0,450	1,800
				Subtotal tornillería de unión entre módulos:	**4,168**
8.3		**Componentes sistema de alimentación**			
CC.013	ud.	Batería Li-Po 5C 7,4V 2000mAh	1,000	13,190	13,190
CC.014	ud.	Batería Li-Po 8C 7,4V 4000mAh 2S	1,000	25,990	25,990
				Subtotal componentes sistema de alimentación:	**39,180**
8.4		**Componentes electrónica**			
CC.015	ud.	Convertidor Regulador de Tensión 3.2-40V a 1.25-35V, modelo Yizhet 5 Piezas LM2596 DC a DC	1,000	10,690	10,690
CC.016	ud.	Microcontrolador ESP-01s con programador USB	1,000	5,500	5,500
CC.017	ud.	Controladora de servos PCA9685	1,000	5,000	5,000
				Subtotal componentes electrónica:	**21,190**
8.5		**Componentes *protoboard***			
CC.018	ud.	Pack de cables protoboard hembra-hembra (20 cm), marca DuPont (40 uds)	1,000	2,000	2,000
CC.019	ud.	Pack de cables protoboard macho-hembra (20 cm), marca DuPont (40 uds)	1,000	2,000	2,000
CC.020	ud.	Pack de cables protoboard macho-macho (20 cm), marca DuPont (40 uds)	1,000	2,000	2,000
CC.021	ud.	Protoboard de ensayos para circuitos electrónicos (12 uds)	1,000	5,000	5,000
				Subtotal componentes *protoboard*:	**11,000**
8.6		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	225,538	4,511
				Costes directos:	**230,049**

UO.009 [mes] Validación experimental del prototipo

Validación experimental del prototipo completo, empleando Arduino como base para la ejecución del movimiento principal. Ensayo de movimientos rectilíneos y curvilíneos

Código	Ud.	Descripción	Rendim.	Precio unitario	Importe (€)
9.1		**Mano de Obra**			
MO.003	h	Ingeniera Técnica especializada en Robótica y Automática Industrial	112,000	30,000	3.360,000
MO.002	h	Ingeniero Senior en Tecnologías Industriales	15,000	45,000	675,000
				Subtotal mano de obra:	**4.035,000**
9.2		**Software**			
SW.002	año	Matlab R2019b License	0,083	800,000	66,667
SW.005	año	Open-source Arduino Software (IDE)	0,083	0,000	0,000
SW.006	año	Gestor de Tarjetas ESP8266 para Arduino	0,083	0,000	0,000
SW.007	año	Adafruit PWM Servo Driver Library para Arduino	0,083	0,000	0,000
				Subtotal software:	**66,667**
9.3		**Costes directos complementarios**			
	%	Costes directos complementarios	2,000	4.101,667	82,033
				Costes directos:	**4.183,700**

5. CUADRO DE MEDICIONES

Código	Ud.	Descripción	Rendim.	Precio unitario	Importe (€)
UO.001	mes	Documentación previa	0,500	3.544,500	1.772,250
UO.002	mes	Diseño mecánico del prototipo	0,500	4.775,300	2.387,650
UO.003	mes	Estudio de la configuración física optima	2,000	5.441,700	10.883,400
UO.004	mes	Desarrollo del sistema de control	1,000	4.183,700	4.183,700
UO.005	mes	Diseño electrónico del prototipo	0,500	4.244,900	2.122,450
UO.006	ud	Adquisición de herramientas para montaje y validación experimental	1,000	90,923	90,923
UO.007	ud	Montaje de un módulo individual	8,000	36,810	294,478
UO.008	ud	Montaje de la serpiente completa	1,000	230,049	230,049
UO.009	mes	Validación experimental del prototipo	1,500	4.183,700	6.275,550
		Costes Indirectos			
	%	Costes Indirectos	1,250	28.240,450	353,006
		Presupuesto de Ejecución Material (PEM):			**28.593,455**
		Gastos Generales			
	%	Gastos Generales	13,000	28.593,455	3.717,149
		Beneficio Industrial			
	%	Beneficio Industrial	6,000	28.593,455	1.715,607
		Presupuesto de Ejecución por Contrata (PEC):			**34.026,212**
		Impuesto Valor Añadido (I.V.A.)			
	%	Impuesto Valor Añadido (I.V.A.)	21,000	34.026,212	7.145,504
		PRESUPUESTO TOTAL DE INVERSIÓN:			**41.171,72**

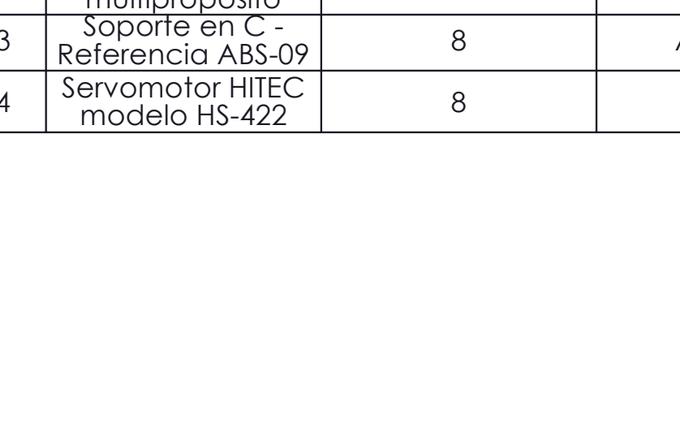
6. PRESUPUESTO

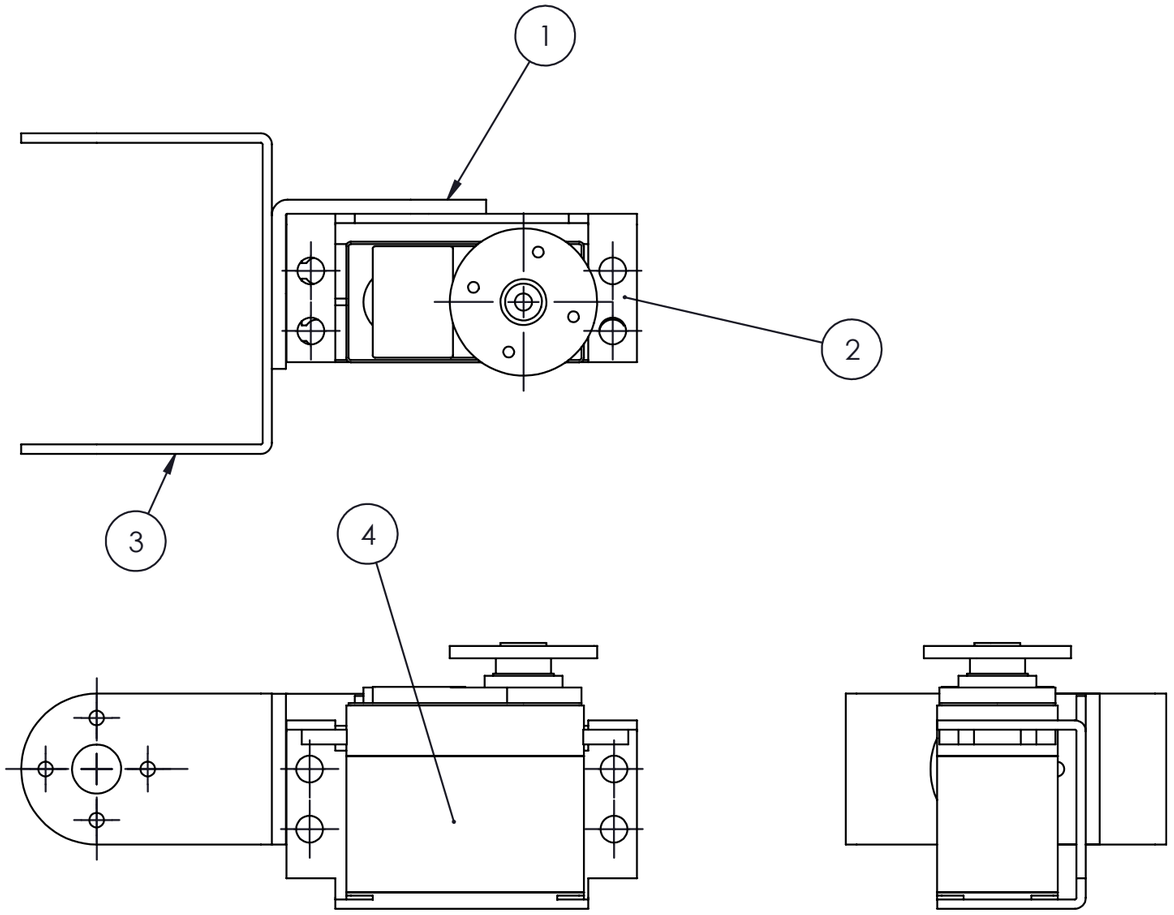
El Presupuesto Total de Inversión del presente Trabajo Final de Master, asciende a la cantidad de CUARENTA Y UN MIL CIENTO SETENTA Y UN euros con SETENTA Y DOS céntimos.

PLANOS

Índice de Planos

Nº	NOMBRE	CANTIDAD	MATERIAL
1	Soporte en L Referencia ABS-06	8	Aluminio
2	Sopoerte servo multipropósito	8	Aluminio
3	Soporte en C - Referencia ABS-09	8	Aluminio
4	Servomotor HITEC modelo HS-422	8	-





TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Proyecto: ESTUDIO Y DISEÑO DEL CONTROL DE MOVIMIENTOS DE UN ROBOT ÁPODO

Plano: P01.- Plano de Conjunto. Módulo Individual

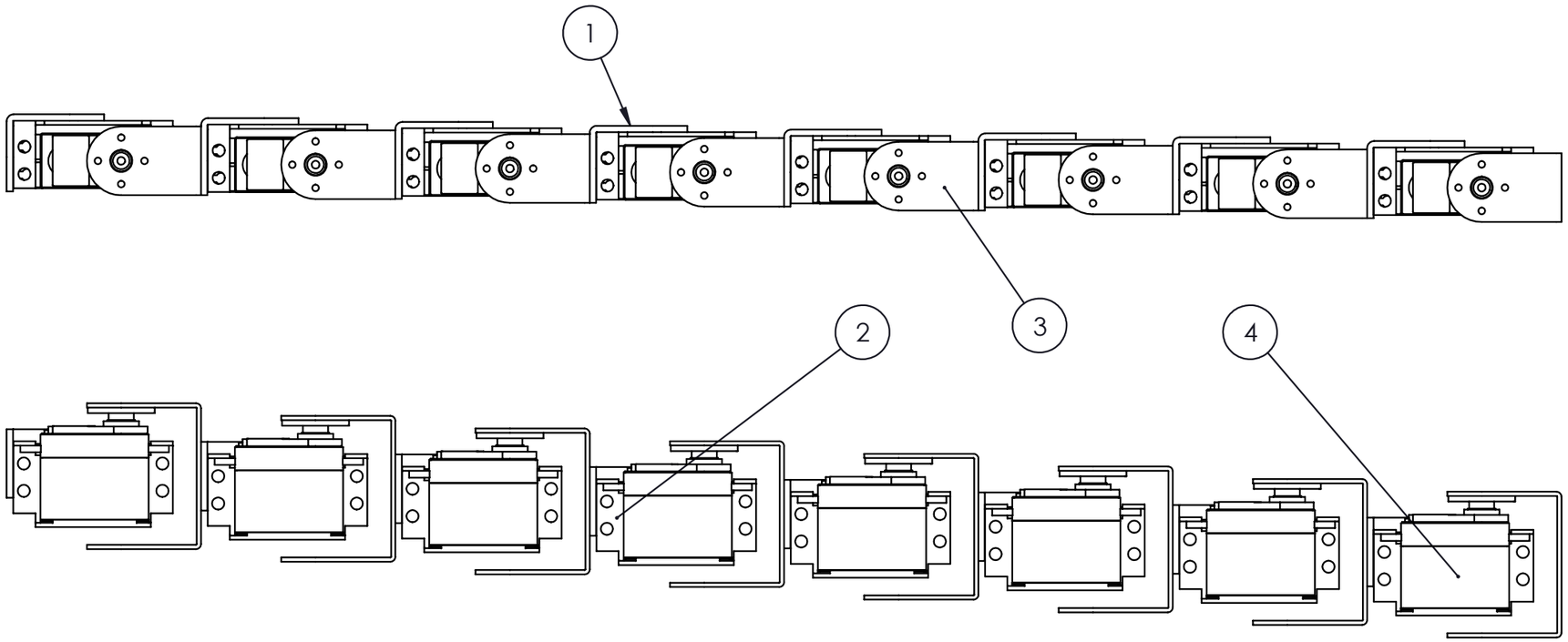
Autor: Cristina Asenjo Madrigal

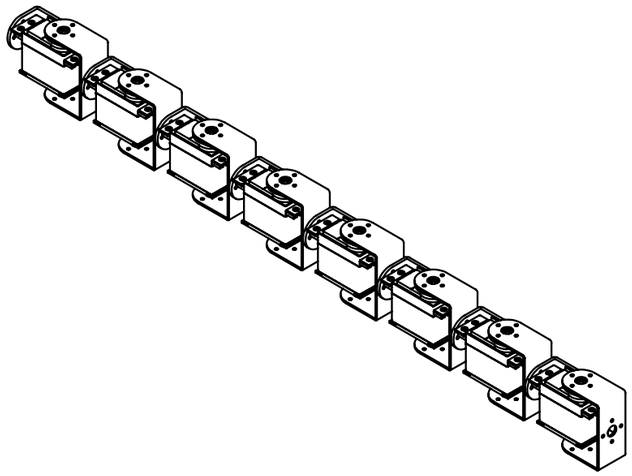
Fecha: Septiembre 2021

Escala: 1:1

Nº Plano: 1

Nº	NOMBRE	CANTIDAD	MATERIAL
1	Soporte en L Referencia ABS-06	8	Aluminio
2	Sopoerte servo multipropósito	8	Aluminio
3	Soporte en C - Referencia ABS-09	8	Aluminio
4	Servomotor HITEC modelo HS-422	8	-





TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Proyecto: ESTUDIO Y DISEÑO DEL CONTROL DE MOVIMIENTOS DE UN ROBOT ÁPODO

Plano: P02.- Plano de Conjunto. Configuracion Vertical/ Horizontal

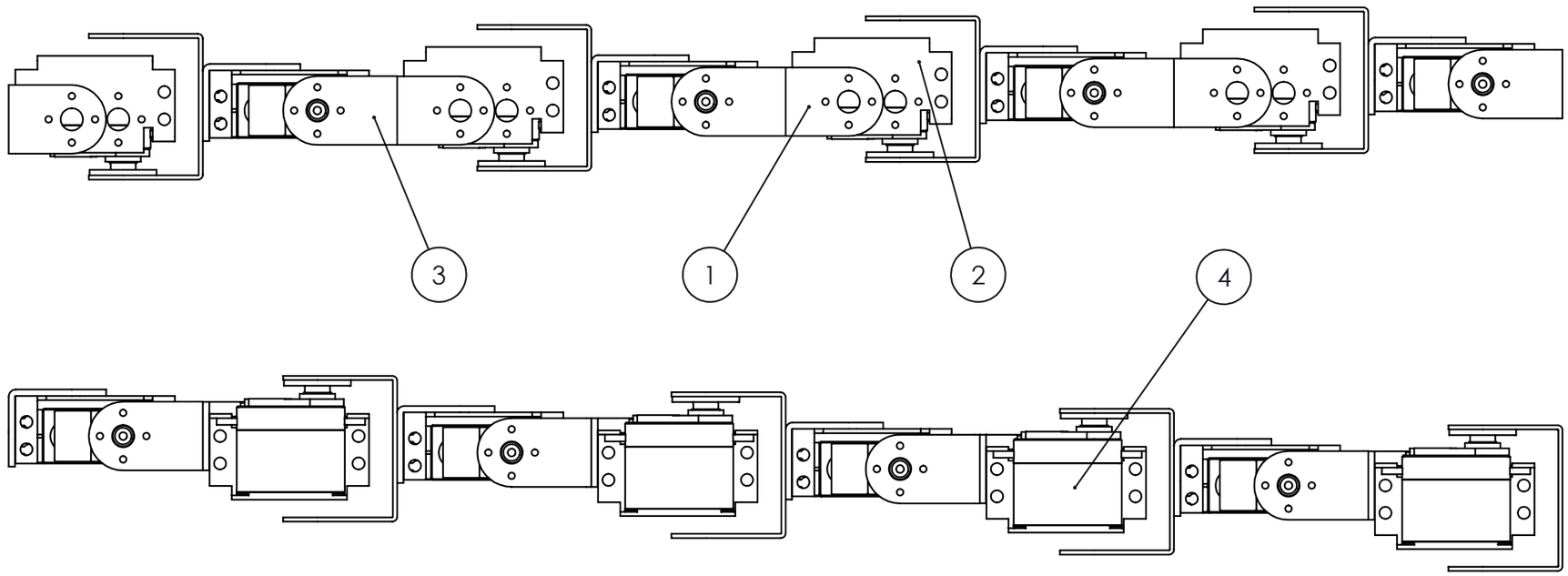
Autor: Cristina Asenjo Madrigal

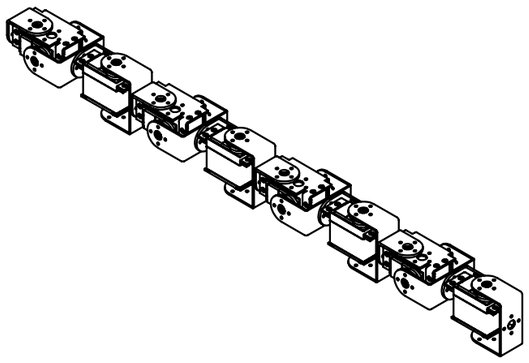
Fecha: Septiembre 2021

Escala: 1:2

Nº Plano: 2

Nº	NOMBRE	CANTIDAD	MATERIAL
1	Soporte en L Referencia ABS-06	8	Aluminio
2	Sopoerte servo multipropósito	8	Aluminio
3	Soporte en C - Referencia ABS-09	8	Aluminio
4	Servomotor HITEC modelo HS-422	8	-





TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Proyecto: ESTUDIO Y DISEÑO DEL CONTROL DE MOVIMIENTOS DE UN ROBOT ÁPODO

Plano: P03.- Plano de Conjunto. Configuracion Combinada

Autor: Cristina Asenjo Madrigal

Fecha: Septiembre 2021

Escala: 1:2

Nº Plano: 3