UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Simulación e implementación del robot cuadrúpedo Spot

Trabajo Fin de Máster

Máster Universitario en Ingeniería Mecatrónica

AUTOR/A: Camarena Morant, Diego Vicente

Tutor/a: Casanova Calvo, Vicente Fermín

CURSO ACADÉMICO: 2023/2024

AGRADECIMIENTOS

Quisiera agradecer a mi tutor por su confianza y tiempo durante la realización de este trabajo. A mis amigos, por su ayuda y apoyo durante tantos años, y a mi familia, especialmente a mi madre y hermana, por su cariño día a día. Sobre todo, a mi padre, quien me motivó a estudiar esta carrera y máster, y quien estaría muy orgulloso de este trabajo.

RESUMEN

El trabajo aborda el desarrollo de un modelo de la simulación del robot cuadrúpedo Spot de Boston Dynamics y la implementación de una versión simplificada del mismo. Empleando la librería de Matlab/Simulink, Simscape Multibody, se construirá un modelo de simulación lo más realista posible que permita analizar y comprender la relación entre el movimiento de las articulaciones y el desplazamiento del robot. Se centra en recrear con precisión su comportamiento, incluyendo dos formas de caminar distintas (crawl y trot), así como su capacidad para superar obstáculos como rampas y escalones. El objetivo principal es comprender el modo de actual del robot antes de su implementación física.

En la segunda parte del trabajo se llevará a cabo el montaje del robot utilizando un modelo simplificado, dada la complejidad del original, y se validarán experimentalmente las simulaciones previas. Para la implementación se imprimirán las piezas que componen el robot, se emplearán motores tipo servo para las articulaciones y un microcontrolador Arduino se encargará del control del movimiento, operado a distancia desde una aplicación Android.

RESUM

El treball aborda el desenvolupament d'un model de la simulació del robot quadrúpede Espot de Boston Dynamics i la implementació d'una versió simplificada d'este. Emprant la llibreria de Matlab/Simulink, Simscape multibodi es construirà un model de simulació el més realista possible que permeta analitzar i comprendre la relació entre el moviment de les articulacions i el desplaçament del robot. Se centra en recrear amb precisió el seu comportament, incloent dos maneres de caminar diferents (crawl i trot), així com la seua capacitat per a superar obstacles com a rampes i escalons. L'objectiu principal és comprendre el mode d'actual del robot abans de la seua implementació física.

En la segona part del treball es durà a terme el muntatge del robot utilitzant un model simplificat, donada la complexitat de l'original, i es validaran experimentalment les simulacions prèvies. Per a la implementació s'imprimiran les peces que componen el robot, s'empraran motors tipus servo per a les articulacions i un microcontrolador Arduino s'encarregarà del control del moviment, operat a distància des d'una aplicació Android.

ABSTRACT

This work deals with the development of a simulation model of the quadruped robot Spot from Boston Dynamics and the implementation of a simplified version of it. Using the Matlab/Simulink library, Simscape Multibody, a simulation model will be built that is as realistic as possible, allowing the relationship between the movement of the joints and the robot's displacement to be analysed and understood. The focus is on accurately recreating its behaviour, including two different ways of walking (crawl and trot), as well as its ability to overcome obstacles such as ramps and steps. The main objective is to understand the robot's current mode before its physical implementation.

In the second part of the work, the assembly of the robot will be carried out using a simplified model, given the complexity of the original one, and the previous simulations will be experimentally validated. For the implementation, the parts that make up the robot will be printed, servo motors will be used for the joints and an Arduino microcontroller will be in charge of controlling the movement, operated remotely from an Android application.

CONTENIDO DEL PROYECTO:

El trabajo está formado por los siguientes documentos:

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

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

SIMULACIÓN E IMPLEMENTACIÓN DEL ROBOT CUADRÚPEDO SPOT

1. MEMORIA

Trabajo Fin de Máster

Máster en Ingeniería Mecatrónica

Realizado por: Diego Vicente Camarena Morant

Tutorizado por: Vicente Casanova

Curso Académico: 2023/2024

Índice de la memoria

Índice de Figuras

Índice de Tablas

1. INTRODUCCIÓN

Los robots cuadrúpedos son sistemas robóticos diseñados para moverse sobre cuatro patas, imitando la locomoción de animales como perros, gatos y caballos. Este tipo de robots ha ganado popularidad en la robótica moderna debido a su versatilidad y capacidad para operar en terrenos irregulares donde los robots con ruedas tienen dificultades.

La estructura y dinámica de este tipo de robots permiten una mayor estabilidad y adaptabilidad, lo que los hace ideales para aplicaciones en entornos complejos, como operaciones de rescate, exploración, vigilancia y transporte de cargas. Además, su diseño les otorga una capacidad de movimiento más fluida, lo que les permite interactuar de manera más efectiva con el entorno y superar obstáculos de manera más eficiente [11].

Un ejemplo destacado de robots cuadrúpedos es el Spot de Boston Dynamics [5], conocido por su avanzada tecnología y versatilidad en aplicaciones industriales y de investigación, equipado con una variedad de sensores que le permiten mapear y navegar de manera automática en entornos complicados. Su estructura modular permite la integración de accesorios y herramientas, tales como brazos robóticos o dispositivos de inspección.

1.1. Spot

El vehículo original tiene las siguientes características [1] [6]:

- Spot mide aproximadamente 1.1 metros de largo, 0.84 metros de alto cuando está de pie y 0.6 metros de ancho.
- Pesa alrededor de 32.5 kilogramos (71.5 libras).
- Puede desplazarse a una velocidad máxima de 1.6 metros por segundo (aproximadamente 5.76 kilómetros por hora).
- Su batería le permite operar durante aproximadamente 1 hora y media con una sola carga, dependiendo de las tareas realizadas y de la carga útil.
- Spot es capaz de cargar hasta 14 kilogramos de peso adicional.
- Cada una de sus cuatro patas tiene tres grados de libertad, proporcionando un movimiento versátil y siendo capaz de adaptarse a terrenos irregulares.
- Está diseñado para operar en un amplio rango de temperaturas y condiciones climáticas, siendo resistente al polvo y agua (clasificación IP54).

Figura 1: Apariencia de Spot

https://alisysrobotics.com//es/blog/asi-es-spot-el-robot-cuadrupedo-mas-avanzando-del-mundo

El robot Spot de Boston Dynamics es un robot cuadrúpedo avanzado equipado con tecnología avanzada, diseñado para una enorme variedad de aplicaciones en entornos complejos. Posee cuatro patas motrices y un sistema de suspensión dinámica que le permite mantener la estabilidad, superando obstáculos y asegurando un contacto eficiente con el suelo, otorgando una alta fiabilidad, especialmente en situaciones de difícil operación y terrenos irregulares, permitiendo la captura y procesamiento de imágenes detalladas del entorno. Al disponer de un diseño modular es posible integrar accesorios y herramientas para tareas más específicas. La combinación de estas características hace de Spot una herramienta indispensable en entornos que precisen de robustez, flexibilidad y autonomía.

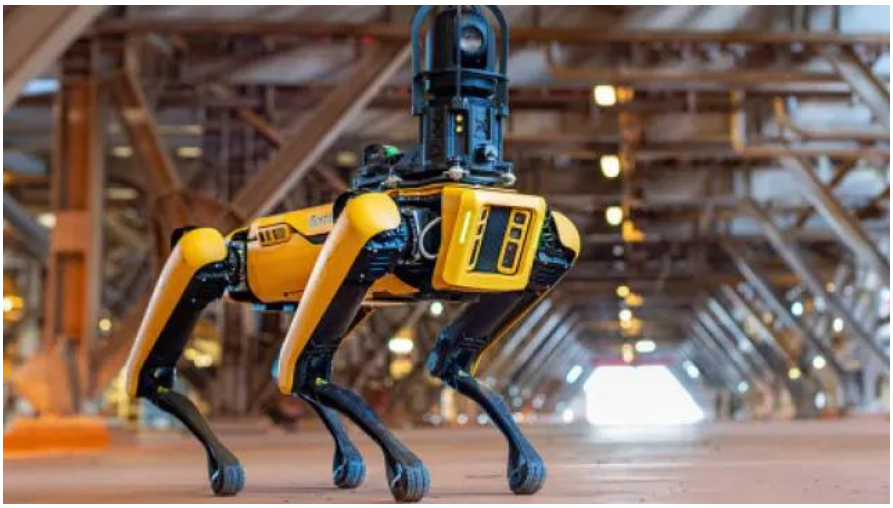


Figura 2: Spot con cámara incorporada

https://www.20minutos.es/tecnologia/moviles-dispositivos/el-perro-robot-de-boston-dynamics-se-actualiza-incluye-una-camara-a-color-y-se-comunica-desde-lugares-sin-cobertura-4994792/

La misión de Spot es proporcionar soluciones de automatización y asistencia en una gran variedad de entornos industriales, mejorando la eficiencia, seguridad y precisión en tareas que han sido desafiantes para los humanos o robots con ruedas. Entre sus misiones específicas se incluyen [7]:

1. Inspección industrial: Realización de inspecciones rutinarias en plantas de energía, instalaciones de fabricación y entornos industriales, siendo capaz de detectar cualquier anomalía, asegurando el correcto y continuo funcionamiento de equipos y estructuras.

2. Vigilancia y seguridad: Patrullar áreas extensas (almacenes o instalaciones de alta seguridad), proporcionando monitoreo constante, pudiendo actuar de forma rápida y eficiente.

3. Operaciones de rescate y emergencias: Acceder a áreas peligrosas o de difícil acceso, proporcionando información crítica, ayudando a localizar y rescatar en situaciones de emergencia.

4. Exploración y mapeo: Realizar el mapeo de terrenos peligrosos e irregulares, facilitando la planificación y ejecución de proyectos (construcción, exploración, minería…).

5. Agricultura y medio ambiente: Monitorear cultivos, recopilar datos ambientales y realizar tareas de mantenimiento en campos de cultivo o áreas de conservación natural.

Se podría decir que la misión principal de Spot es reducir riesgos para los humanos, aumentar la eficiencia operativa y proporcionar datos en tiempo real para tomas de decisiones.

1.2. Motivación y objetivos del proyecto

Una de las principales motivaciones para este proyecto es realizar una simulación del funcionamiento de un robot cuadrúpedo que ha ganado considerable renombre, diseñado y fabricado por la reconocida empresa Boston Dynamics. Esta compañía es pionera en el desarrollo de robots avanzados y ha revolucionado el mundo de la robótica con sus innovaciones y avances tecnológicos. El objetivo es entender y replicar el comportamiento del robot "Spot" en un entorno controlado, con el fin de analizar sus capacidades y optimizar su desempeño en diversas aplicaciones.

Otra motivación fundamental es diseñar y llevar a cabo una implementación simplificada de Spot, entendiendo las complicaciones inherentes a este proceso y aplicando los conocimientos adquiridos durante la simulación.

Además, esta implementación simplificada servirá como una plataforma educativa y experimental, ofreciendo una oportunidad valiosa para poner en práctica teorías y técnicas de robótica avanzada. La experiencia adquirida durante este proceso contribuirá al desarrollo de competencias técnicas y analíticas, esenciales para abordar desafíos futuros en el campo de la robótica. Por ende, este proyecto no solo se enfoca en la replicación y análisis de un robot avanzado, sino también en la creación de un entorno de aprendizaje integral que fomente la innovación y la mejora continua en el diseño y control de robots cuadrúpedos.

Para la realización del proyecto se empleará Simscape Multibody de Matlab para la simulación de Spot, SolidWorks para el diseño de las piezas del robot simplificado, y, finalmente, Arduino para programar el comportamiento de éste. Gran parte de los conocimientos para la realización de este trabajo han sido adquiridos a lo largo del Máster en Ingeniería Mecatrónica impartido en la Universidad Politécnica de Valencia, en materias como *Control Aplicado de Sistemas Mecatrónicos*, *Robótica* y *Sistemas de Medición y Actuación*.

1.3. Estructura del proyecto

El proyecto estará dividido en los siguientes apartados:

- Memoria
- Pliego de Condiciones
- Planos
- Presupuesto

En la Memoria se muestran los resultados obtenidos en las simulaciones realizadas, así como la implementación del robot, incluyendo su diseño, montaje y programación, además de los problemas que surgieron y soluciones aplicadas.

En el Pliego de Condiciones se detallará la normativa que debe seguir el proyecto, así como las condiciones de los materiales y de ejecución de este. También se explicará cómo realizar la prueba de servicio del robot.

En los planos se representan las partes diseñadas del robot con las vistas y medidas necesarias para su recreación, incluyendo también el despiece del ensamblaje del robot.

En el Presupuesto se calculará el valor final que tendrá el trabajo, incluyendo los materiales utilizados, la mano de obra y las licencias y equipos que se necesitan.

2. FUNDAMENTOS TEÓRICOS

El robot que se va a estudiar, Spot de Boston Dynamics, cuenta con una arquitectura modular que le permite realizar una gran cantidad de tareas en función del entorno de manera eficiente y adaptable.

Dispone de un diseño cuadrúpedo, imitando el movimiento de un animal. Cada pata tiene múltiples grados de libertad, permitiéndole realizar movimientos complejos como caminar, trotar y escalar.

Cada articulación está equipada con actuadores eléctricos que proporcionan la potencia necesaria para que el robot pueda moverse de forma dinámica, manteniendo el equilibrio en diferentes tipos de terreno.

Spot emplea dos patrones principales de movimiento para caminar: el crawl (gateo) y el trot (trote). A continuación, se describe brevemente el patrón de movimiento del patas necesario para realizar cada uno de estos movimientos.

1. Crawl (Gateo)

El gateo es un patrón de movimiento lento y estable, adecuado para terrenos difíciles y operaciones que requieren alta estabilidad. En este modo, las patas se mueven de manera secuencial, asegurando que al menos tres patas estén en contacto con el suelo en todo momento. Este patrón minimiza el riesgo de pérdida de equilibrio y permite al robot adaptarse a irregularidades del terreno [12].

Patrón de Movimiento:

1. Levantar la pata delantera izquierda.
2. Mover la pata delantera izquierda hacia adelante y bajar al suelo.
3. Levantar la pata trasera derecha.
4. Mover la pata trasera derecha hacia adelante y bajar al suelo.
5. Levantar la pata delantera derecha.
6. Mover la pata delantera derecha hacia adelante y bajar al suelo.
7. Levantar la pata trasera izquierda.
8. Mover la pata trasera izquierda hacia adelante y bajar al suelo.

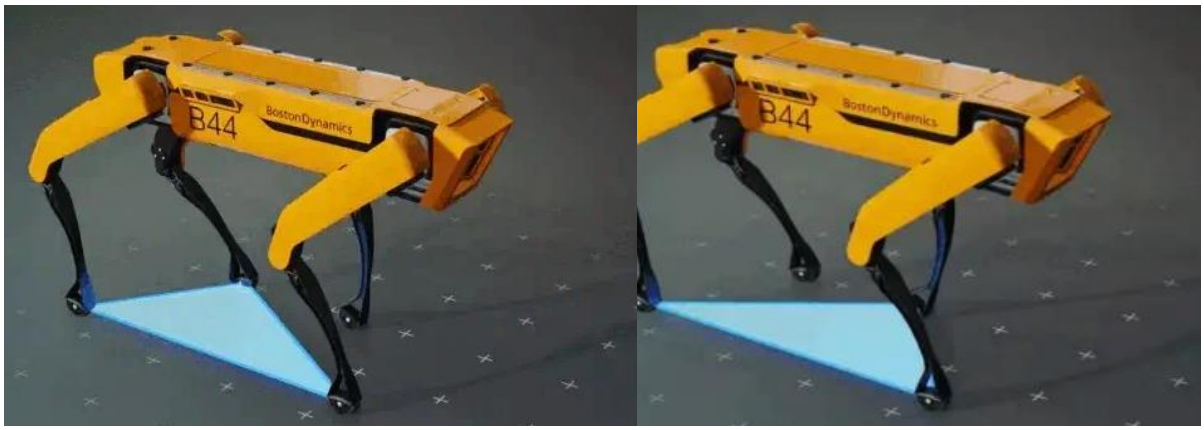


Figura 3: Patrón de movimiento durante el crawl.

Este ciclo se repite continuamente. La secuencia alternada garantiza que el robot tenga siempre un soporte triangular en el suelo, lo que maximiza la estabilidad.

2. Trot (Trote)

El trote es un patrón de movimiento más rápido y eficiente, adecuado para cubrir distancias mayores a una velocidad moderada. En este modo, las patas se mueven en pares diagonales, lo que significa que la pata delantera izquierda y la pata trasera derecha se mueven simultáneamente, seguidas por la pata delantera derecha y la pata trasera izquierda [12].

Patrón de Movimiento:

1. Levantar la pata delantera izquierda y la pata trasera derecha simultáneamente.
2. Mover ambas patas hacia adelante y bajar al suelo.
3. Levantar la pata delantera derecha y la pata trasera izquierda simultáneamente.
4. Mover ambas patas hacia adelante y bajar al suelo.

Figura 4: Patrón de movimiento durante el trot.

Este ciclo se repite continuamente. El trote permite que el robot se desplace más rápidamente que el gateo, aunque con menos puntos de contacto con el suelo en cada instante, lo que requiere un mejor control dinámico para mantener el equilibrio.

3. SIMULACIONES Y EXPERIMENTACIÓN

Se han realizado diversas simulaciones para poder comprobar cómo sería el comportamiento del Spot en un pavimento firme con las siguientes características: rigidez de 10000 N/m, amortiguamiento de 100 N/(m/s), coeficiente de fricción estática de 0.5 y coeficiente de fricción dinámica de 0.3 [8].

Las simulaciones consisten, en primer lugar, en realizar movimientos básicos, como que el robot se sentara o que se agachara. También se estudiarán las dos formas que tiene Spot de caminar, trot, la forma más rápida y estable que tiene de caminar, donde mueve las patas diagonales en pares, proporcionando un equilibrio dinámico y desplazarse rápidamente sobre terrenos varios, y crawl, una forma de caminar más lenta y controlada, donde se mueve una pata a la vez, ideal cuando se requiere una precisión extrema en el movimiento. Además, se observará cómo sortea diversos obstáculos tales como escalones, baches y rampas. Finalmente, se ha simulado a Spot caminando de lado.

Para poder realzar todo esto se ha utilizado el entorno de simulación multicuerpo para sistemas tridimensionales Simscape Multibody de Matlab donde, mediante bloques de fuerzas de contacto espaciales para modelar las interacciones entre las patas de Spot y la superficie sobre la que se desplaza. Estos bloques permiten aplicar las fuerzas normales y de fricción entre las geometrías conectadas, siendo fundamental para simular el movimiento realista del robot [10][17].

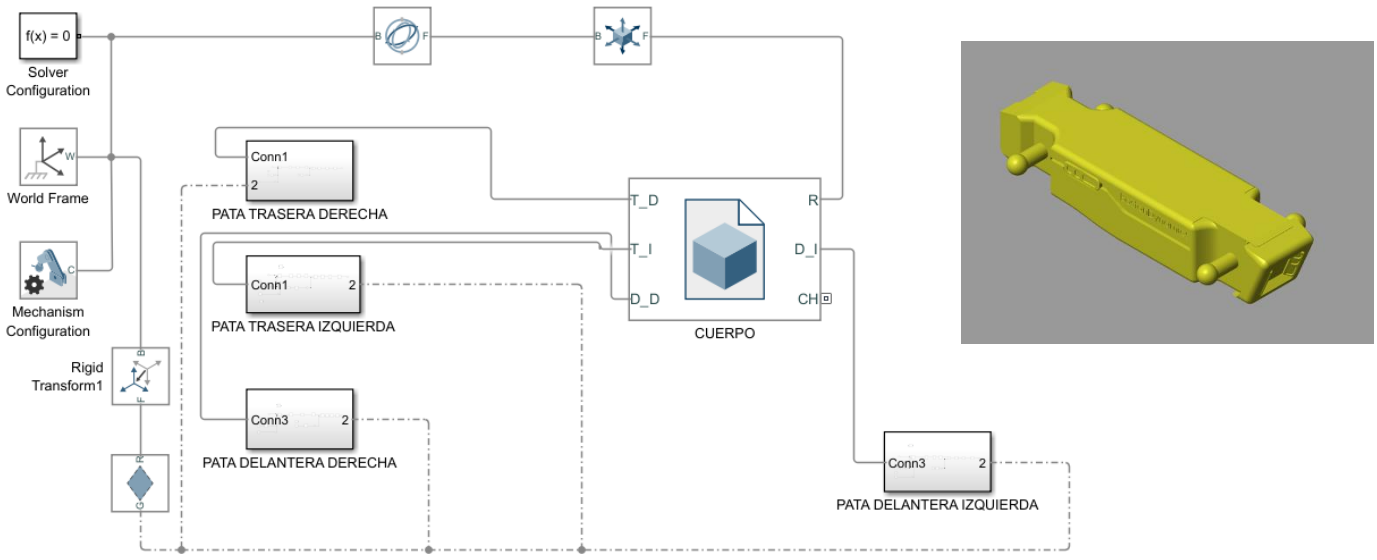


Figura 5: Modelo de Simulación de Spot

Como puede observarse, el modelo de simulación de la Figura 5: Modelo de Simulación de Spot está formado por el cuerpo y cuatro subsistemas, uno por cada pata del robot, las cuales se unen al cuerpo mediante una junta universal, otorgando los grados de libertad necesarios para el movimiento de éstas.

También se observa un plano infinito, representando el suelo, que hará contacto con cada pata mediante fuerzas de contacto, modelando las interacciones físicas entre los dos sólidos en contacto, calculando la fuerza de contacto normal y de fricción en función de los materiales y las características del suelo.

Finalmente se han añadido los bloques que agregan los grados de libertad rotacionales y transnacionales a Spot, siendo éstos el 'Gimbal Joint' y 'Cartesian Joint', respectivamente.

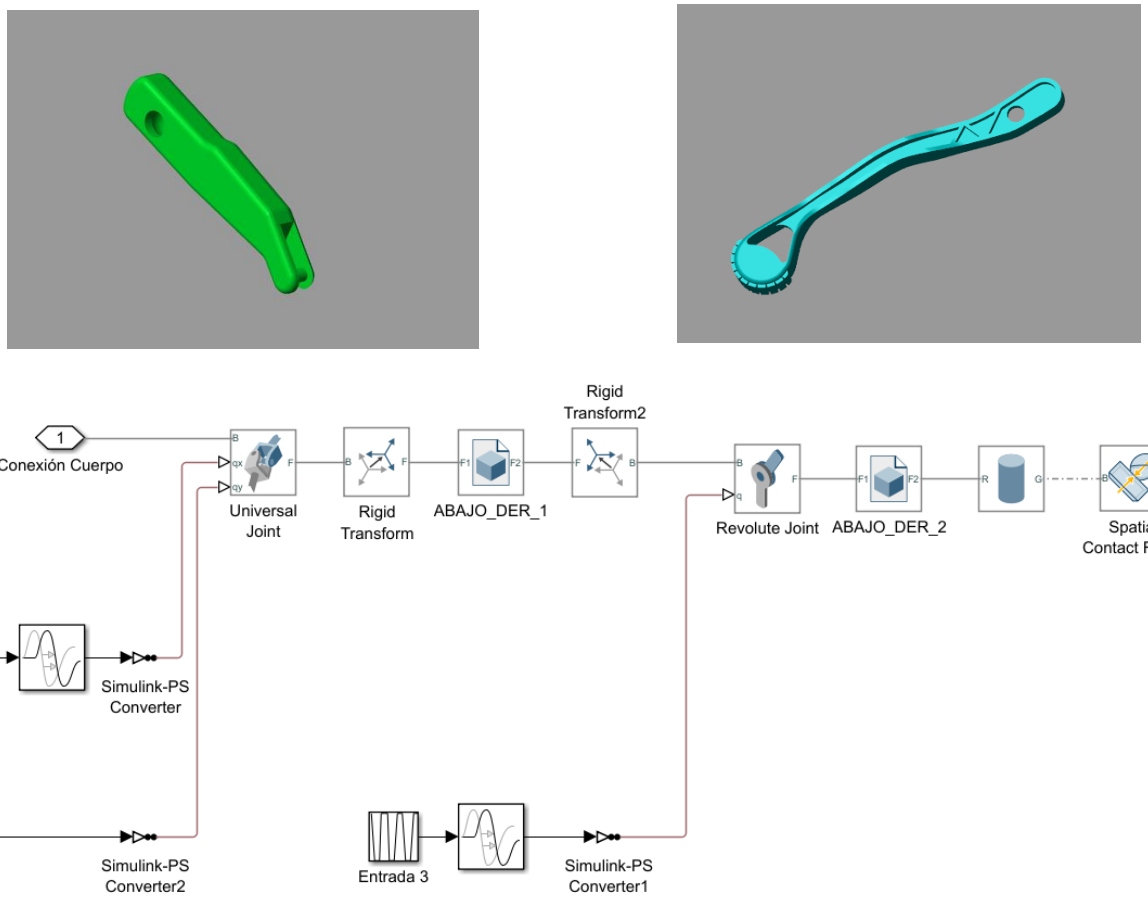


Figura 6: Modelado de las patas

En la Figura 6 se muestra el modelado de las patas, específicamente la pata trasera derecha, si bien el proceso es idéntico para las cuatro extremidades. Se pueden identificar los bloques de la junta universal y el de contacto con el suelo, mencionados anteriormente. También se encuentra presente una junta de revolución que especifica el movimiento de la articulación que conecta la parte superior e inferior de la pata, similar a la articulación de una rodilla. A las juntas le pasaremos los datos de entrada, es decir, la secuencia de ángulos que realizarán las articulaciones para el movimiento del robot.

Hay que comentar que la conexión entre la pata y el suelo es a través de un cilindro invisible que está situado en la parte redondeada del final de la pata. Esto es por precaución para que el contacto sea lo más exacto posible debido a que, a la hora de exportar la geometría de la pieza, es probable que se modifique su forma, pues el programa trata de simplificarla.

En la Figura 6 se puede observar cómo queda el robot en la simulación en posición de reposo:

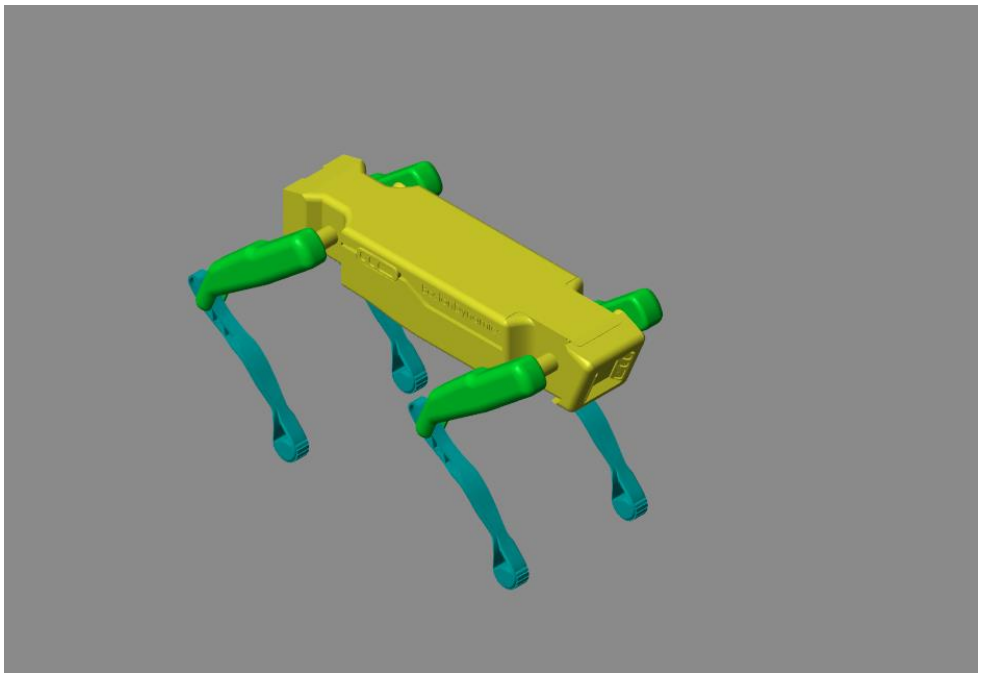


Figura 7: Robot simulado

A continuación, se explicarán cada una de las simulaciones que se han hecho para comprender el comportamiento de Spot mostrando, en cada simulación gráficas de interés en función del movimiento del robot.

3.1. El robot se sienta

En esta primera simulación se trató deber si el robot es capaz de sentarse, un movimiento sencillo para comprobar la estabilidad de éste.

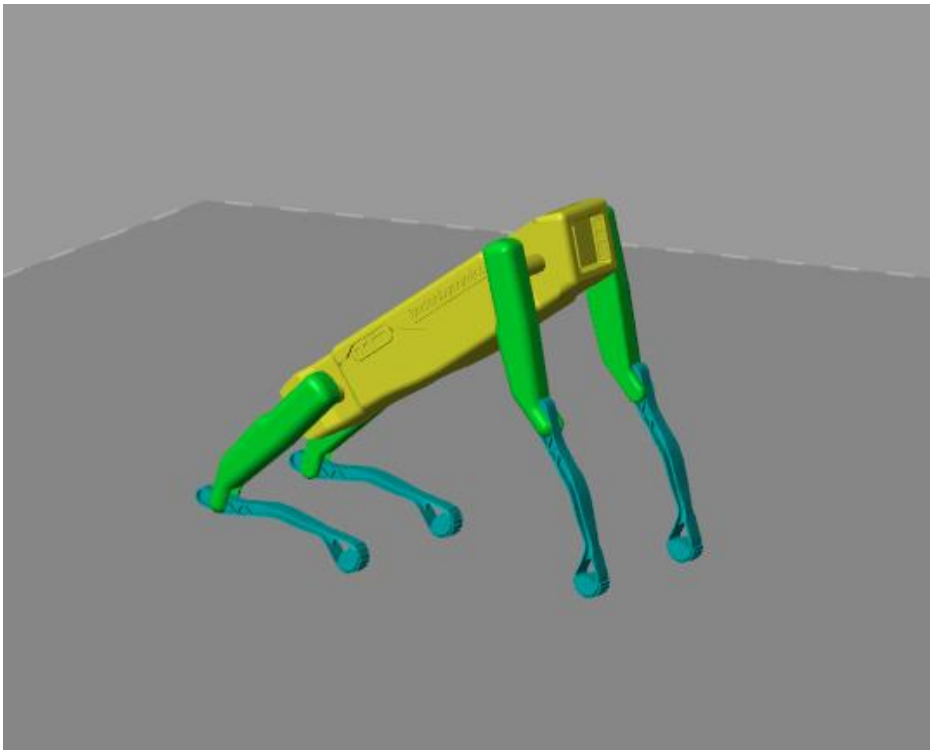


Figura 8: Primera simulación: el robot se sienta

En este caso, el robot no avanza ni se balancea, por lo que el único parámetro interesante es su ángulo de cabeceo. En la Figura 9 puede observarse su evolución a lo largo del tiempo:

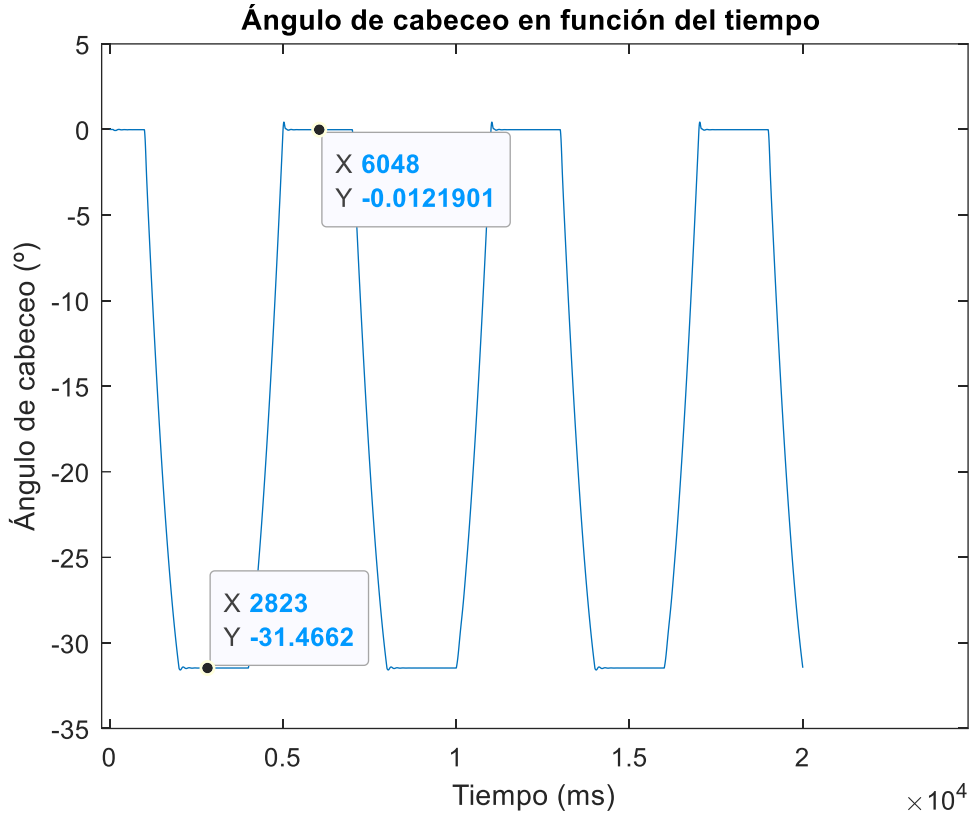


Figura 9: Ángulo de cabeceo del primer experimento

Se trata de un movimiento repetitivo en el que el robot se sienta y se levanta en varias ocasiones. Se aprecia que, cuando está sentado, el ángulo de cabeceo es ligeramente superior a 30º.

3.2. El robot baja

Esta segunda simulación vuelve a ser un movimiento sencillo en el que el robot baja el cuerpo doblando las patas.

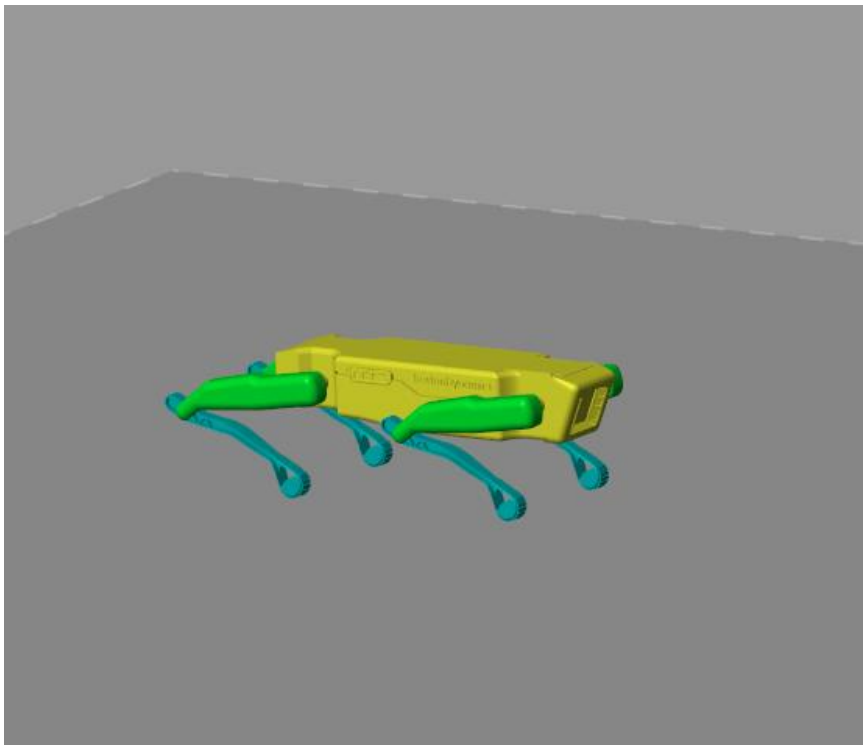


Figura 10: Segunda simulación: el robot baja

En este caso el parámetro en el que nos centraremos será la altura a la que se encuentra el cuerpo del robot, es decir, su posición respecto al eje Z. En la Figura 11 se puede observar esta evolución en función del tiempo.

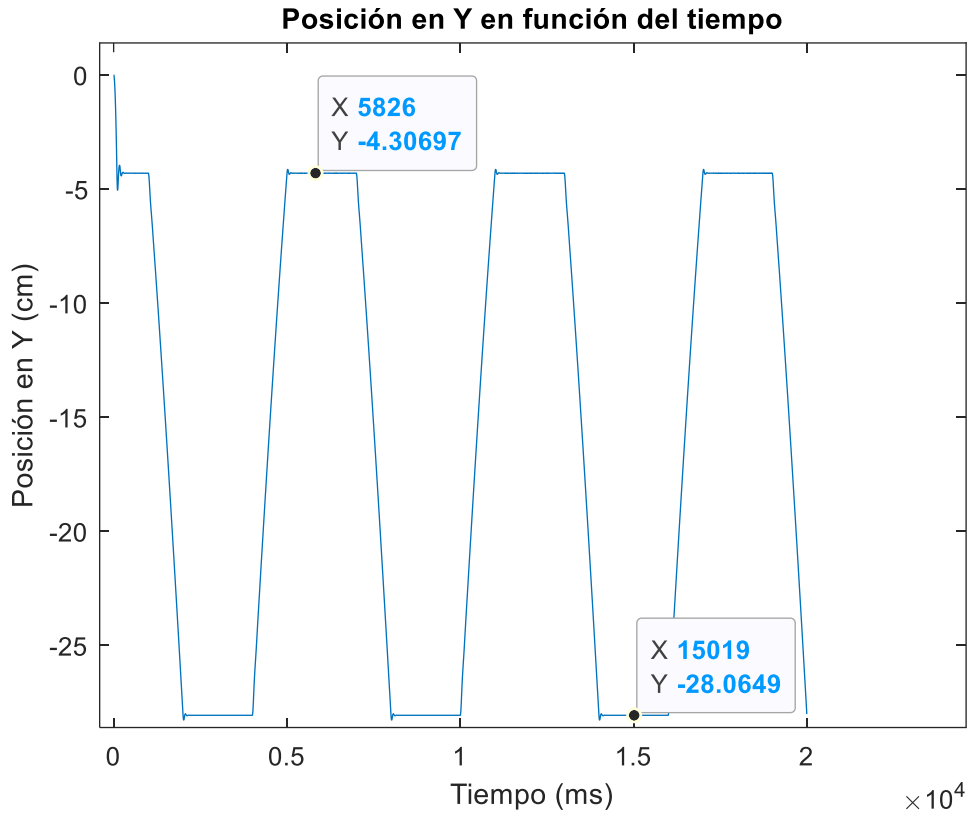


Figura 11: Posición del cuerpo en la segunda simulación en función del tiempo

Se trata de un movimiento secuencial en el que el robot baja el cuerpo unos 28cm con respecto a la posición inicial y luego sube hasta cerca de 4cm por debajo del momento inicial.

3.3. Trot

Como se ha comentado, trot es la forma más rápida y estable que tiene el robot de caminar, moviendo las patas diagonales en pares. En la Figura 12 puede observarse claramente la forma que tiene de avanzar de forma que, mientras dos patas están elevadas, las otras dos impulsan al robot.

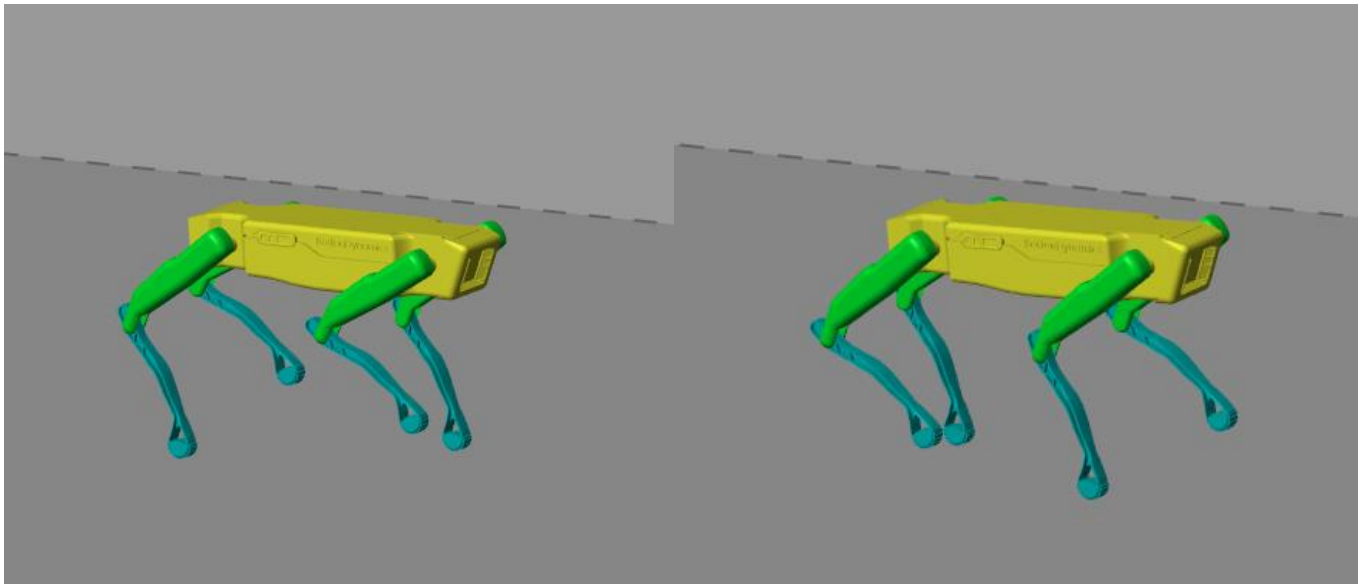


Figura 12: Tercera simulación: trot

En primer lugar, se va a observar la evolución del ángulo de balanceo en función de la posición en X del robot (su avance). También se podrá observar cuántos metros es capaz de avanzar durante los 20 segundos que dura la simulación.

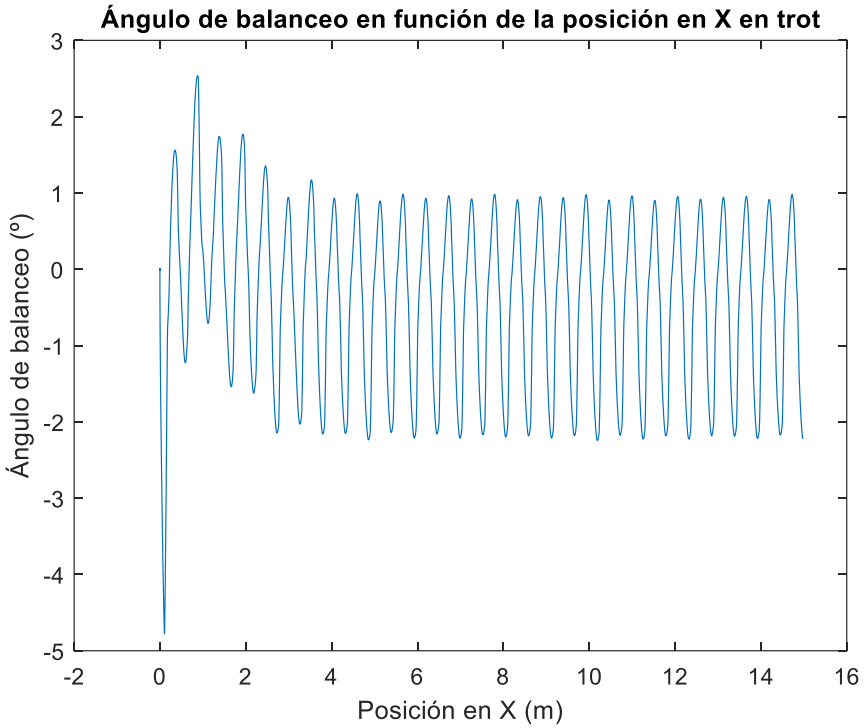


Figura 13: Evolución del ángulo de balanceo durante el avance en trot

Se aprecia que al principio el ángulo de balanceo es mayor debido a que sale del reposo, pero una vez está en movimiento se mantiene muy pequeño entre 1 o -2 grados. Además, durante los 20 segundos de simulación el robot avanza 15 metros.

Se hace lo mismo en la Figura 14 teniendo en cuenta esta vez el ángulo de cabeceo.

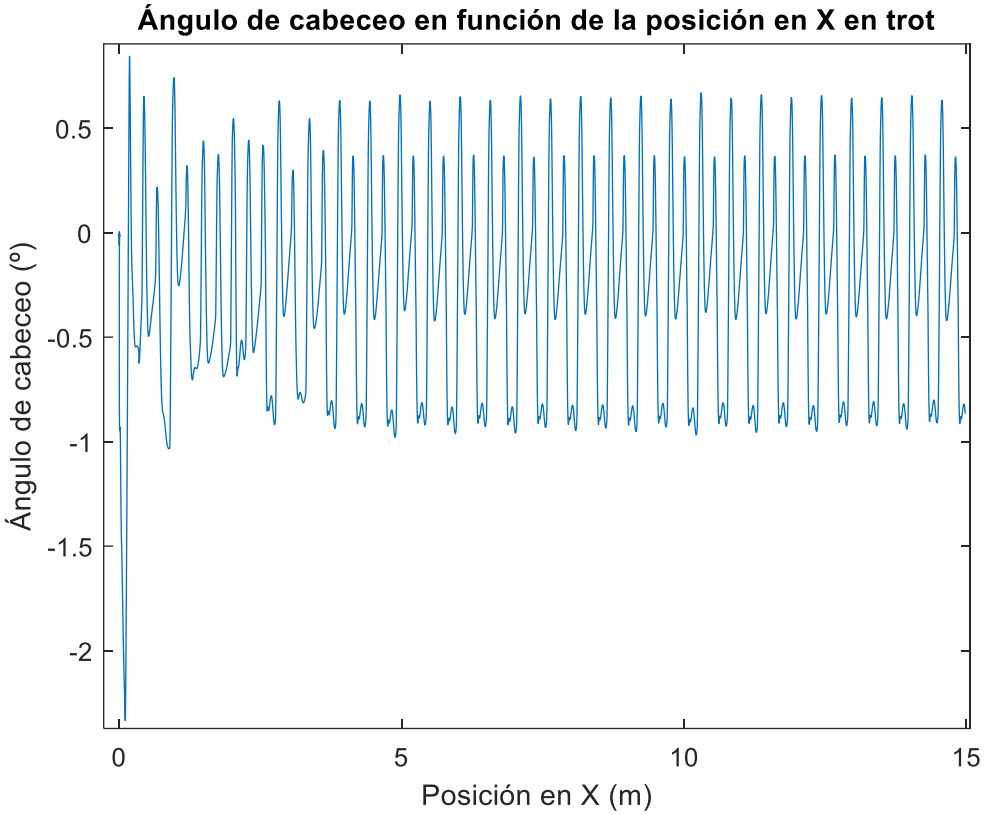


Figura 14: Evolución del ángulo de cabeceo durante en avance en trot

Como se observa, el ángulo de cabeceo es también muy pequeño a lo largo de los 15 metros que se recorren, lo cual quiere decir que el cuerpo se mantiene bastante estable mientras camina en el modo trot.

También es interesante observar cómo evolucionan el ángulo de balanceo y de cabeceo en función de la altura a la que se encuentre el robot (posición con respecto al eje Z). En la Figura 15 y la Figura 16, respectivamente, se grafican estas relaciones.

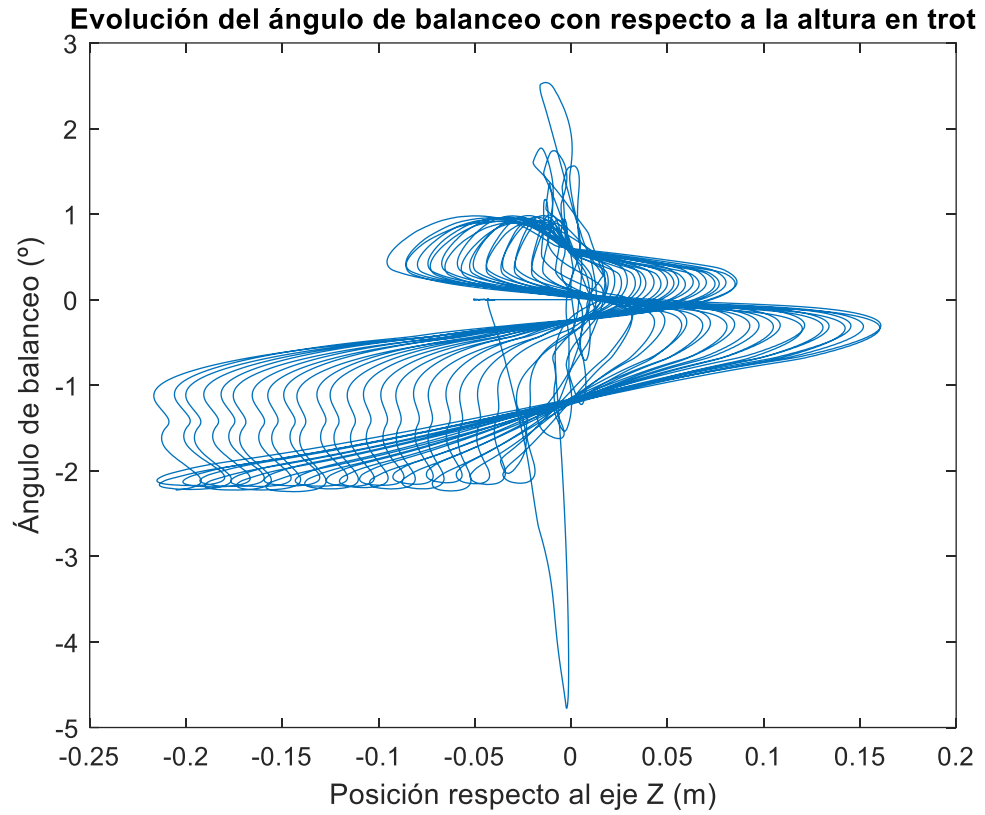


Figura 15: Evolución del ángulo de balanceo en función de la altura del robot funcionando en trot

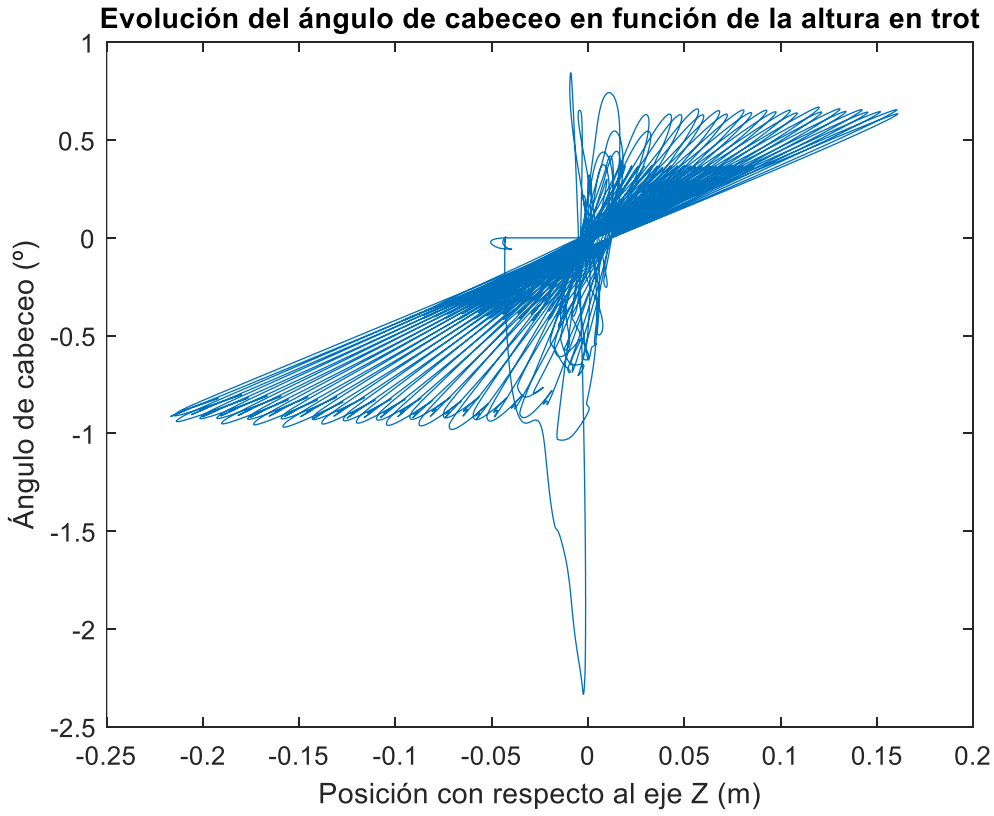


Figura 16: Evolución del ángulo de cabeceo en función de la altura del robot funcionando en trot

Como puede observarse, la variación de la altura del robot oscila hasta 15 y -20 centímetros como máximo. Este aumento gradual de la posición vertical podría ser interpretada como el resultado acumulativo de pequeñas fluctuaciones causadas por el levantamiento de las patas que, aunque se mueven verticalmente, el cuerpo del robot no se eleva ni desciende sustancialmente, manteniendo una posición general.

En cuanto al ángulo de balanceo, se observa que es muy pequeño, oscilando entre 1° y -2°, mientras que el ángulo de cabeceo varía entre 0.6° y -1°. Esto sugiere que, durante el avance del robot en modo trot, mantiene una posición del cuerpo bastante estable, con mínimas variaciones en su posición vertical y en los ángulos de balanceo y cabeceo.

3.4. Crawl

La segunda forma de avanzar es el Crawl, donde se mueve una pata a la vez, siendo un movimiento más lento, pero a su vez más controlado. Se puede apreciar de mejor forma la secuencia de las patas en la Figura 17.

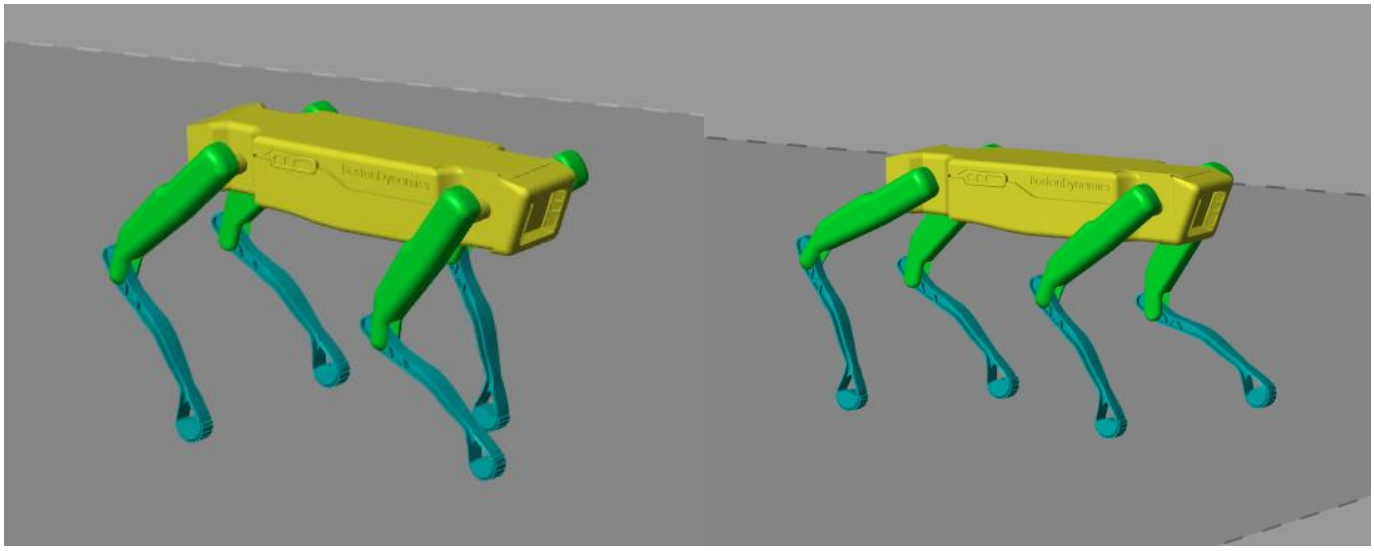


Figura 17: Cuarta simulación: crawl

Al igual que en la simulación anterior, se empezará observando la evolución del ángulo de balanceo en función de la posición en horizontal del robot (su avance).

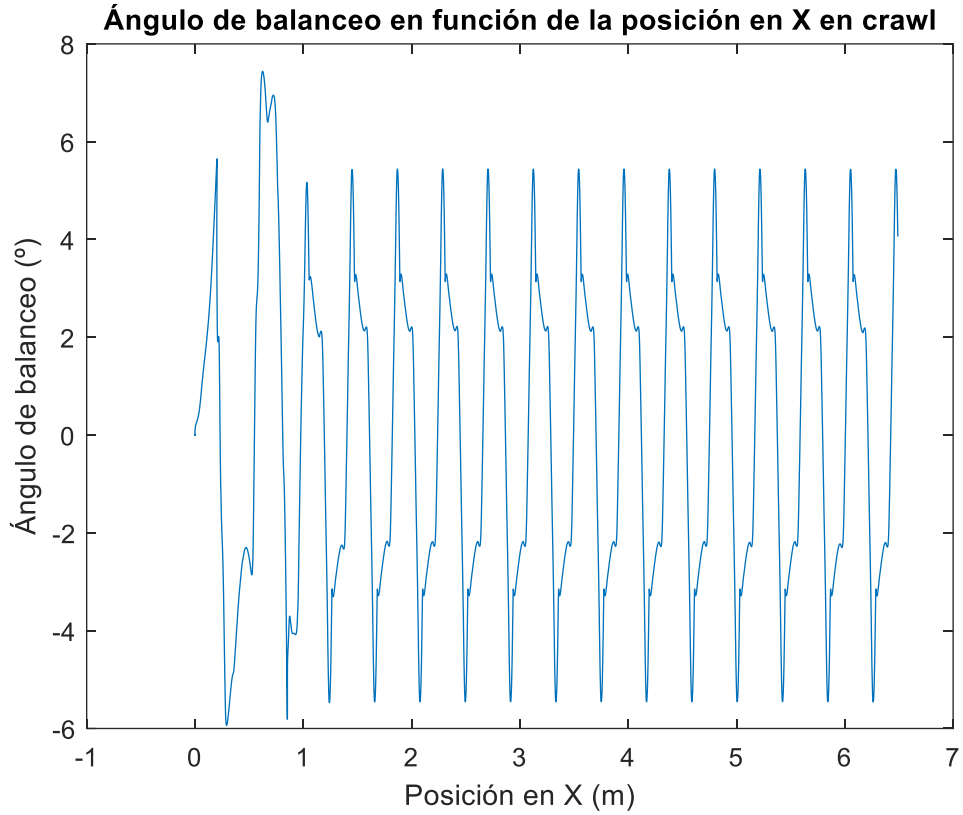


Figura 18: Evolución del ángulo de balanceo en función del avance funcionando en crawl

La primera diferencia del crawl que podemos observar en la Figura 18 es que el robot avanza bastante menos con respecto al trot, poco más de 6 metros, por lo que se deduce que esta forma de funcionamiento tiene una velocidad significativamente menor. Este modo es más adecuado para situaciones donde la velocidad no es una prioridad.

Fijándonos en el ángulo de cabeceo, sigue siendo pequeño, aunque superior, oscilando entre los 5º y los -6º.

En la Figura 19 se realiza el mismo estudio, pero comparado con el ángulo de cabeceo, el cual oscila entre 1.5º y -0.2º

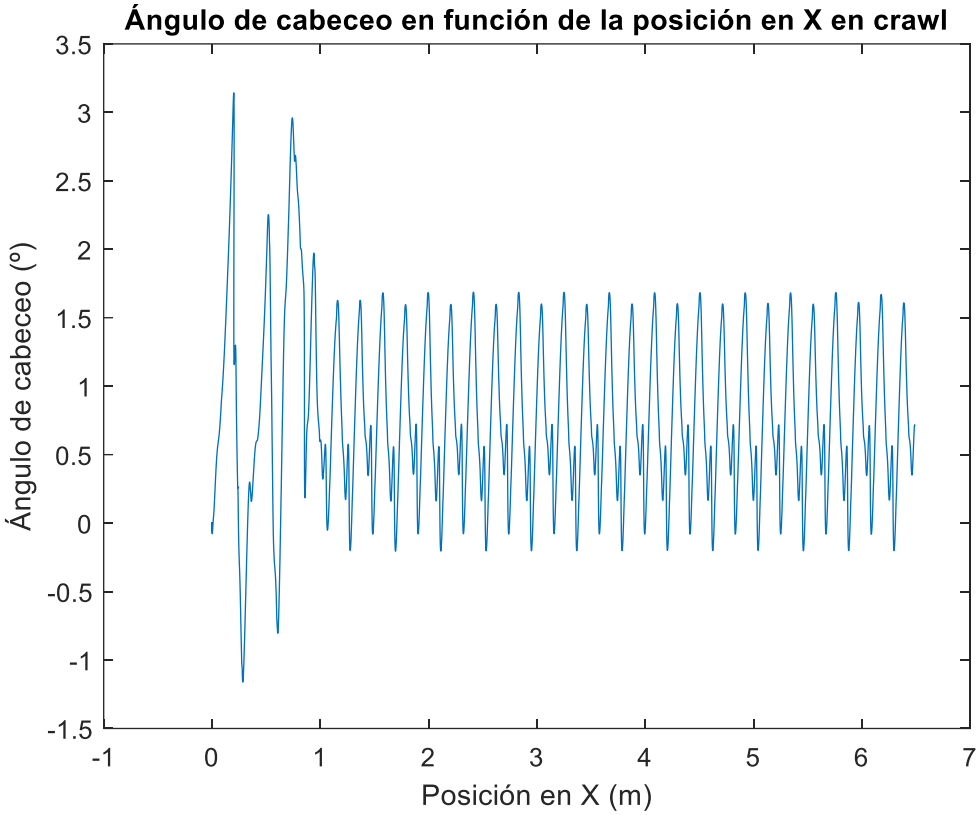


Figura 19: Evolución del ángulo de cabeceo en función del avance funcionando en crawl.

En general, los ángulos de cabeceo y de balanceo son mayores cuando funciona en crawl que en trot, lo que sugiere una menor estabilidad. Sin embargo, el movimiento de una pata a la vez permite un control más preciso del posicionamiento, lo que es ideal para movimientos que requieran una precisión extrema.

Para finalizar, en las Figura 20 y Figura 21 se muestra, respectivamente, la evolución del ángulo de balanceo y de cabeceo en función de la posición vertical del robot, la cual va aumentando gradualmente hasta alcanzar los 18 cm y -2 cm. Este aumento ocurre por lo mismo que en trot, debido al levantamiento de las patas ya que, como se ha demostrado, los ángulos de cabeceo y balanceo son bastante pequeños, por lo que el cuerpo se mantiene bastante fijo durante el avance.

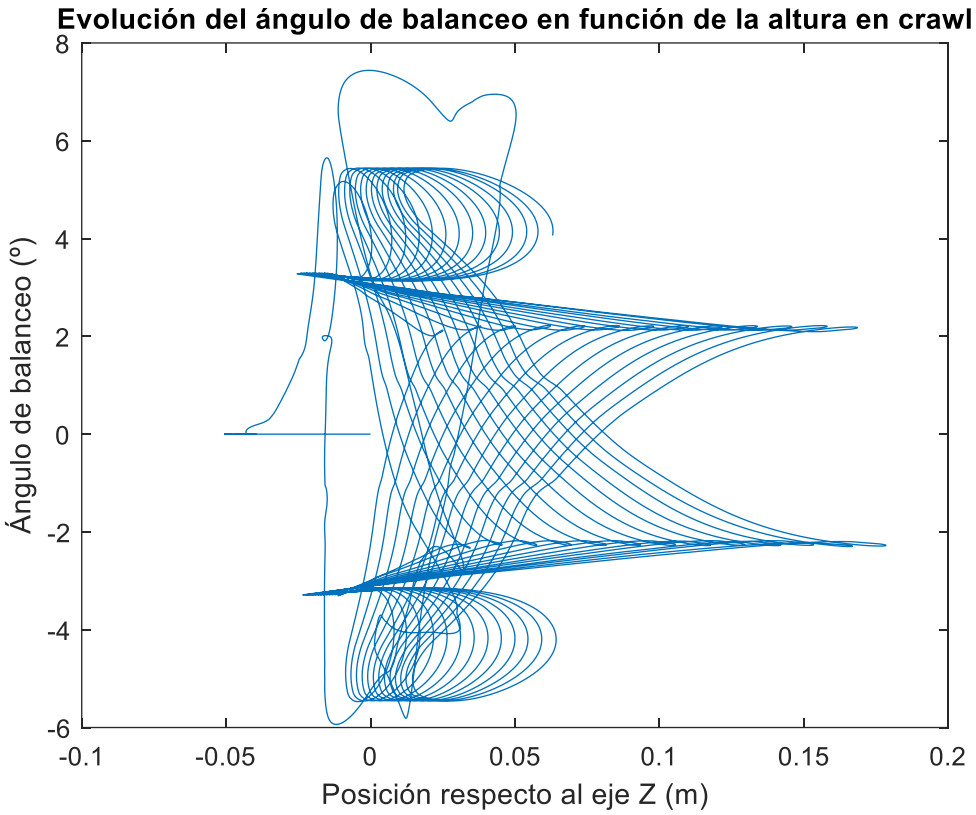


Figura 20: Evolución del ángulo de balanceo en función de la altura del robot funcionando en crawl

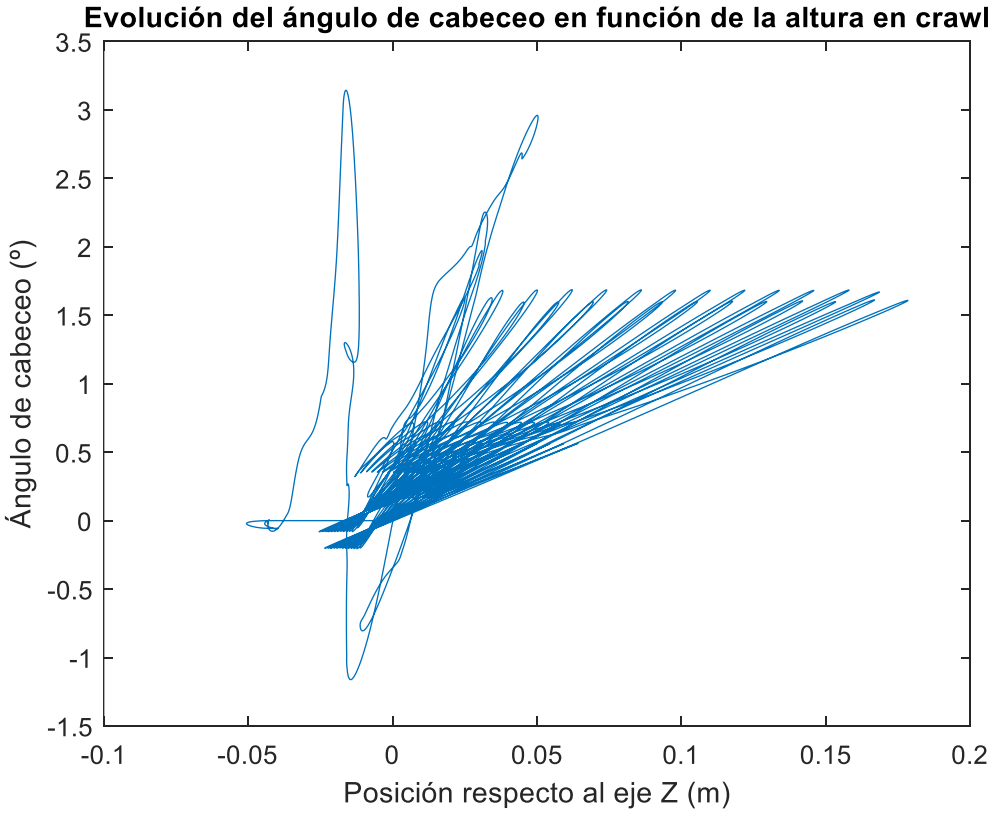


Figura 21: Evolución del ángulo de cabeceo en función de la altura del robot funcionando en crawl.

3.5. Obstáculos

A continuación, se mostrarán los resultados de diversas simulaciones en las que el robot tratará de superar diversos obstáculos, tales como una rampa o unos escalones.

3.5.1. Rampa

Para añadir la rampa se utilizó el modelo anterior y se añadió los contactos de las patas con ésta. En la Figura 22 se observa cómo quedó el modelo con el subsistema del obstáculo añadido.

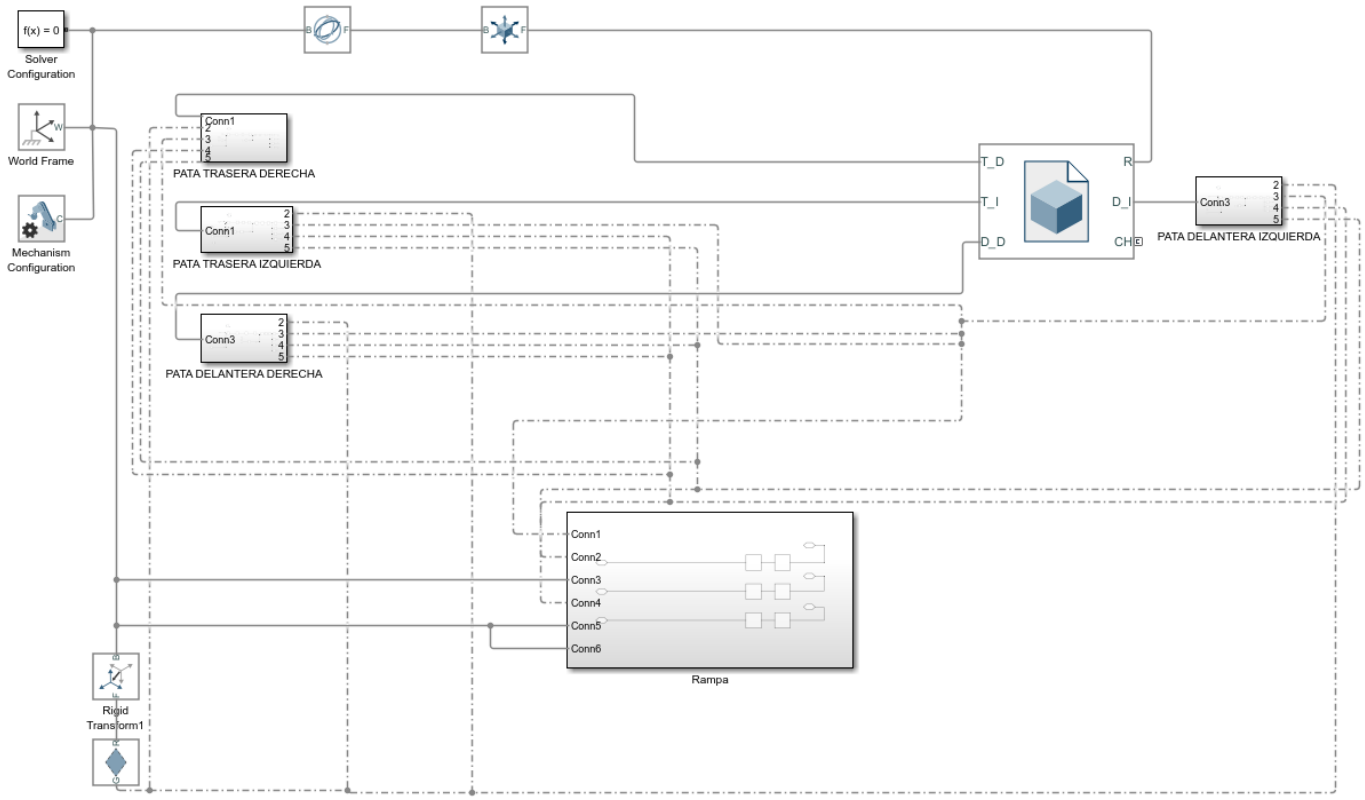


Figura 22: Modelo de simulación de Spot con la rampa.

La rampa está formada por una parte ascendente, una parte llana y una parte descendente, con una inclinación de 8 grados.

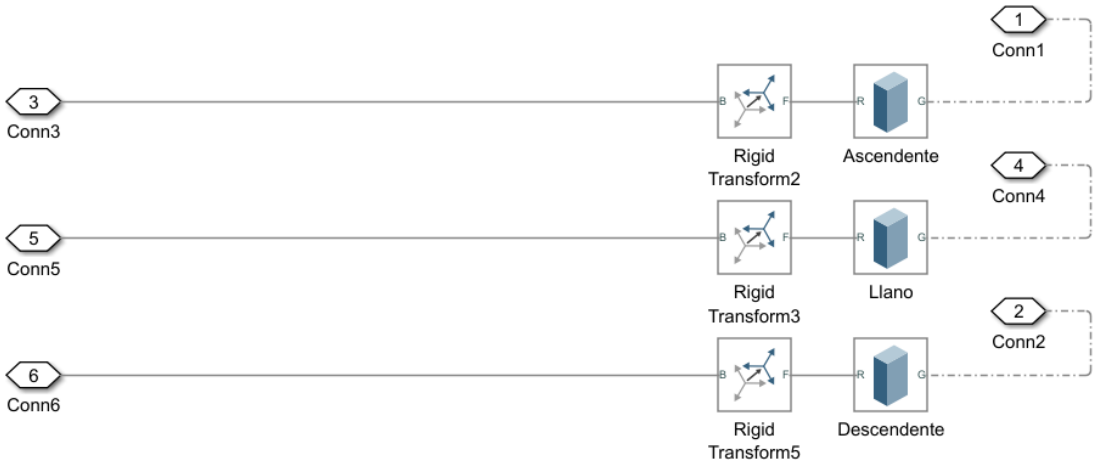


Figura 23: Modelo de la rampa

Se hicieron dos simulaciones, una con el robot avanzando en el modo trot y otra en el modo crawl, con la finalidad de comprobar que es capaz de salvar la rampa en sus dos principales modos de funcionamiento.

En primer lugar, se simulará con el modo trot, donde Spot llega al final de forma bastante rápida.

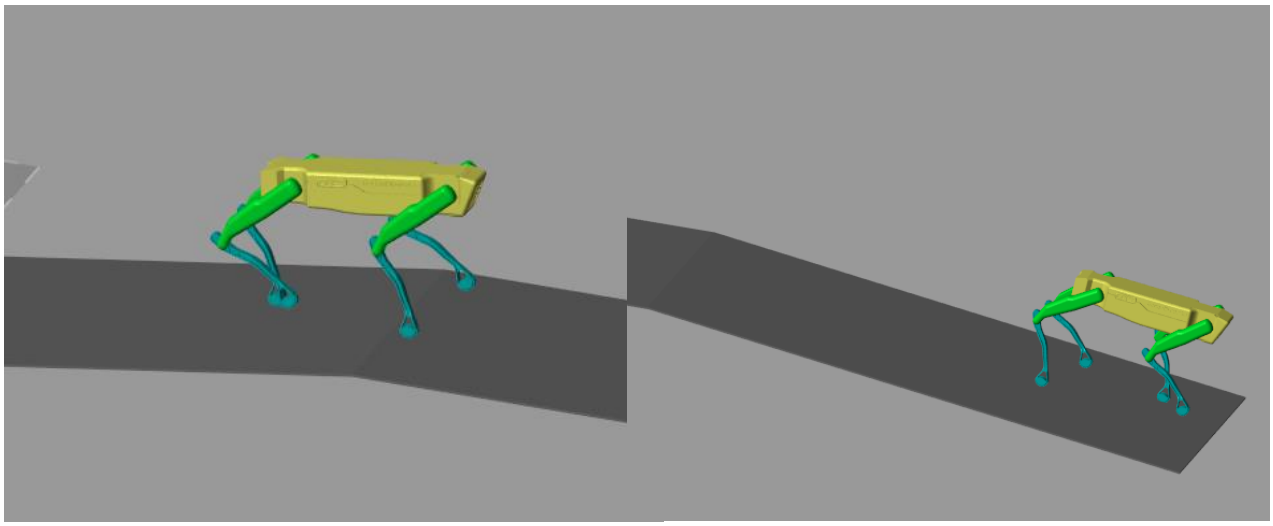


Figura 24: Simulación del robot sorteando la rampa en modo trot

Para poder observar su comportamiento, se obtendrán de nuevo gráficas que relacionen las posiciones con respecto a los ejes X y Z del robot con los ángulos de balanceo y de cabeceo.

En primer lugar, se muestra la variación del ángulo de balanceo a medida que el robot avanza por la rampa mientras funciona en el modo trot (Figura 25). Hay que comentar que esta simulación tiene una duración de 20 segundos.

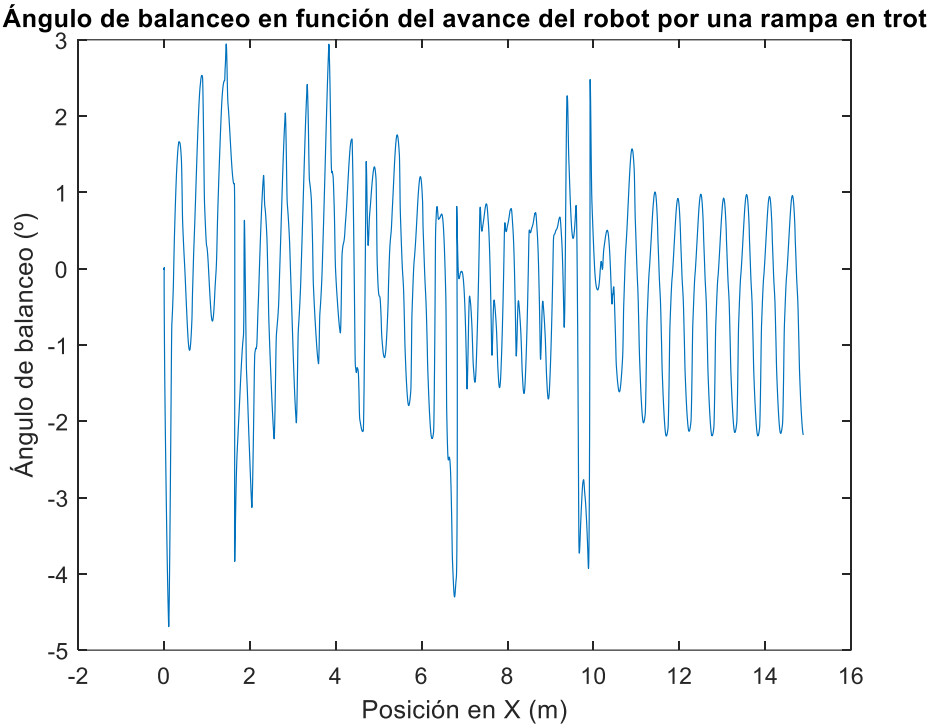


Figura 25: Evolución del ángulo de balanceo en función del avance del robot a través de la rampa en el modo trot.

El ángulo de balanceo, como era de esperar, no tiene una variación significativa a lo largo de los 15 metros que recorre el robot sorteando la rampa.

Pasando al ángulo de cabeceo (Figura 26), si que se ven claramente las fases de la rampa, pudiendo diferencias la parte ascendente, las partes llanas y la parte descendente. Como se observa, los valores que se alcanzan coinciden con la inclinación que se le ha dado a la rampa por la que circula.

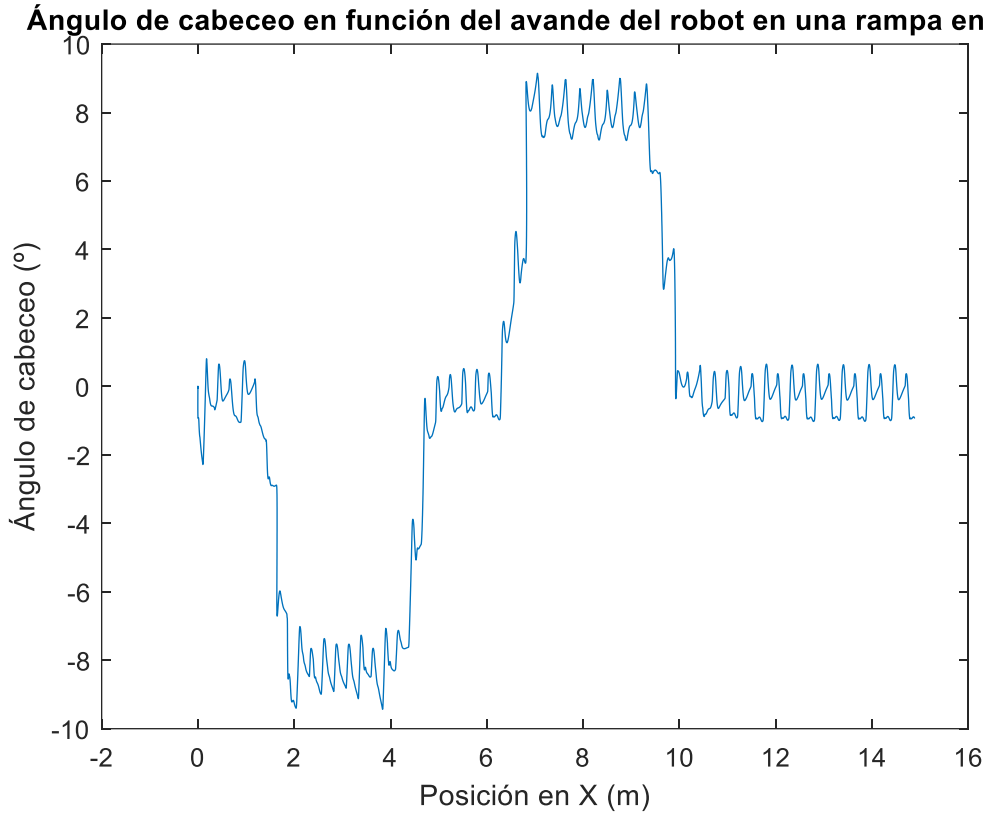


Figura 26: Evolución del ángulo de cabeceo en función del avance del robot a través de la rampa en el modo trot.

Finalmente, se pasará a graficar los ángulos de balanceo y de cabeceo en función de la posición vertical del robot (Figura 27 y Figura 28, respectivamente).

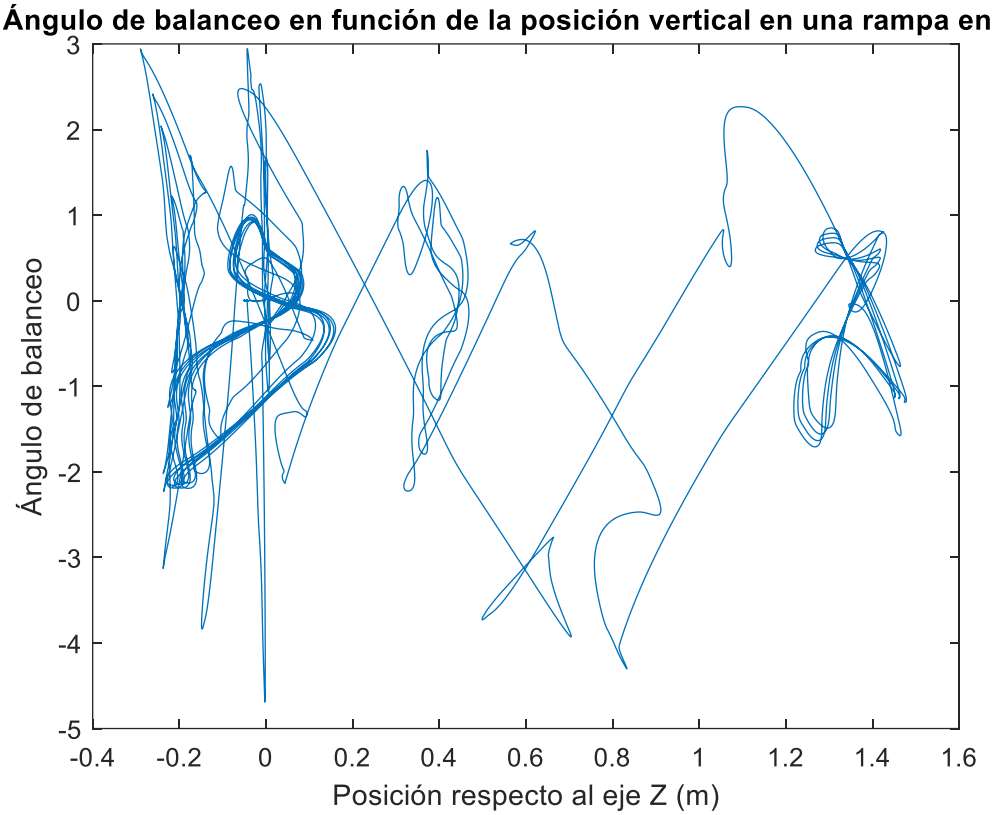


Figura 27: Evolución del ángulo de balanceo en función de la posición vertical en modo trot.

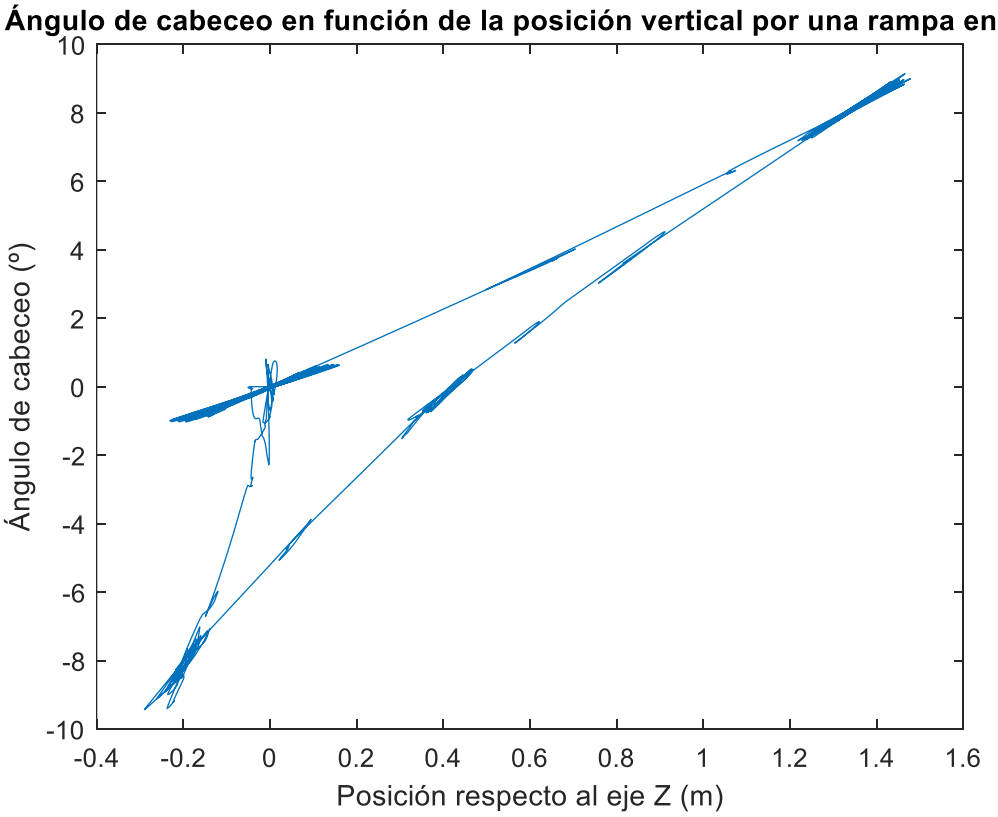


Figura 28: Evolución del ángulo de cabeceo en función de la posición vertical en modo trot.

La variación de la posición vertical oscila entre los -0.2 metros hasta los 1.6 metros, que correspondería con la parte llana de la rampa. Se observa mayor variabilidad en el

ángulo de cabeceo con respecto al ángulo de balanceo, indicando que el robot tiene que ajustar frecuentemente su cabeceo para mantener su estabilidad, lo cual es habitual en terrenos inclinados o irregulares.

Una vez se ha analizado a Spot sorteando una rampa funcionando en modo trot, se procederá a hacer lo mismo, pero en modo crawl el cual, como se ha comentado, es más lento, pero nos proporciona un mayor control. Las gráficas mostrarán, de nuevo, la evolución de los ángulos de cabeceo y balanceo en función de la posición vertical y horizontal del robot, pero, a diferencia que la simulación anterior, será a lo largo de 40 segundos. En la Figura 29 puede observarse a Spot subiendo y bajando por el obstáculo mientras funciona en el modo crawl.

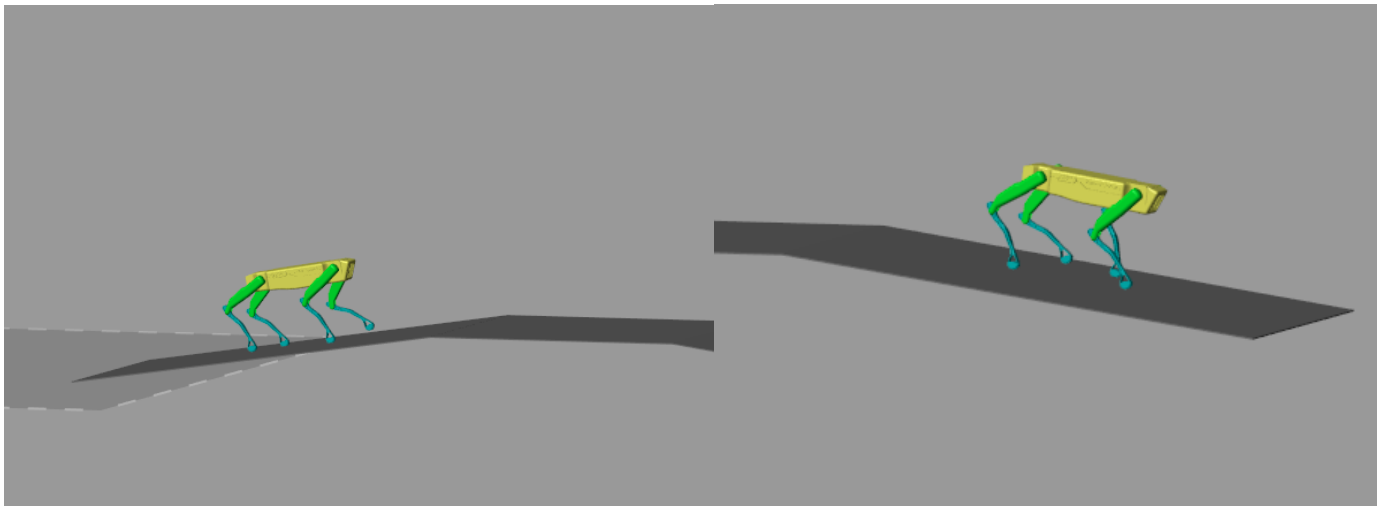


Figura 29: Simulación del robot sorteando la rampa en modo crawl.

En la Figura 30 se representa el ángulo de balanceo en función de la posición horizontal del robot, obteniendo un valor máximo de ±8º, casi el doble que cuando su modo de funcionamiento era el trot, lo cual indica que el robot mantiene una postura más estable lateralmente cuando se mueve a mayor velocidad y con un patrón de marcha más dinámico.

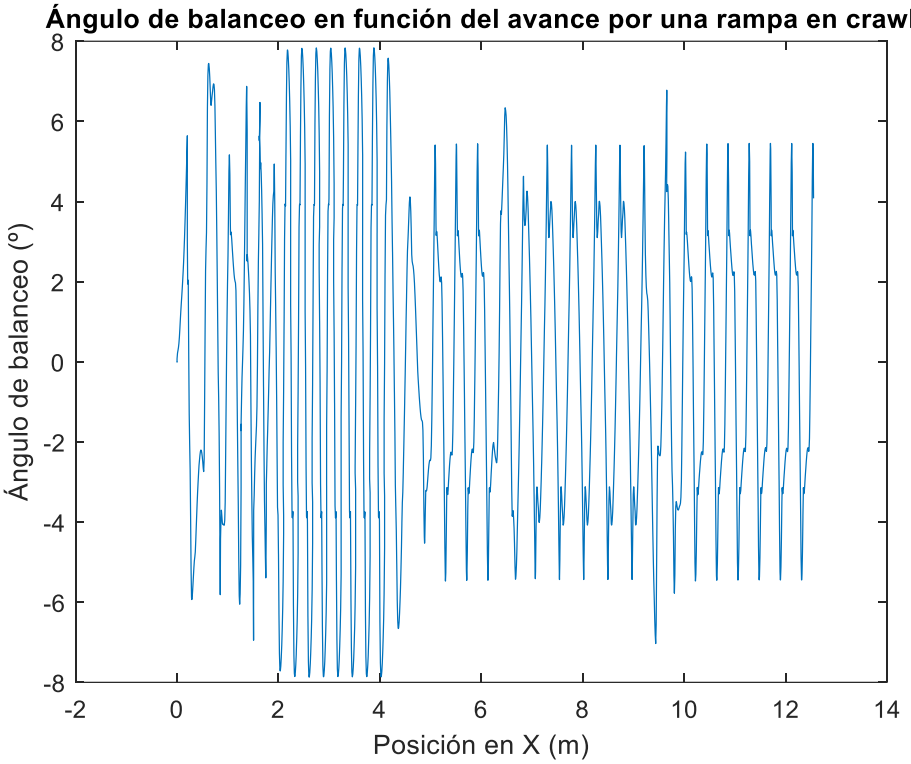


Figura 30: Ángulo de balanceo en función de la posición en X del robot por una rampa funcionando en crawl.

En la Figura 31 se muestra la evolución del ángulo de balanceo en función de la posición en X de Spot. Al igual que durante el trot, se pueden diferenciar claramente las fases ascendente, llana y descendente de la rampa, aunque, en este caso, es ±2º mayor, sugiriendo que el control del cabeceo es más robusto en trot.

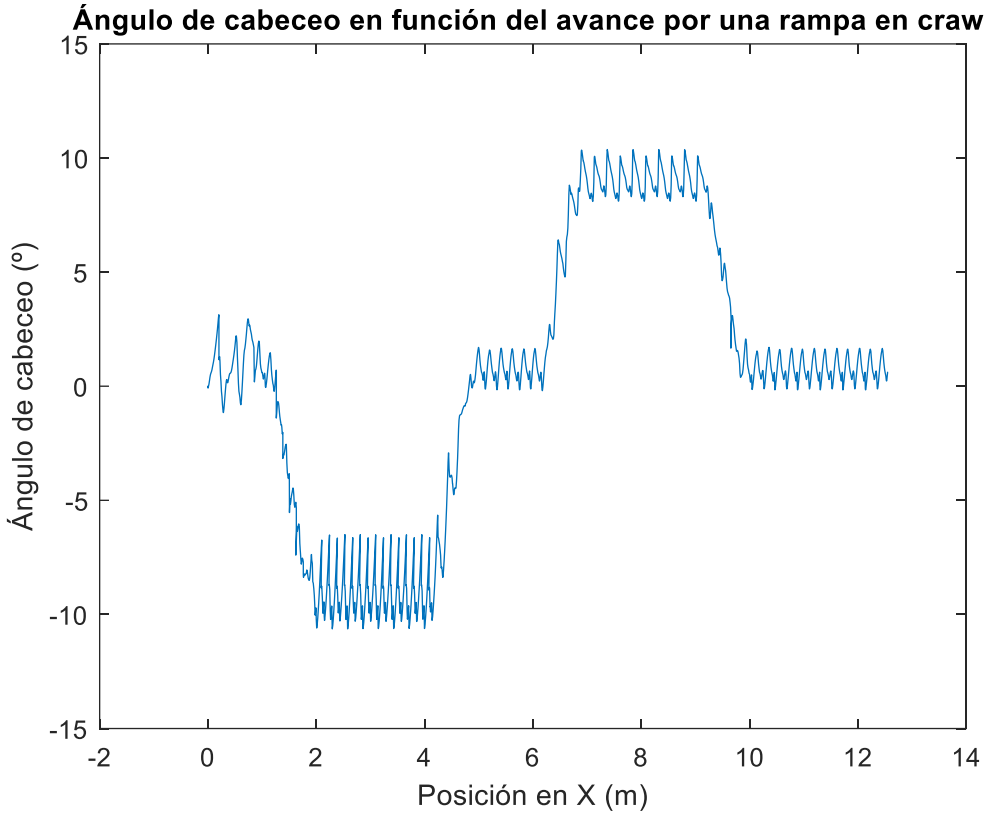


Figura 31: Ángulo de cabeceo en función de la posición en X del robot.

Finalmente se mostrará cómo evolucionan los ángulos de balanceo y de cabeceo en función de la posición vertical del robot al circular sobre la rampa (Figura 32y Figura 33, respectivamente).

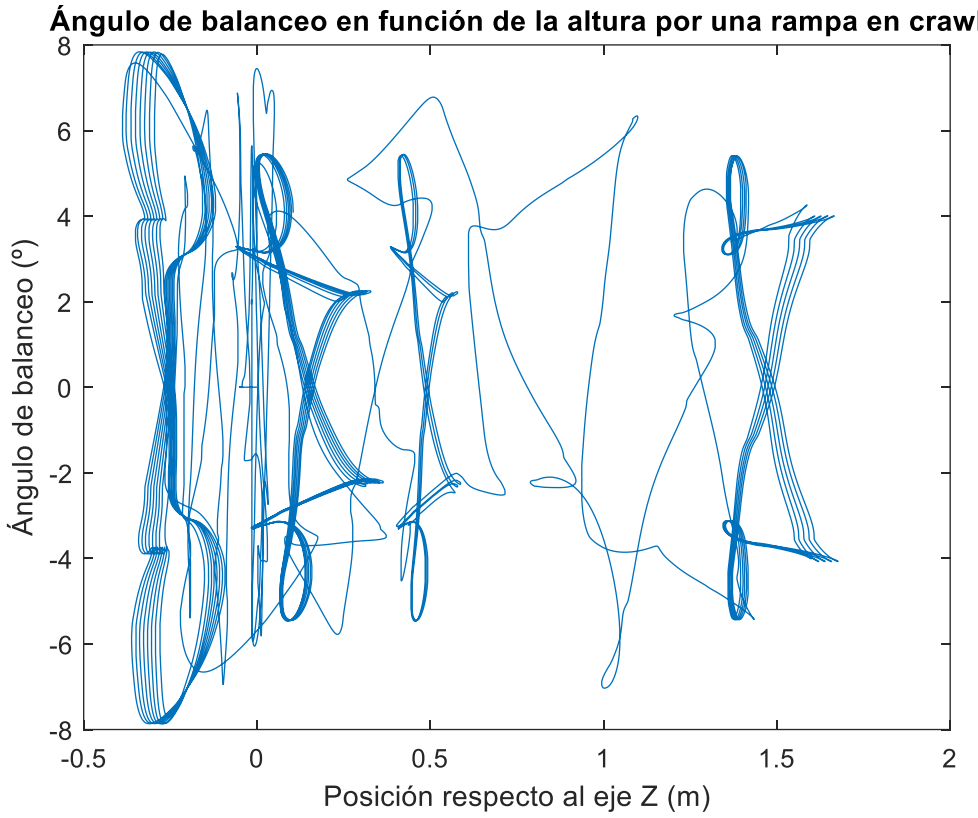


Figura 32: Ángulo de balanceo en función de la posición vertical del robot por una rampa funcionando en crawl.

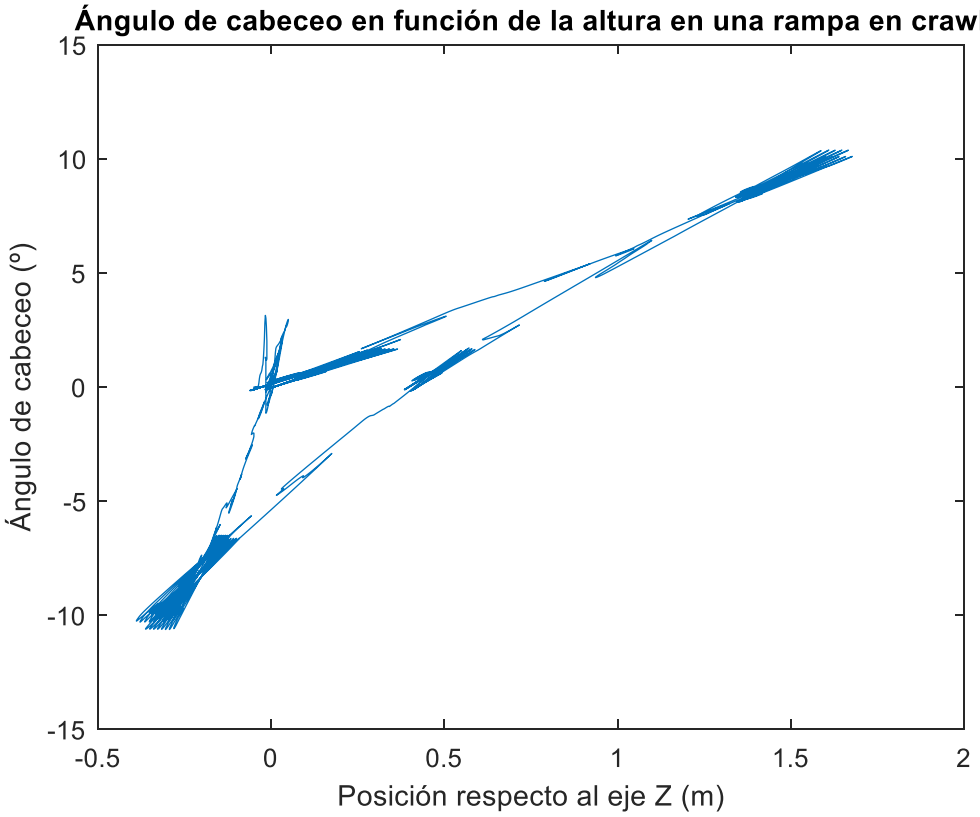


Figura 33: Ángulo de cabeceo en función de la posición vertical del robot por una rampa funcionando en crawl.

En el modo crawl, el ángulo de balanceo muestra oscilaciones más pronunciadas que en el modo trot. Esto sugiere que el robot tiene que hacer mayores ajustes en su balanceo para mantener la estabilidad durante movimientos más lentos y precisos. Esto puede ser debido a la necesidad de levantar y colocar cada pata de forma individual, o que crea mayores perturbaciones en el equilibro del robot.

Lo mismo ocurre con el ángulo de cabeceo, pese a que se comporta de una forma muy parecida al modo trot, tiene unos valores superiores, lo que puede indicar que el robot enfrenta más desafíos para mantener su orientación frontal estable cuando se mueve más lentamente y de manera más controlada.

3.5.2. Escalones

El modelo en Simscape de los escalones es el mismo que el utilizado para la simulación de la rampa, modificando el tipo de obstáculo

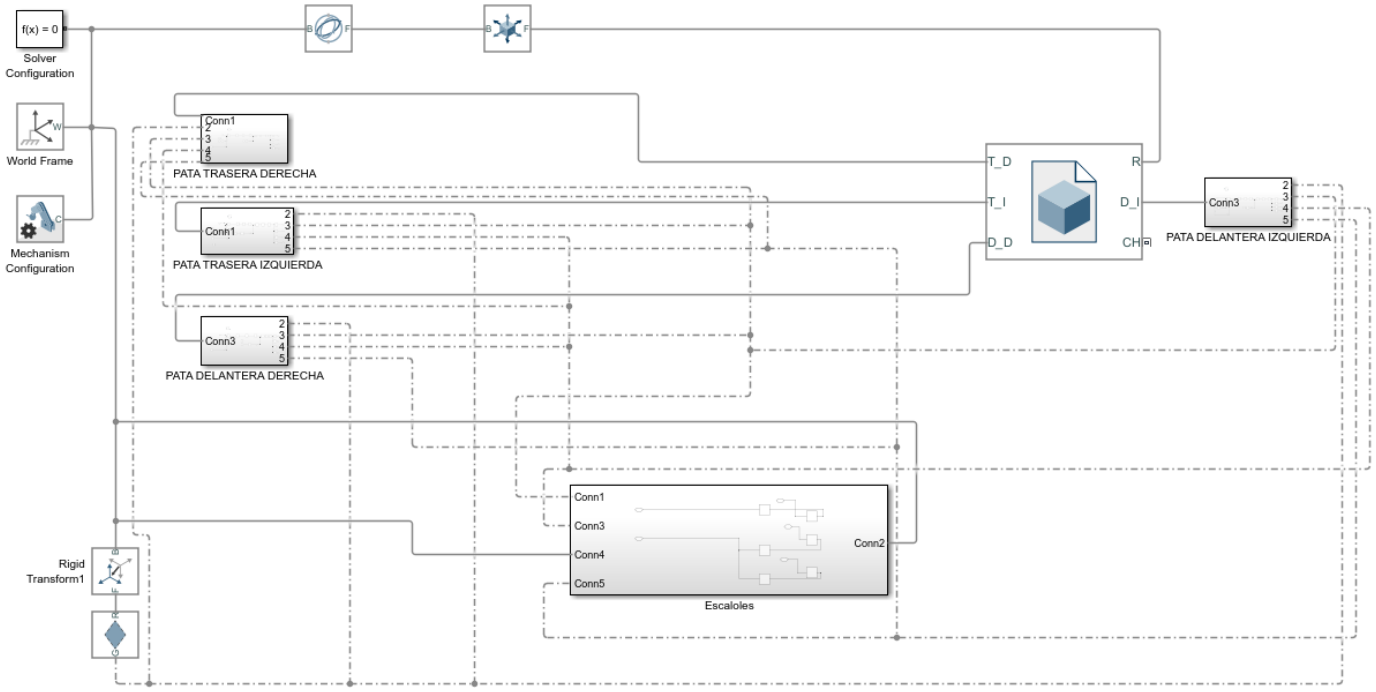


Figura 34: Modelo de simulación de los escalones

El subsistema del obstáculo está formado por 3 escalones y sus respectivas transformadas para colocarlos como se desea.

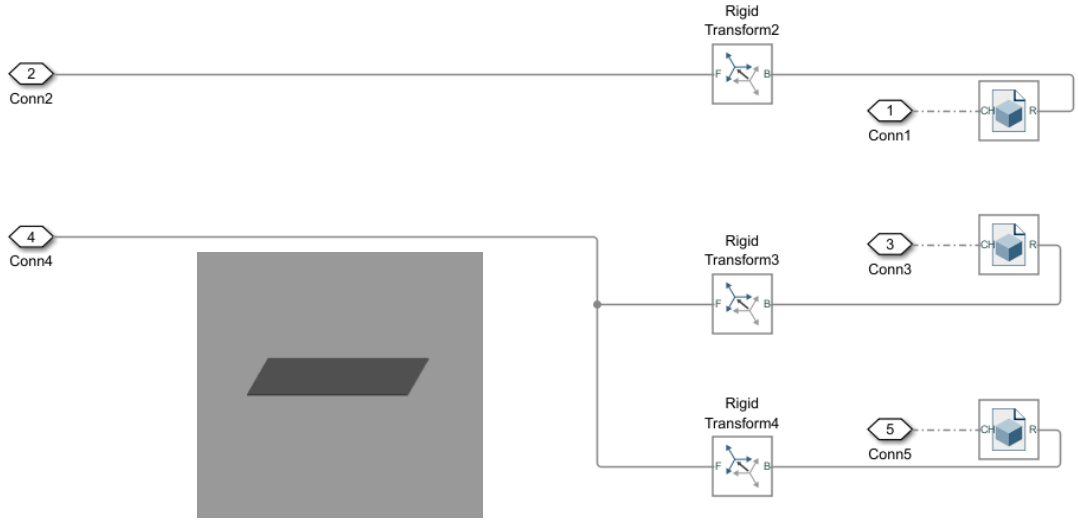


Figura 35: Modelo del subsistema de los escalones.

Esta simulación únicamente se ha podido llevar a cabo cuando el robot se encuentra en el modo de funcionamiento de trot, pudiendo subir estos pequeños escalones sin complicaciones (Figura 36).

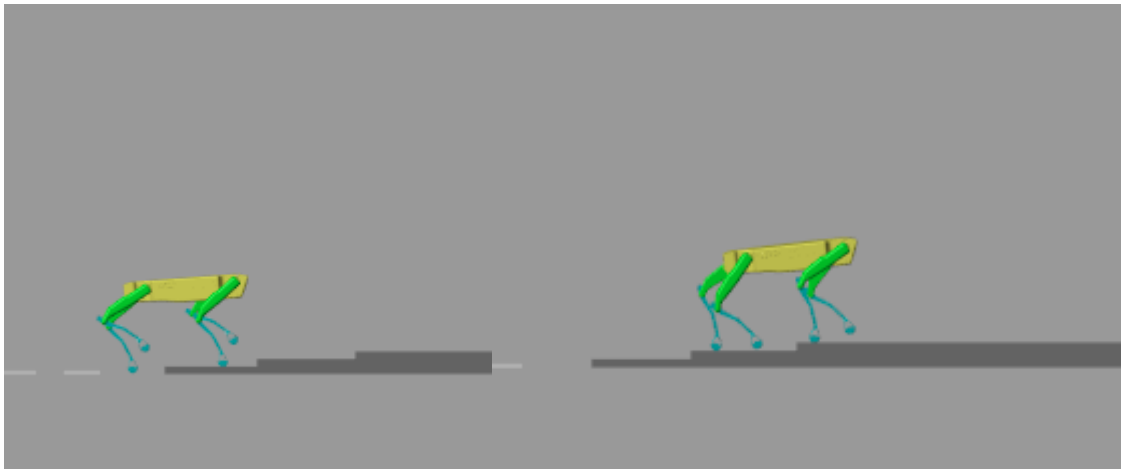


Figura 36: El robot subiendo unos escalones.

Para poder mostrar su comportamiento, se volverán a relacionar las posiciones del robot con respecto a los ejes X y Z con sus ángulos de balanceo y cabeceo, tal y como se ha hecho en las simulaciones previas.

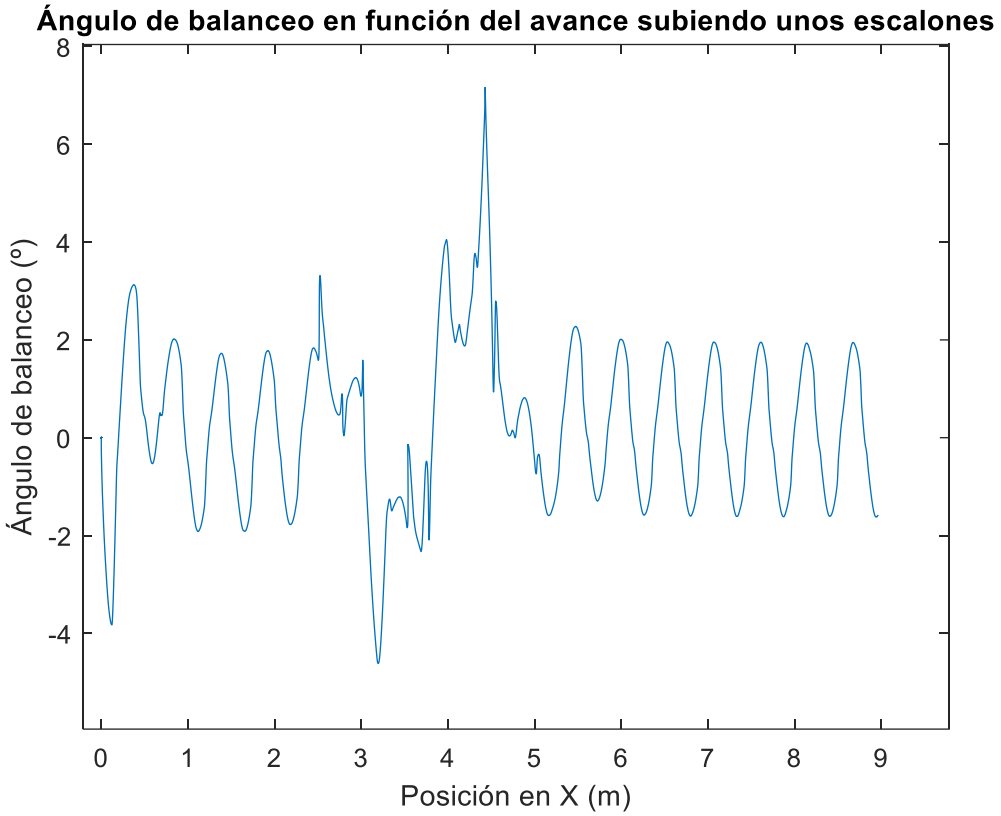


Figura 37: Ángulo de balanceo en función del avance del robot cuando sube escalones.

En la Figura 37 se muestra cómo varía el ángulo de balanceo cuando el robot sube los tres escalones. Puede notarse cuándo llega al obstáculo porque prácticamente se duplica esta medida, dado a entender que el robot tiene que hacer mayores ajustes en su balanceo para mantener la estabilidad.

Por otra parte, en la Figura 38 se analiza cómo evoluciona el ángulo de cabeceo ante el avance de la máquina y, es en el mismo punto que en la Figura 37 donde pasa de ser de ±1º a -6º, dando a entender que es durante ese tramo en el que el robot está subiendo los escalones.

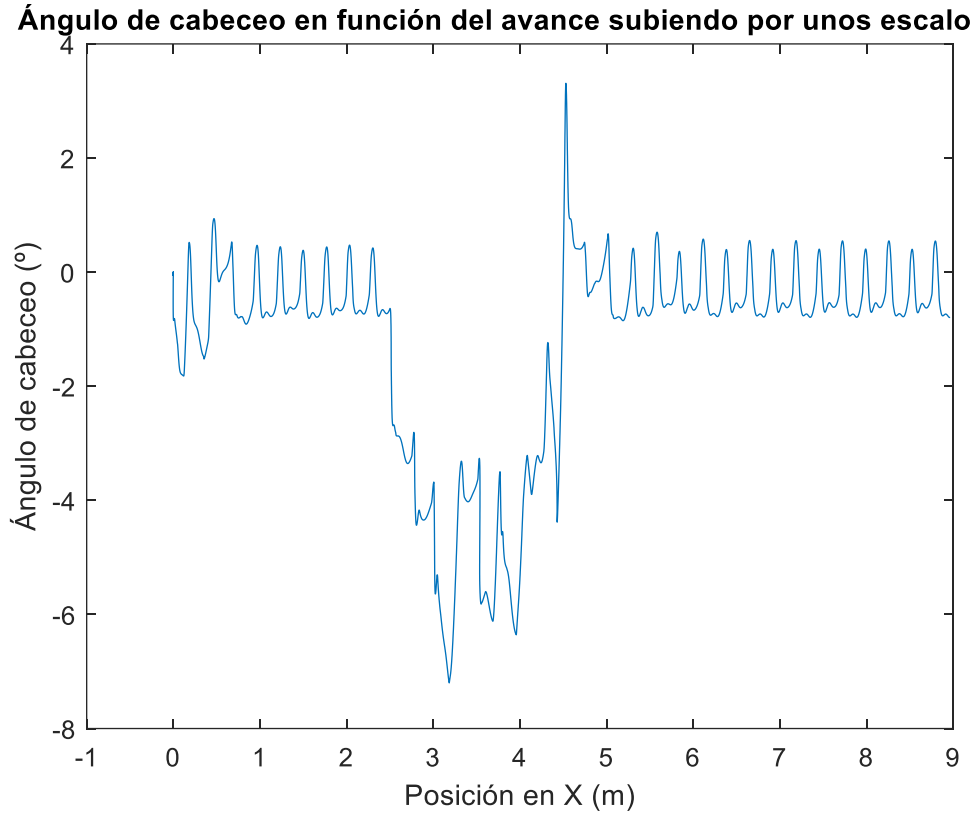


Figura 38: Ángulo de cabeceo en función del avance del robot cuando sube escalones.

En la Figura 39 y Figura 40 se representan los ángulos de balanceo y de cabeceo en función de la posición vertical que tiene el robot mientras sortea el obstáculo.

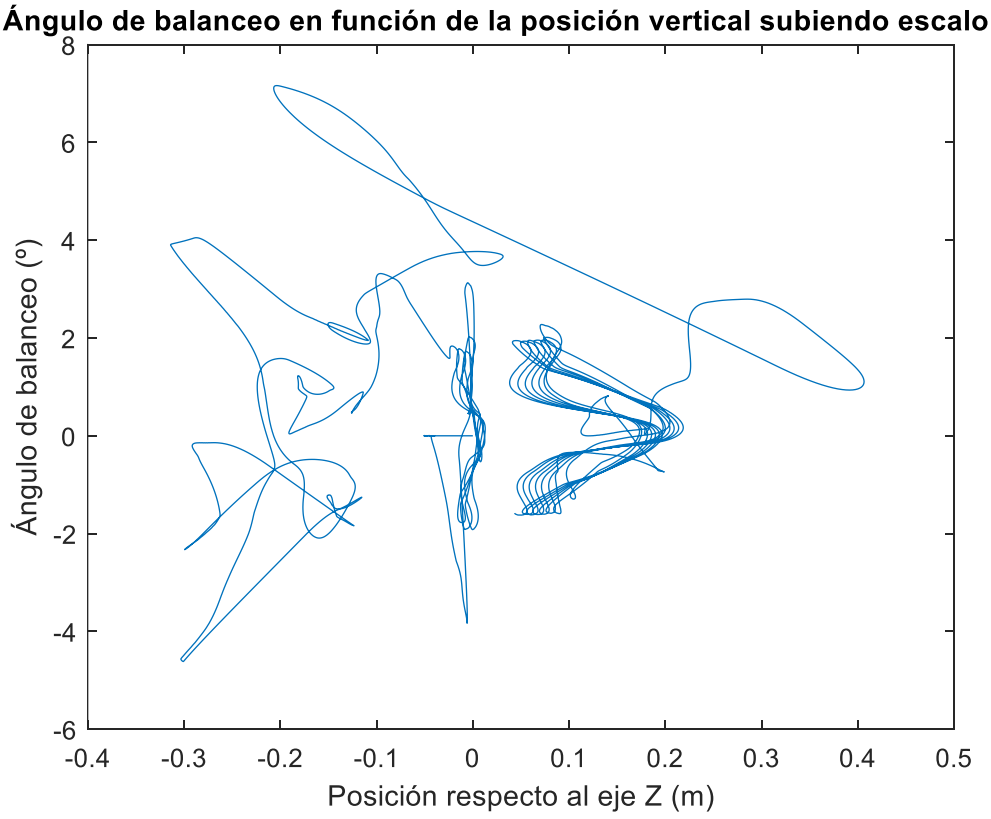


Figura 39: Ángulo de balanceo en función de la posición vertical del robot cuando sube escalones.

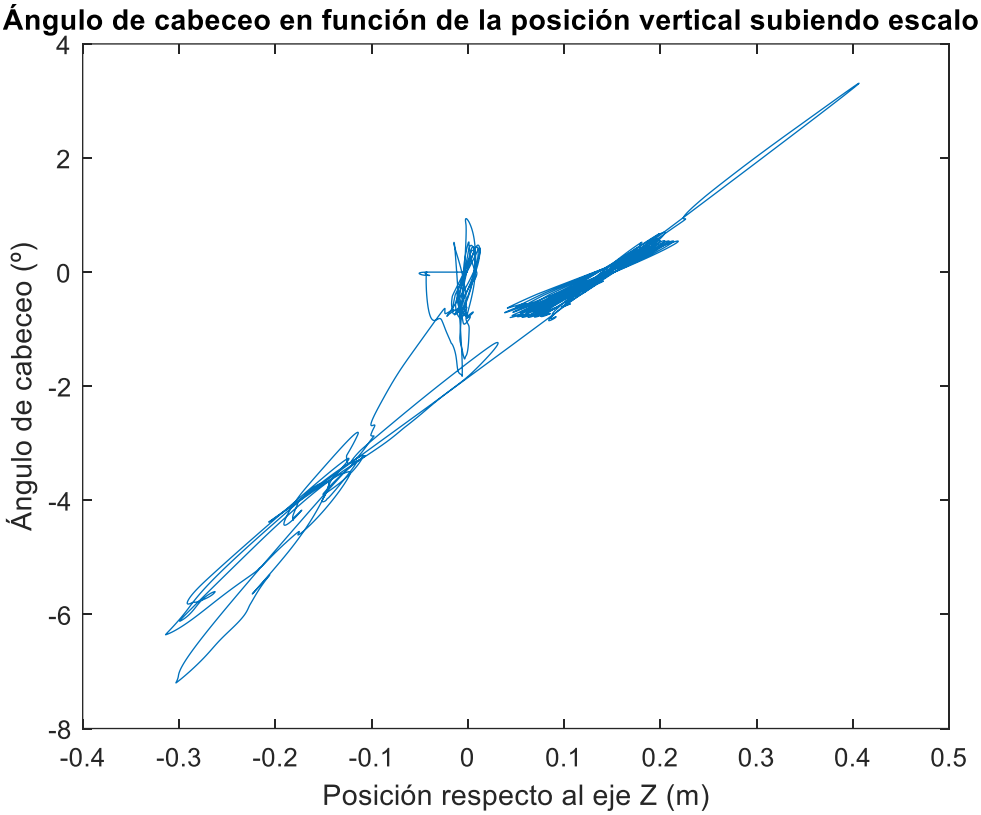


Figura 40: Ángulo de cabeceo en función de la posición vertical del robot cuando sube escalones.

El ángulo de balanceo es bastante reducido tanto cuando el robot se encuentra en 0 metros como en 0.2 metros (la parte más alta de la escalera), y mientras se están subiendo los escalones es cuando varía más.

Lo mismo ocurre con el ángulo de cabeceo, es prácticamente nulo cuando camina sobre llano, y tiene ciertas variaciones cuando está sorteando los escalones.

3.5.3. Baches

En las pruebas más recientes, se han colocado dos tipos de baches para evaluar la capacidad del robot de sortear obstáculos. Estos baches están diseñados para poner a prueba tanto la agilidad como la estabilidad del robot en diferentes situaciones:

- Primer Bache: Este bache está configurado de tal manera que el robot debe sortearlo utilizando únicamente las patas de un lado. Este tipo de obstáculo evalúa la capacidad del robot para mantener el equilibrio y la coordinación al enfrentar un obstáculo que afecta solo a un lado de su cuerpo.

- Segundo Bache: Este bache está dispuesto para que el robot deba pasar completamente por encima de él. Este obstáculo está diseñado para probar la

habilidad del robot de superar baches que afectan a todas sus patas simultáneamente, evaluando su capacidad para elevarse y mantener la estabilidad mientras avanza sobre el obstáculo.

El modelo de Simscape no tiene grandes variaciones con respecto a las simulaciones anteriores, se han añadido los dos baches con sus respectivas transformadas y contactos con las patas.

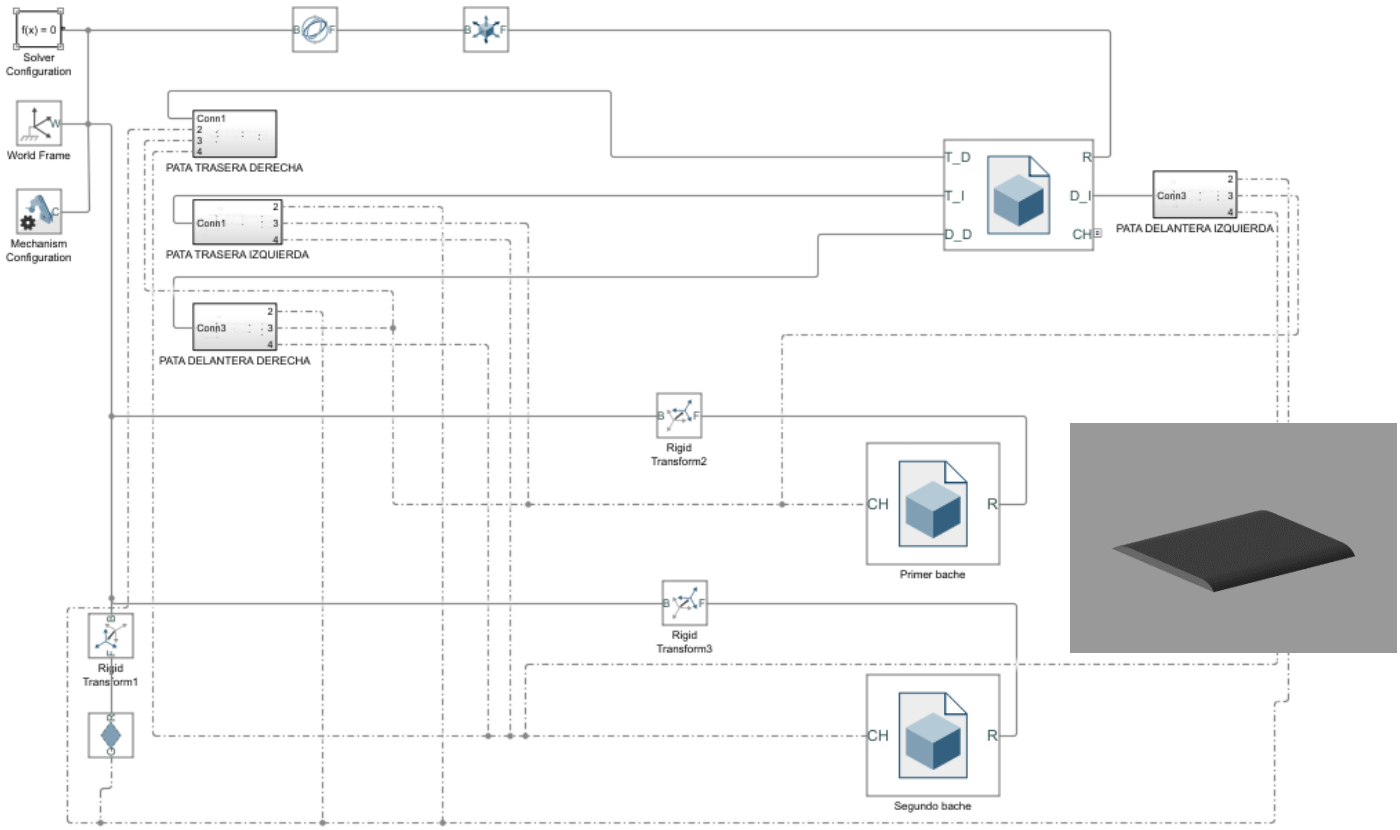


Figura 41: Modelo de la simulación con baches.

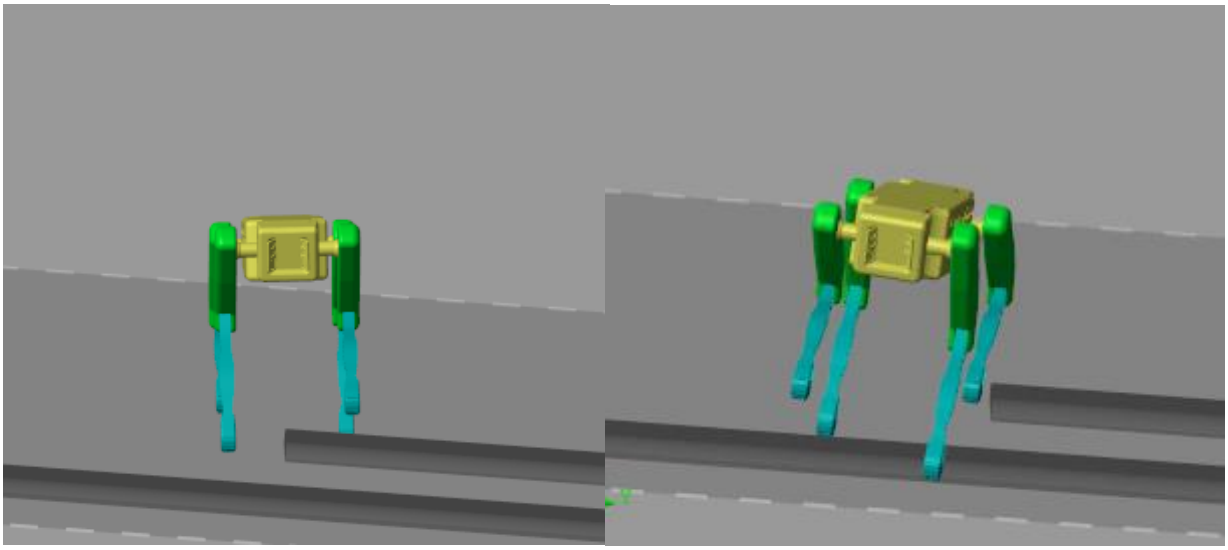


Figura 42: Spot sorteando los dos tipos de baches.

En la Figura 42 se muestra al robot sorteando los dos baches. No obstante, se volverá a estudiar su comportamiento mediante gráficas relacionando los ángulos de balanceo y cabeceo con las posiciones en X y en Z de Spot. Esta simulación también se ha podido realizar únicamente en el modo de funcionamiento trot.

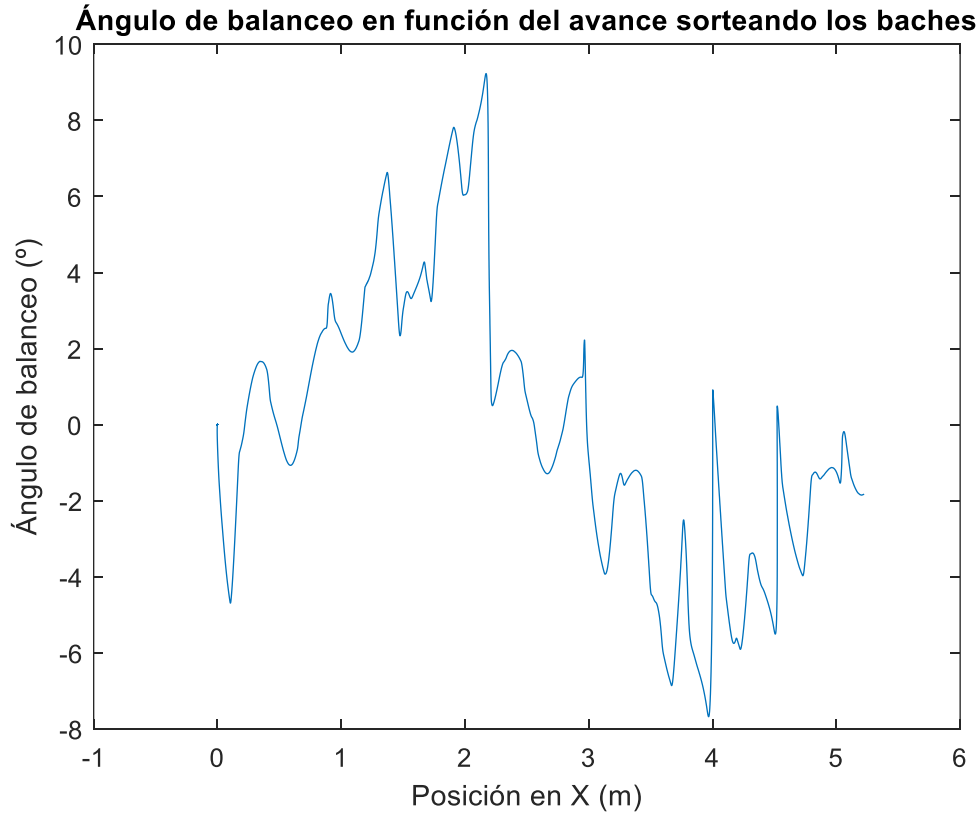


Figura 43: Relación del ángulo de balanceo con respecto al avance del robot cuando sortea los baches.

Como puede observarse en la Figura 43, el ángulo de balanceo tiene un notable aumento (hasta casi 10º) cuando está superando el primer bache, y otro (menor, de casi -8º) cuando está por el segundo. Tiene lógica que sea superior con el primer obstáculo debido a que el robot levanta únicamente las patas de su lazo izquierdo.

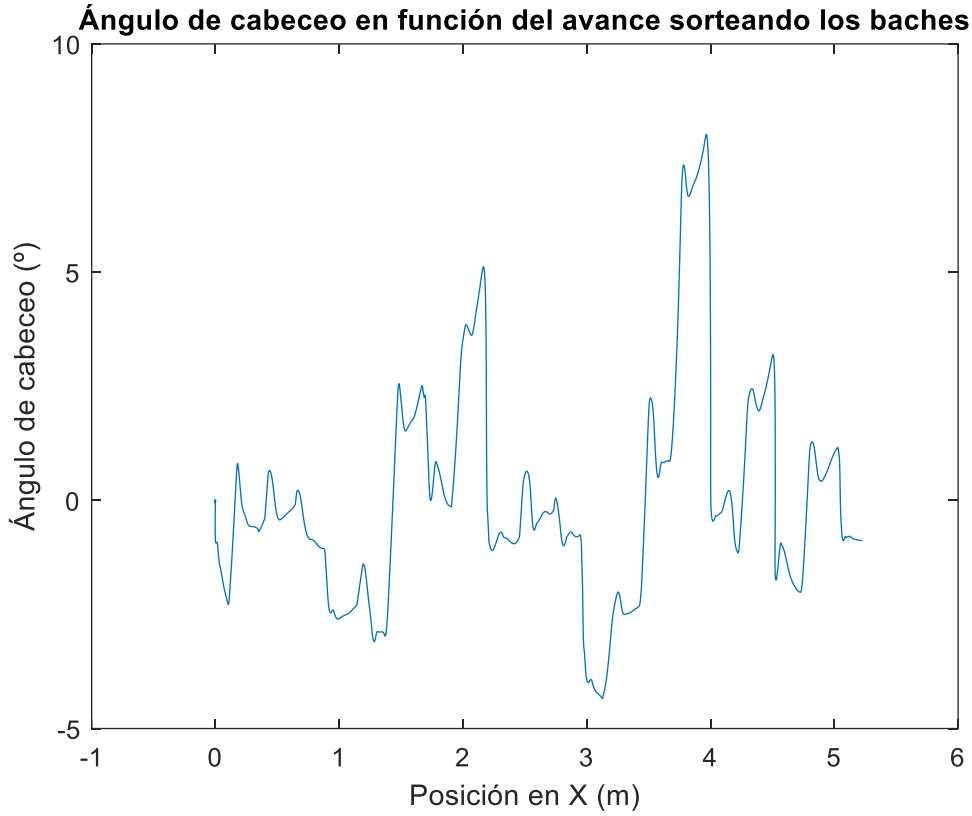


Figura 44: Relación del ángulo de cabeceo con respecto al avance del robot cuando sortea los baches.

Ocurre lo mismo en la Figura 44 cuando se muestra la evolución del ángulo de cabeceo, donde aumenta significativamente cuando el robot se encuentra sorteando los obstáculos.

Para terminar, se muestra la relación de éstos mismos ángulos con la posición vertical de Spot (Figura 45 y Figura 46).

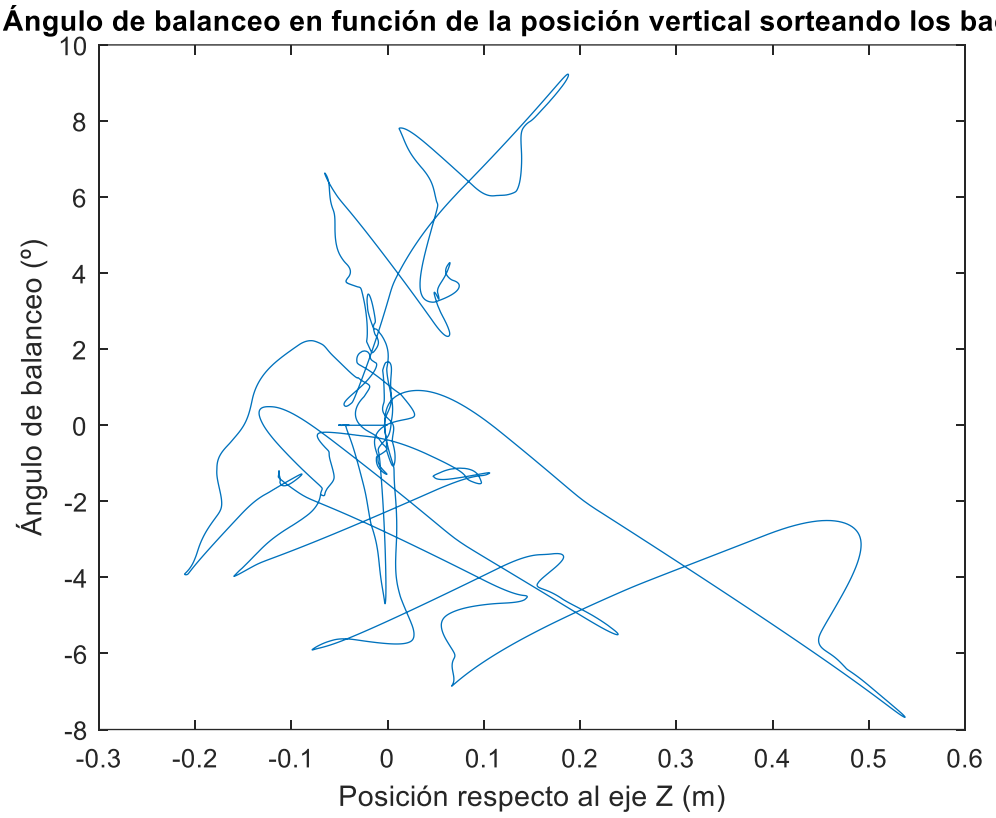


Figura 45: Relación del ángulo de balanceo en función de la posición vertical del robot cuando sortea los baches.

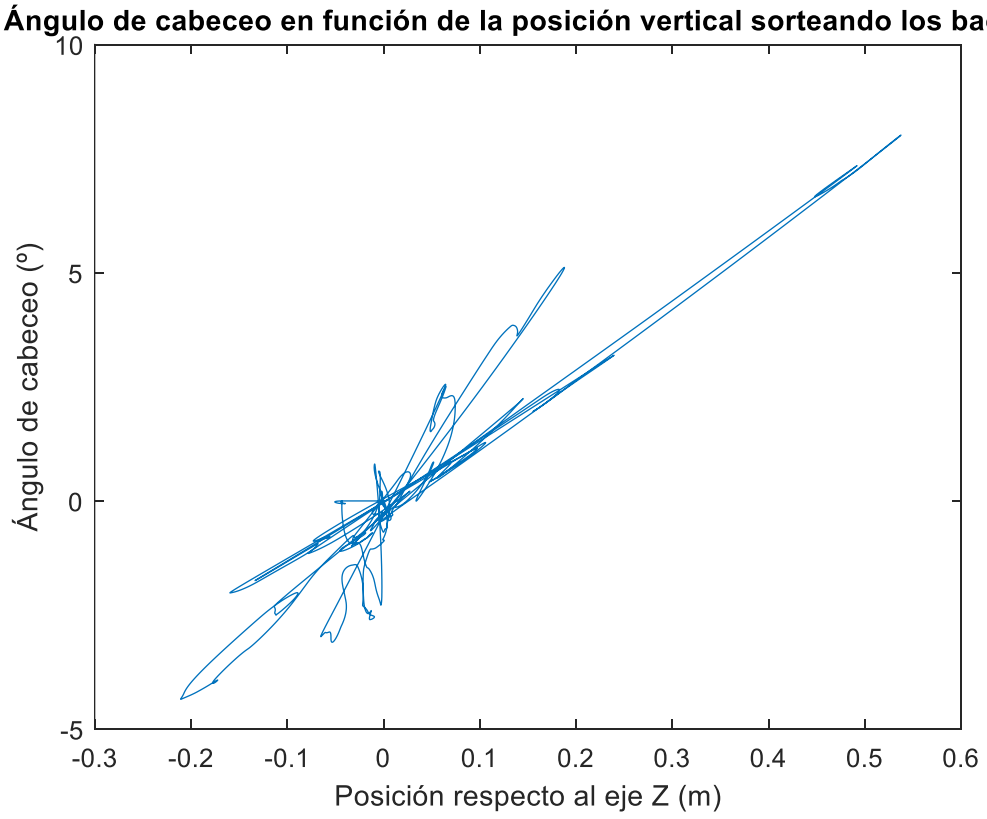


Figura 46: Relación del ángulo de cabeceo en función de la posición vertical del robot cando sortea los baches.

En cuanto al ángulo de balanceo, es mayor cuando el robot está sorteando el primer bache, pero no coincide con el punto de mayor altura del robot. Esto se debe a que, al levantar únicamente las patas de un lado, el cuerpo no se eleva tanto como en el segundo bache, donde el ángulo de balanceo también tiene un aumento significativo. Al ángulo de cabeceo le ocurre algo similar; mientras el robot camina sobre terreno llano, este ángulo es muy pequeño, pero al sortear el segundo bache, alcanza su valor más alto. Además, en el primer bache, también muestra un valor elevado, pero al no levantar las patas de ambos costados, no es tan pronunciado.

El robot ha demostrado su capacidad para sortear el primer bache, manteniendo el equilibrio y la coordinación al levantar únicamente las patas de un costado. Asimismo, ha superado el segundo bache, elevándose y manteniendo la estabilidad mientras avanzaba sobre el obstáculo.

3.6. Caminar de lado

La última simulación realizada sirve para comprobar que Spot es capaz de caminar de lado. Esto es posible a que las articulaciones que unen el cuerpo con la parte superior de cada una de sus patas son de tipo esféricas, permitiendo rotaciones en múltiples planos, proporcionando tres grados de libertad. Gracias a esta capacidad de movimiento complejo y preciso, Spot es capaz de realizar maniobras avanzadas, como caminar de lado, manteniendo su estabilidad y control en diversas condiciones de terreno.

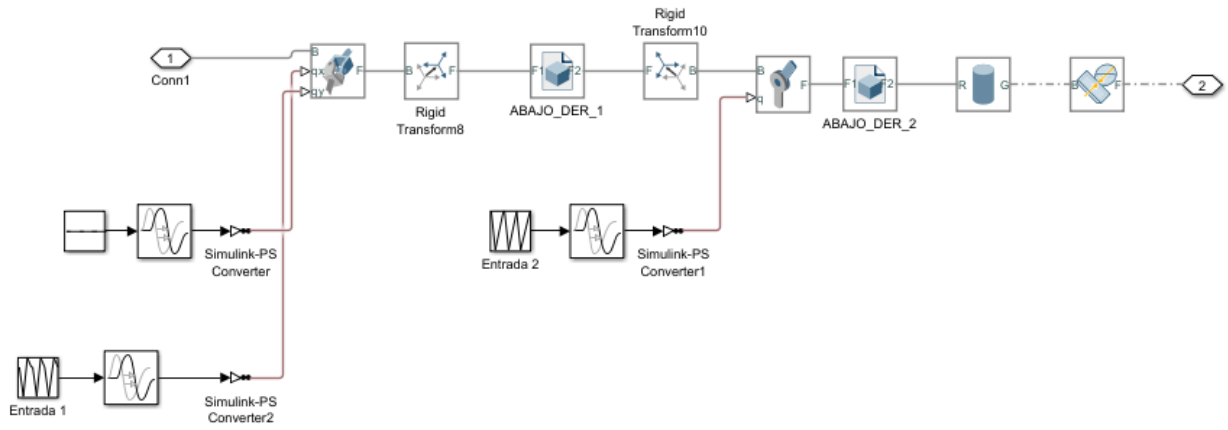


Figura 47: Modelo de simulación de cada pata cuando el robot camina de lado

La única diferencia en el modelo de Simscape con respecto a las simulaciones anteriores es que la entrada de la junta universal de cada una de las patas pasará a ser el segundo eje de rotación, alrededor del eje y (Figura 47).

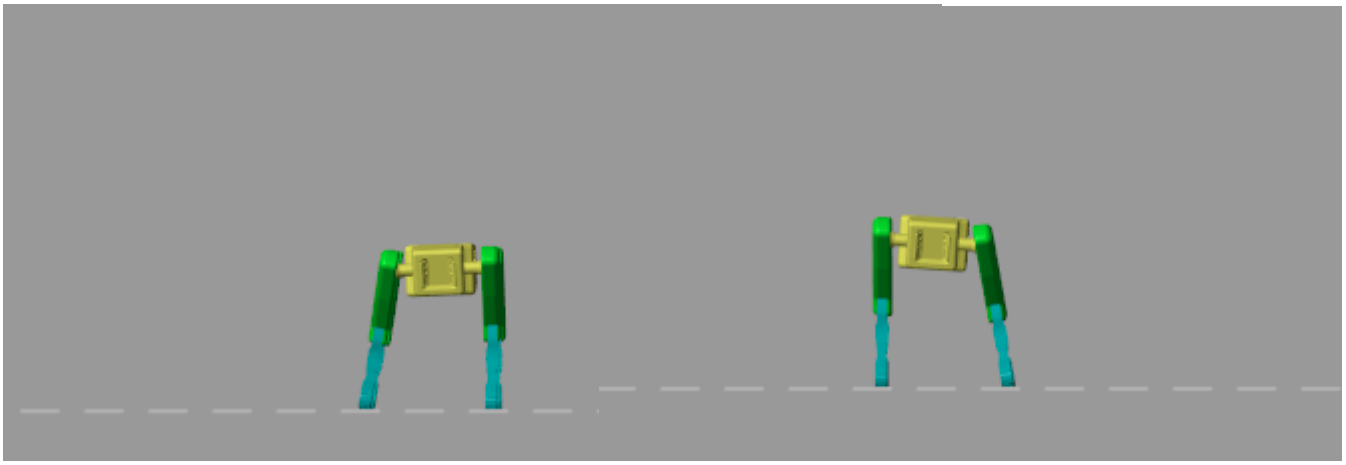


Figura 48: Spot caminando de lado.

Para poder observar el comportamiento del robot, se generarán las gráficas comparando los ángulos de cabeceo y de balanceo en función de la posición respecto al eje Y del robot.

Se empezará analizando la variación de los ángulos en función del avance del robot (Figura 49y Figura 50)

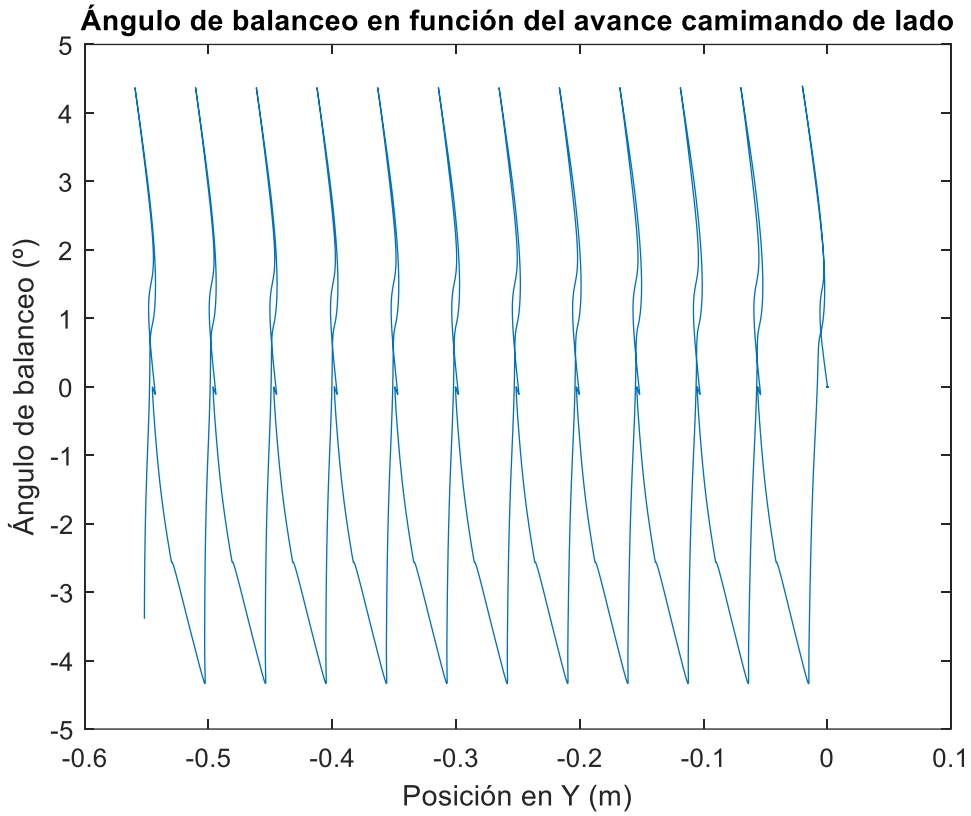


Figura 49: Evolución del ángulo de balanceo en función del avance del robot cuando camina de lado.

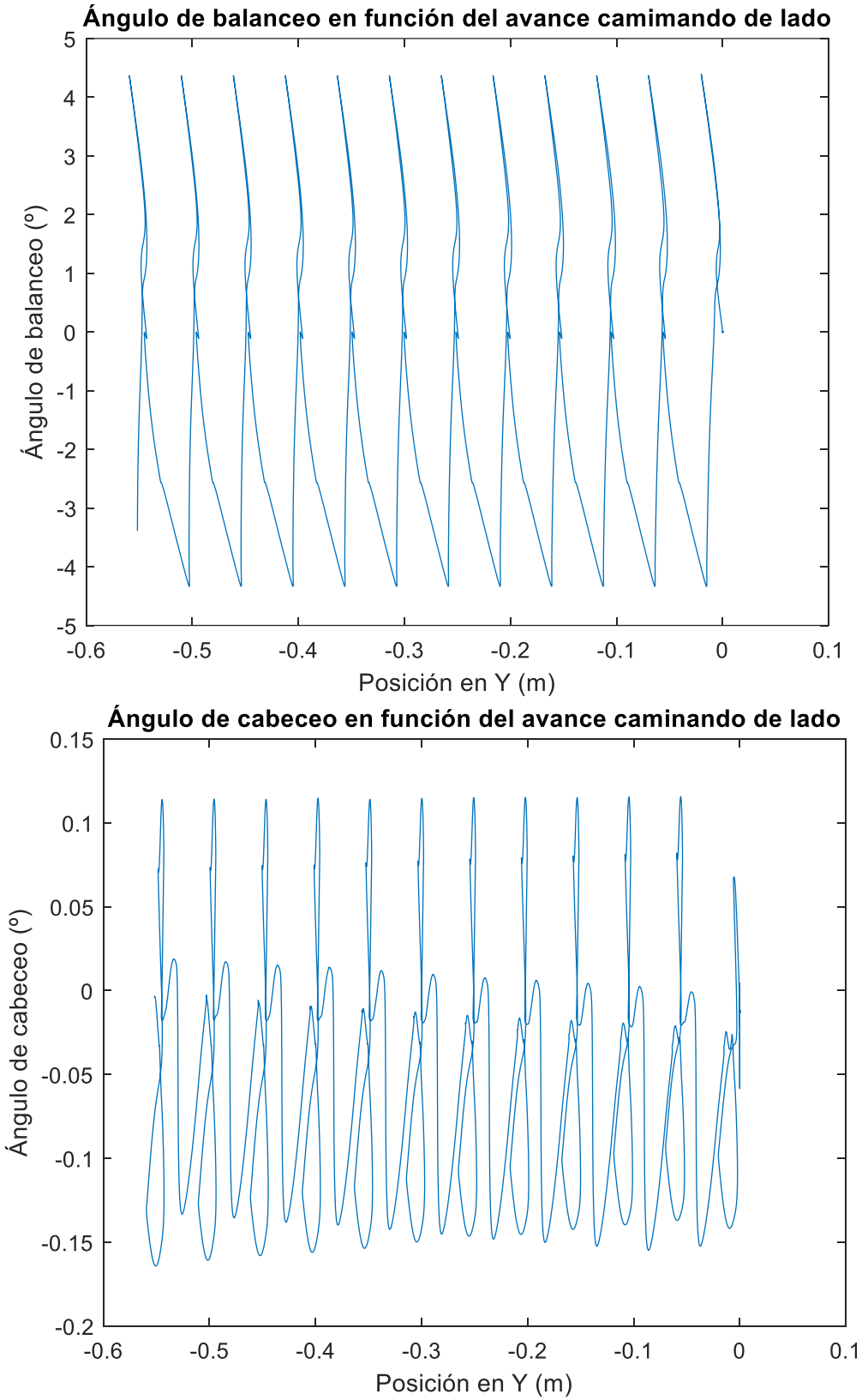


Figura 50: Evolución del ángulo de cabeceo en función del avance del robot cuando camina de lado.

Lo primero que llama la atención es que en 10 segundos el robot se ha desplazado poco más de medio metro, lo que indica que este modo de

desplazamiento es bastante lento. Sin embargo, su principal utilidad radica en la capacidad de evitar ciertos obstáculos o irregularidades en el terreno.

Cada oscilación en el gráfico corresponde a un movimiento de las patas para desplazarse de lado. Como puede observarse, el ángulo de balanceo presenta una variación bastante notable, lo cual es lógico y necesario para poder lograr este tipo de movimiento lateral. Por otro lado, el ángulo de cabeceo prácticamente es nulo, manteniendo el cuerpo del robot fijo en ese aspecto y asegurando su estabilidad durante este desplazamiento.

La posición vertical del robot no tiene demasiada relevancia durante esta simulación debido a que, tal y como se muestra en la Figura 51, esta variación es bastante pequeña, obteniendo un valor máximo de poco más de 10 cm.

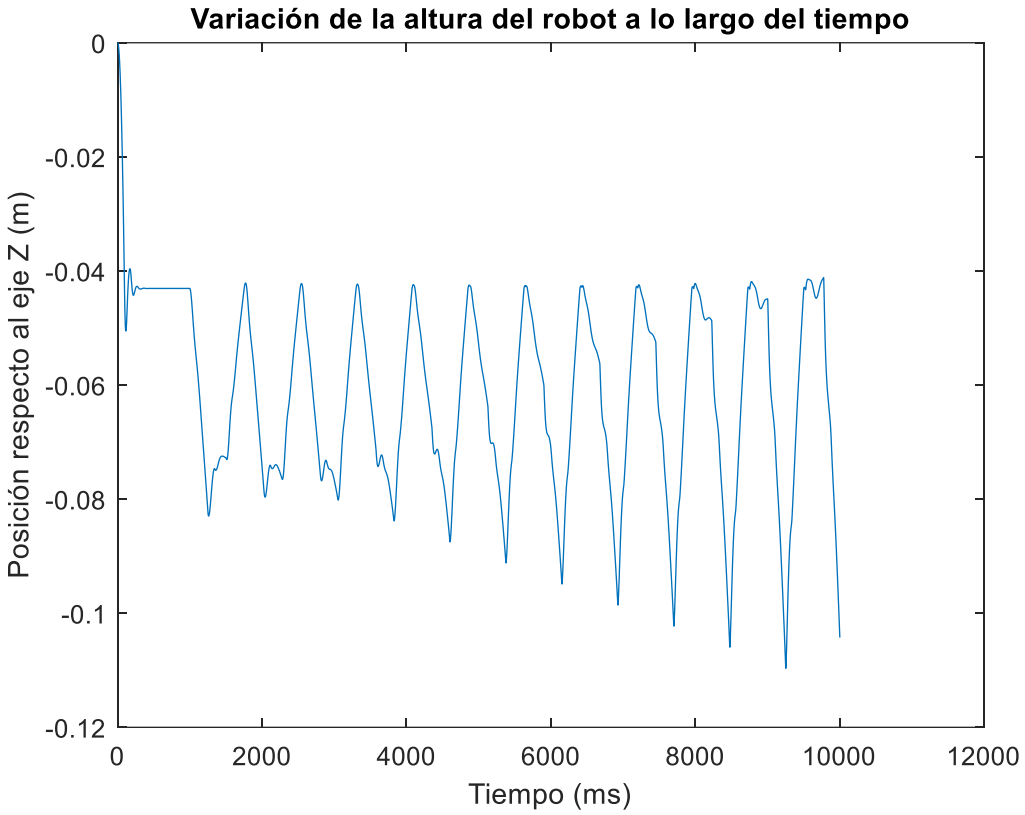


Figura 51: Evolución de la posición vertical del robot durante la simulación.

4. IMPLEMENTACIÓN

4.1. Diseño en SolidWorks

Para la implementación real del robot cuadrúpedo es necesario diseñar las piezas que lo conforman, para ello se utilizó el programa SolidWorks. Dado que no es factible realizar una impresión 3D tan compleja como la de Spot de Boston Dynamics, se optó por buscar diseños CAD más sencillos disponibles en internet. Estos diseños fueron modificados adecuadamente para ajustarse a la idea planteada [15].

El cuerpo del robot está compuesto por dos piezas planas, mientras que cada una de las ocho patas está formada por dos. Además, se requieren cuatro soportes para los servos que corresponden con las articulaciones que unen las patas con el cuerpo.

Tanto el cuerpo como las patas del robot se imprimieron en metacrilato, debido a que su composición plana facilita su fabricación. Por otro lado, los soportes de los servomotores se imprimieron en nylon. Las partes impresas en metacrilato se componen por capas de 5 mm cada una que se han unido entre sí para obtener el grosor diseñado.

A continuación, se mostrará el diseño CAD de cada una de las piezas comentadas. Las operaciones realizadas son bastante sencillas, las más utilizadas han sido Cortar-Extruir, Saliente-Extruir y Redondeo.

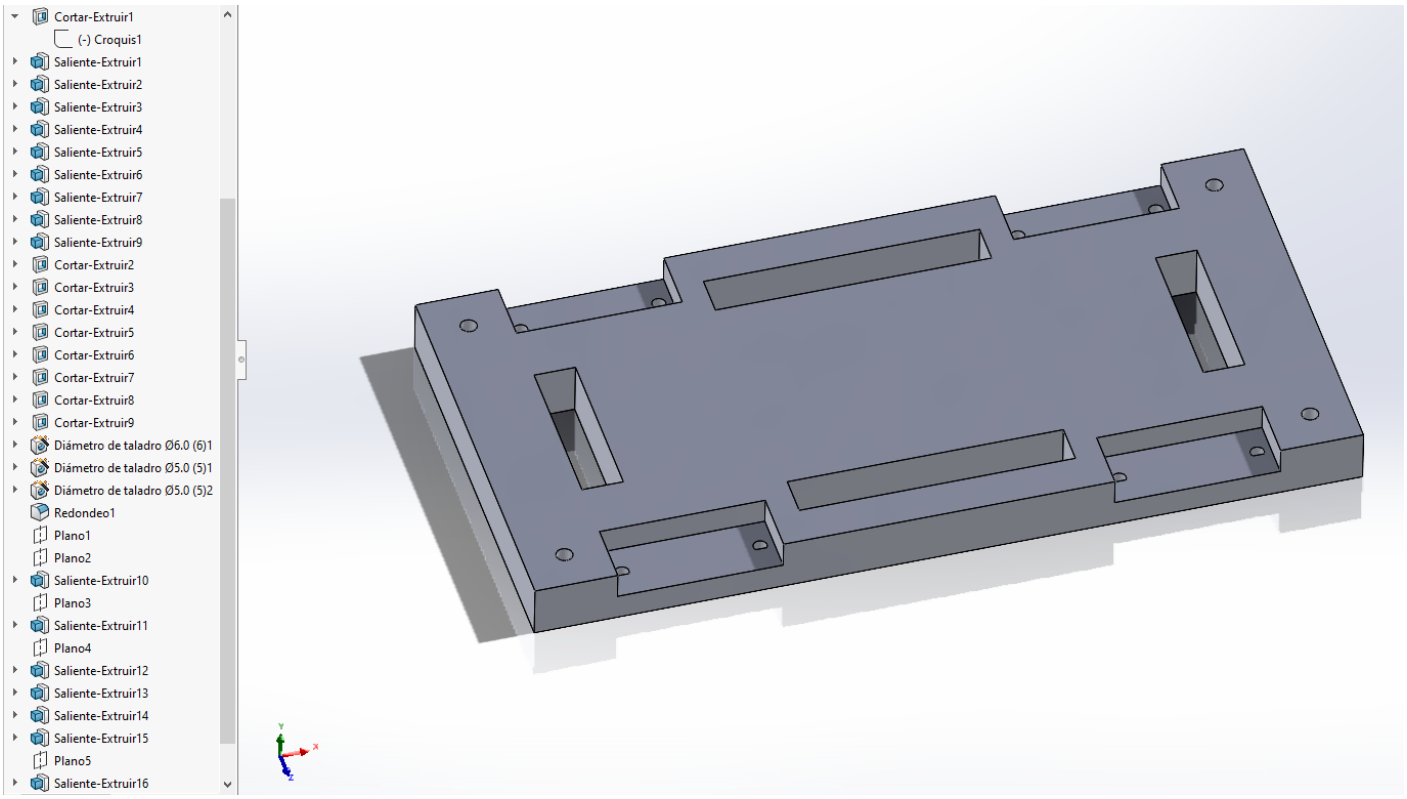


Figura 52: Diseño de la parte inferior del cuerpo.

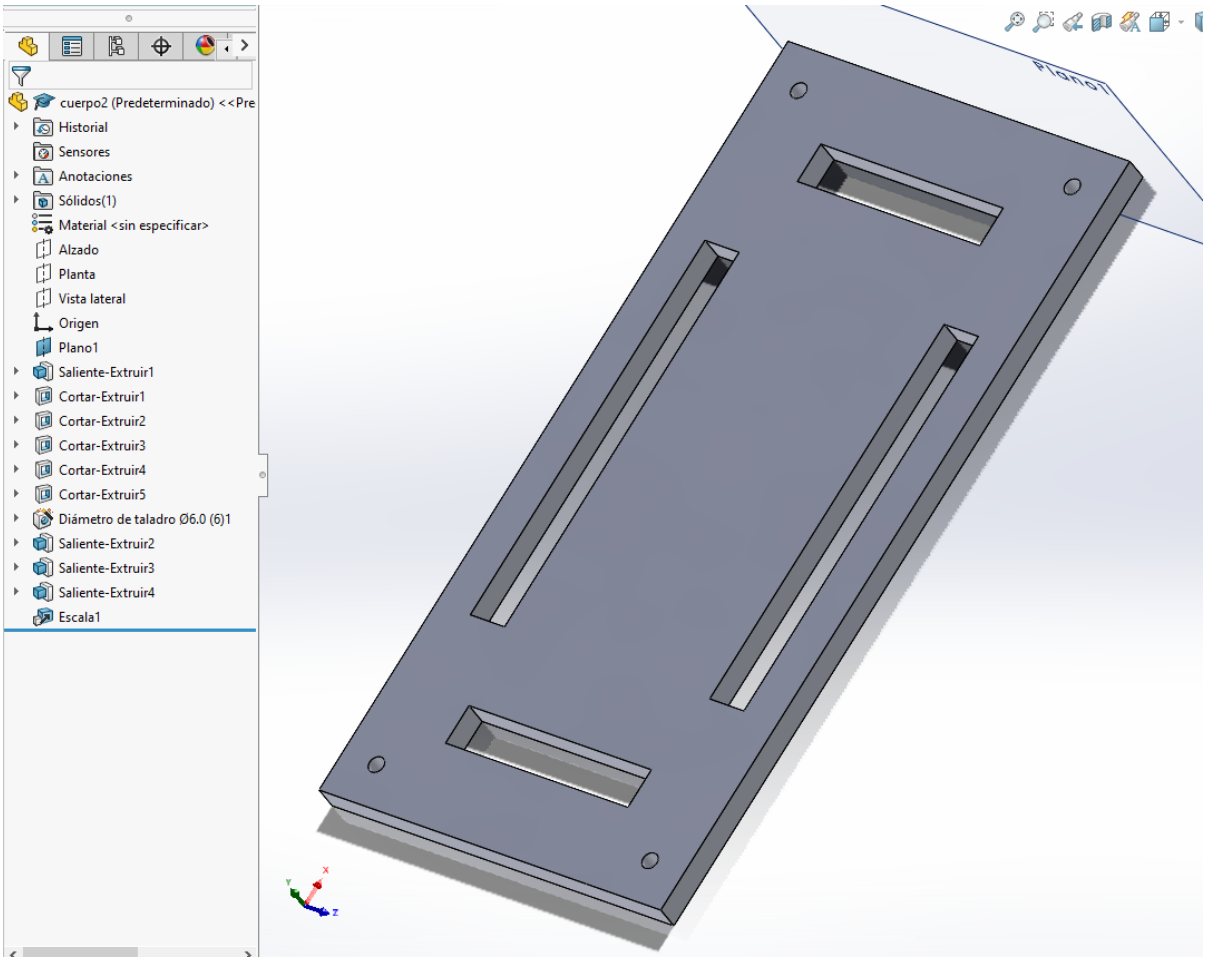


Figura 53: Diseño de la parte inferior del cuerpo.

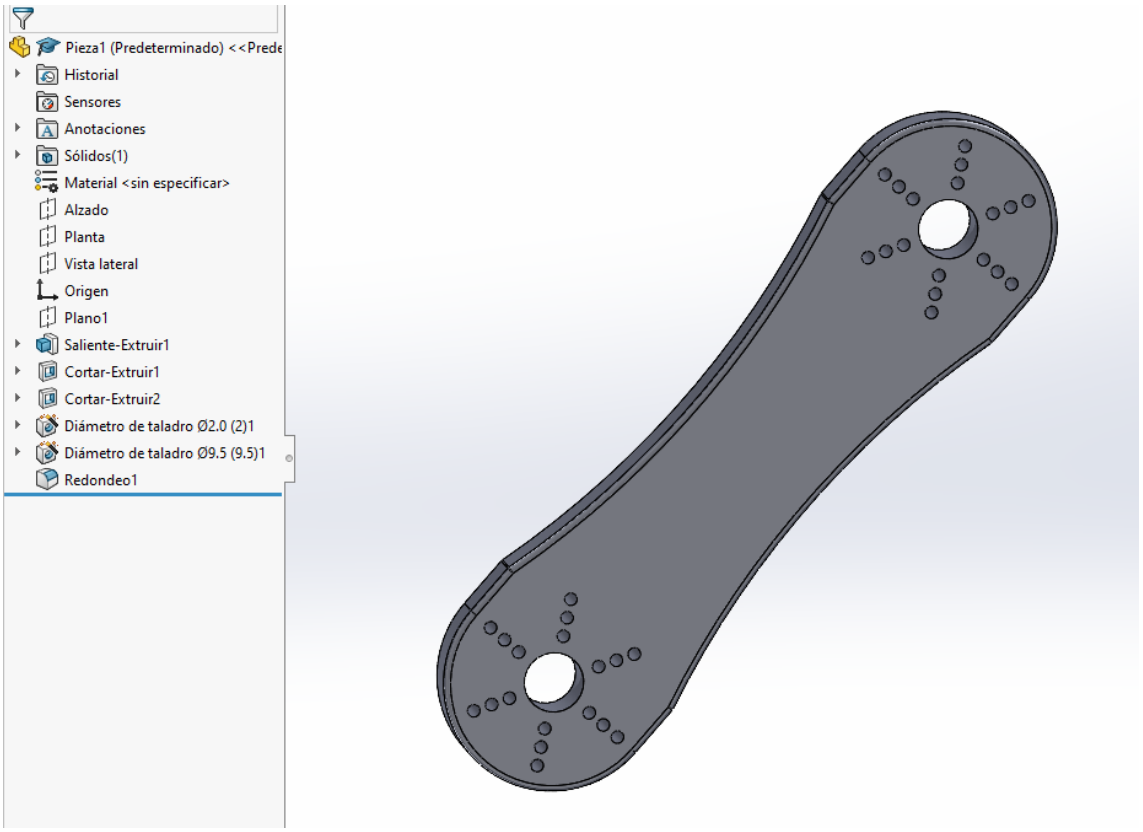


Figura 54: Diseño de la parte superior de las patas.

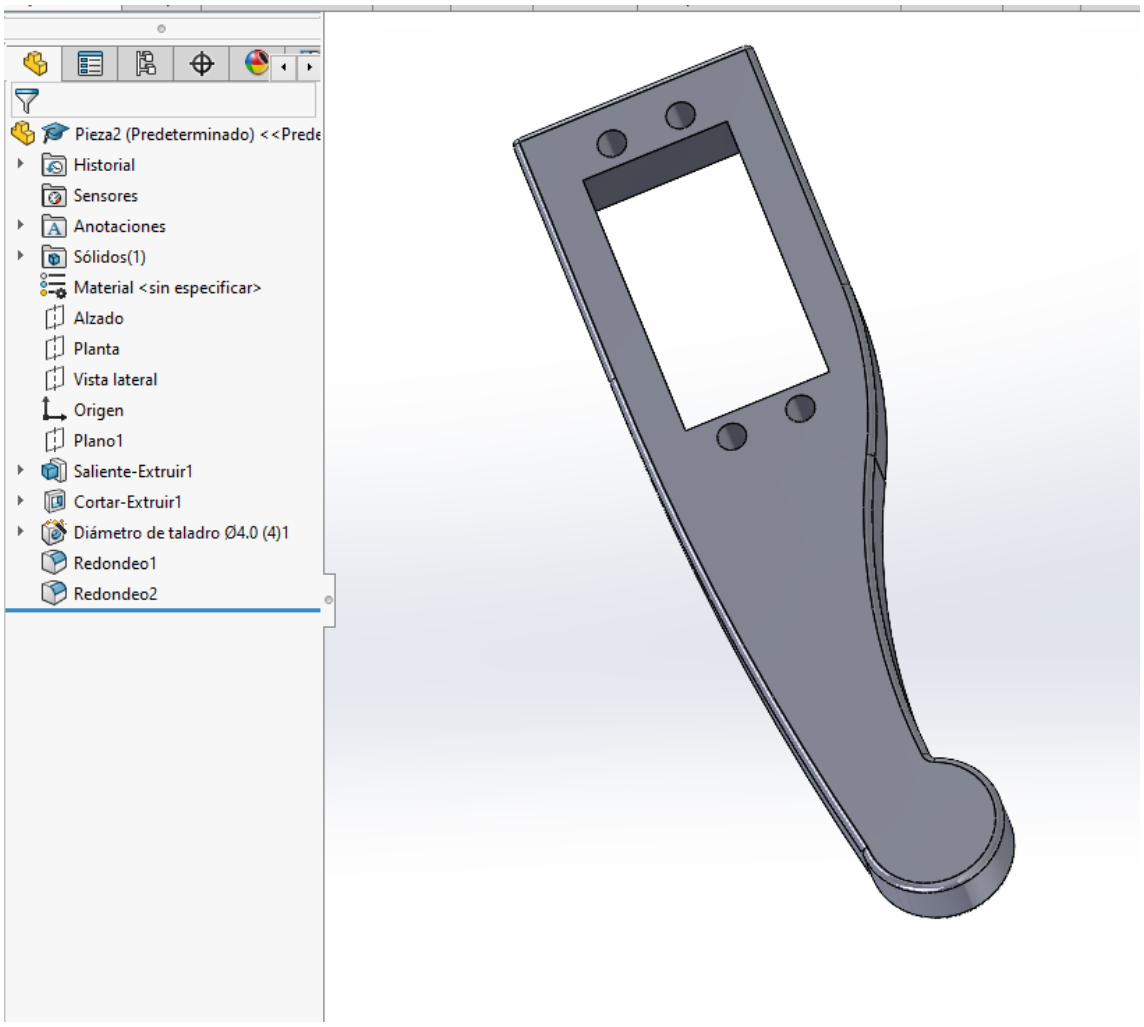


Figura 55: Diseño de la parte inferior de las patas.

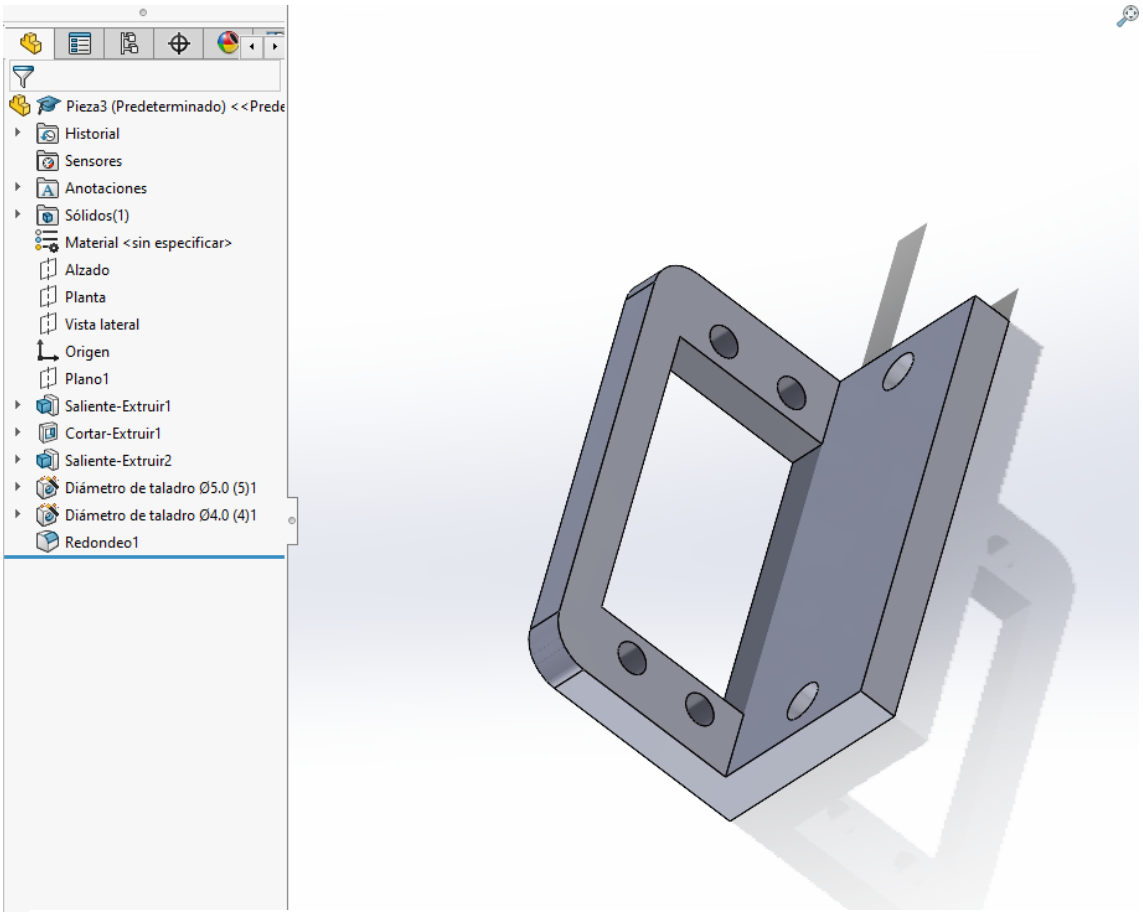


Figura 56: Diseño de la base de los servos.

En las próximas imágenes se muestra cómo quedaron las piezas una vez fueron impresas.

Figura 57: Piezas de las patas impresas.

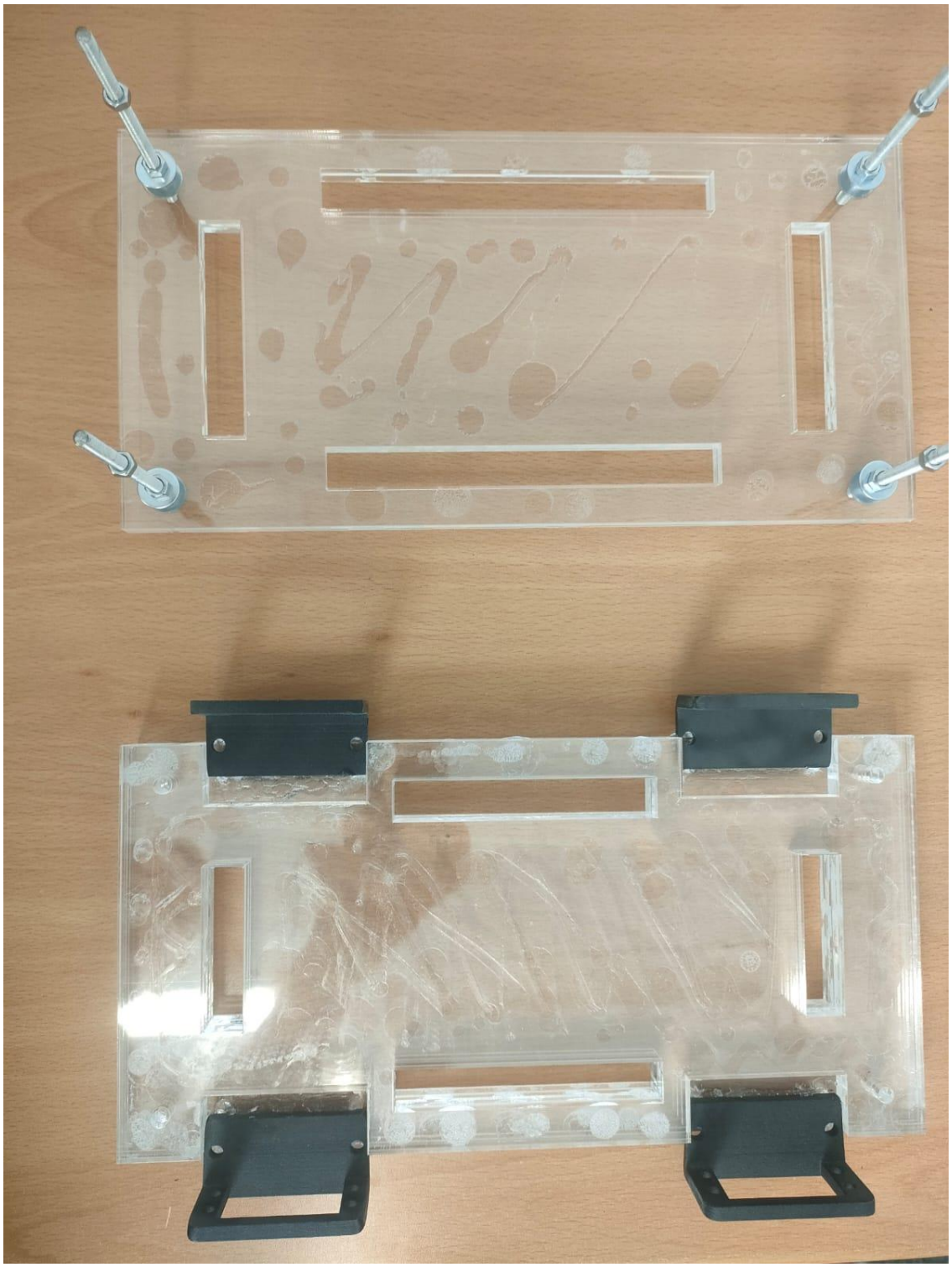


Figura 58: Piezas del cuerpo junto con las bases de los servos impresas.

4.2. Electrónica

Queda por explicar la electrónica que se utilizó para poder realizar la implementación del robot. Para ello se empleó una placa de Arduino Due, compuesta por un microcontrolador ARM de 32 bits, ofreciendo más potencia y capacidad que las basadas en microcontroladores de 8 bits y, además, dispone de una gran cantidad de pines entrada/salida y de interfaces de comunicación, lo que la hace ideal para proyectos que requieren más procesamiento y memoria [4].

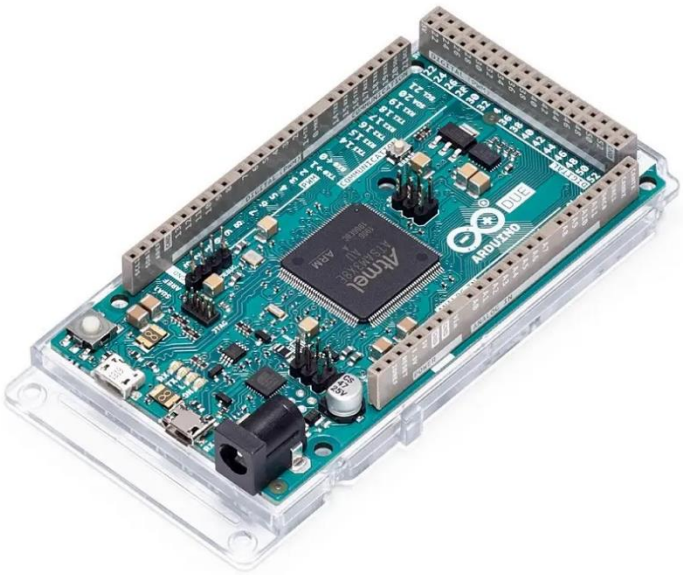


Figura 59: Placa Arduino Due.

No obstante, esta placa presenta limitaciones en la corriente y voltaje que puede suministrar, por lo que no sería posible controlar más de 2 motores con ella simultáneamente. Es por ello por lo que se conectó un controlador PCA9685 de 16 canales, permitiendo aumentar la capacidad de la Arduino Due para controlar los 8 motores del robot [3][14].

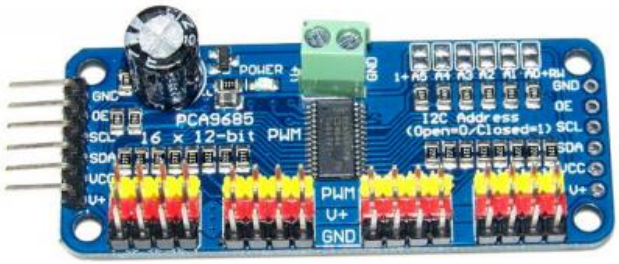


Figura 60: Controlador PCA3685.

https://www.tiendatec.es/maker-zone/robotica/1218-controlador-servo-pwm-16-canales-12-bits-pca9685-iic-i2c-8472496016834.html

El controlador se alimentará a 6V gracias a 4 pilas de 1,5V cada una colocadas en un portapilas. Así quedan los componentes en la vida real conectados entre sí para poder controlar los 8 motores simultáneamente.

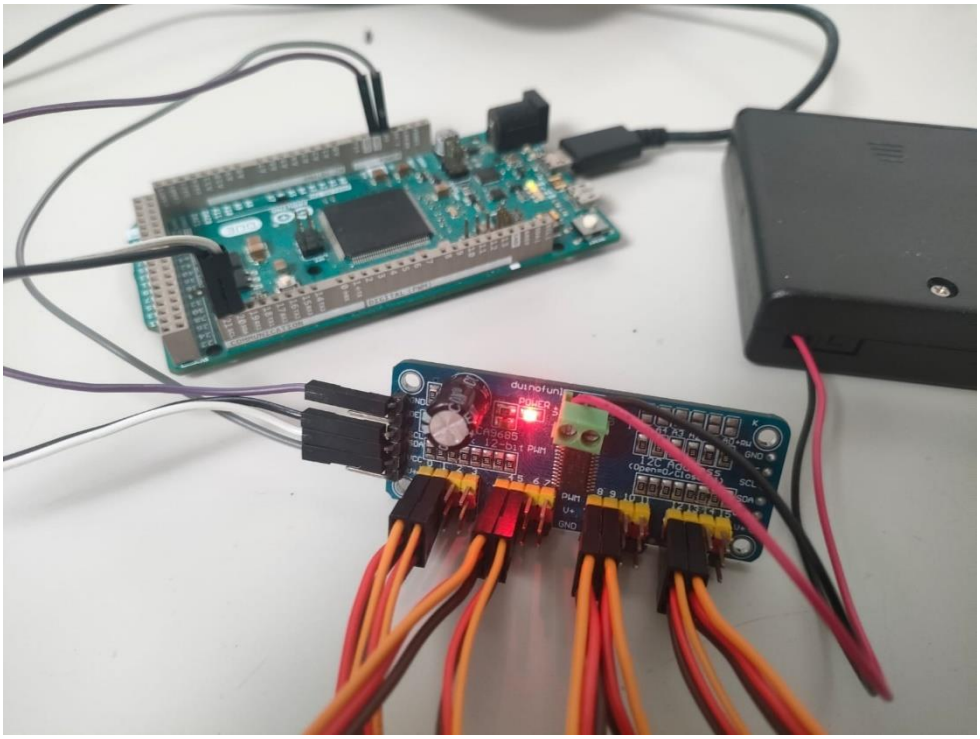


Figura 61: Arduino DUE, PCA3685 y portapilas conectados entre sí.

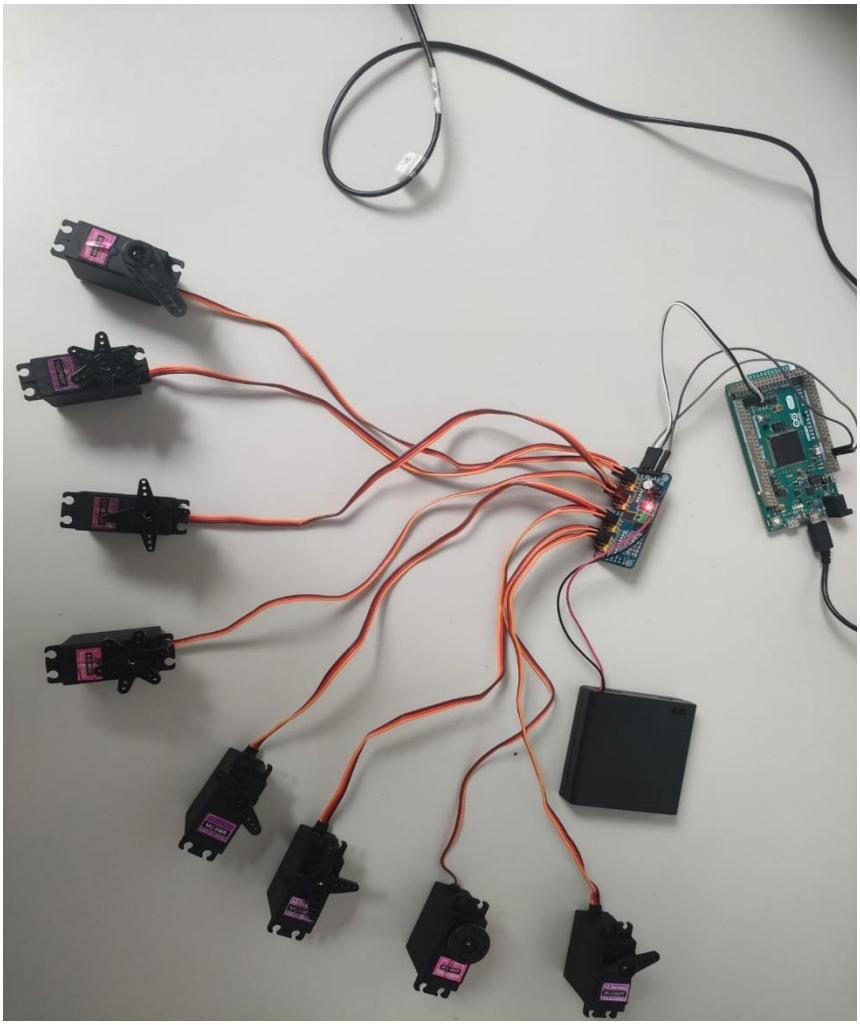


Figura 62: Los 8 motores conectados al controlador PCA3685.

La conexión entre el controlador y el Arduino es bastante sencilla [13][14]:

- El pin GND del controlador se conecta con uno de los pines de GND del apartado de power de Arduino.
- El pin VCC del controlador se conecta a los 5V de Arduino.
- El pin SCL del controlador se conecta al pin SCL1 de Arduino.
- El pin SDA del controlador se conecta al pin SDA1 de Arduino.

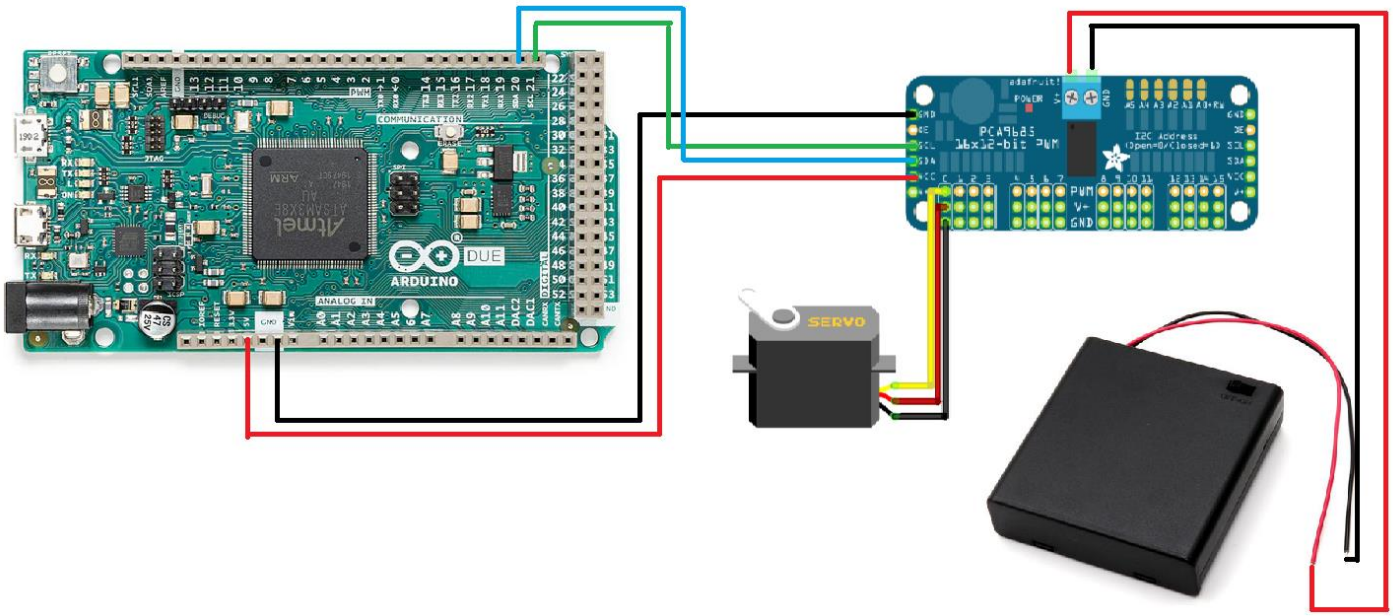


Figura 63: Conexionado Arduino Due - controlador PCA3685 – Porta pilas - Motor

Los motores utilizados son del modelo MG996R, los cuales presentan las siguientes características [16].

- Ofrecen un alto torque, permitiendo mover y levantar las patas del robot de forma eficiente.
- Son muy veloces y precisos, proporcionando una velocidad de respuesta rápida y una alta precisión para los movimientos que requiere un robot cuadrúpedo.
- Debido a su alta durabilidad son capaces de soportar el descaste y las tensiones mecánicas bajo cargas pesadas durante largos períodos de tiempo.
- Funcionan con un rango de voltaje de 4.8V a 7.2V y presentan un ángulo de rotación estándar de 120º, suficiente para las articulaciones del robot.
- Debido a su alta disponibilidad y que son bastante económicos son una buena opción para proyectos de robótica sin incrementar significativamente el presupuesto.

Por todas estas características, este tipo de motores son idóneos para que un robot cuadrúpedo tipo Spot pueda moverse de manera estable y eficiente.

Figura 64: Servo MG996R.

https://hobbyking.com/es_es/towerpro-mg996r-10kg-servo-10kg-0-20sec-55g.html

4.3. Montaje

Para llevar a cabo el montaje del robot han sido necesarios diferentes tipos de tornillos, tuercas, arandelas, clavos y una varilla roscada con métricas estandarizadas. Todos estos componentes están registrados en la Tabla 1.

Tabla 1: Componentes necesarios para el montaje.

Componente	Utilizado en	Cantidad
Varilla roscada M5 x 100 mm	Unión de las partes del cuerpo	4
Tuercas M5	Unión de las partes del cuerpo	16
Arandelas M5	Unión de las partes del cuerpo	16
Clavos 1.5 mm x 20 mm	Sujeción eje del servo con la pata	24
Clavos 1 mm x 20 mm	Sujeción eje del servo con la pata	24
Tornillo M3 x 20 mm	Sujeción eje del servo con la pata	8
Arandela M3	Sujeción eje del servo con la pata	8
Tornillo pasante M5 x 20 mm	Sujeción servos con articulaciones	32
Tuercas M5	Sujeción servos con articulaciones	32

A continuación, se ha llevado a cabo el ensamblaje del robot. Primero, se han lijado las piezas de las bases de los servos para colocarlas con mayor facilidad en los huecos correspondientes de la parte inferior del cuerpo y se colocan los servos. A continuación, se ha juntado la otra parte del cuerpo con ayuda de la varilla roscada. El siguiente paso es montar las patas, primero uniendo la parte de arriba de éstas a los servos ya colocados, y terminar con la parte inferior junto con los otros cuatro servomotores. Para finalizar, se coloca el Arduino, el porta pilas y el controlador sobre el cuerpo, obteniendo como resultado el robot mostrado en la

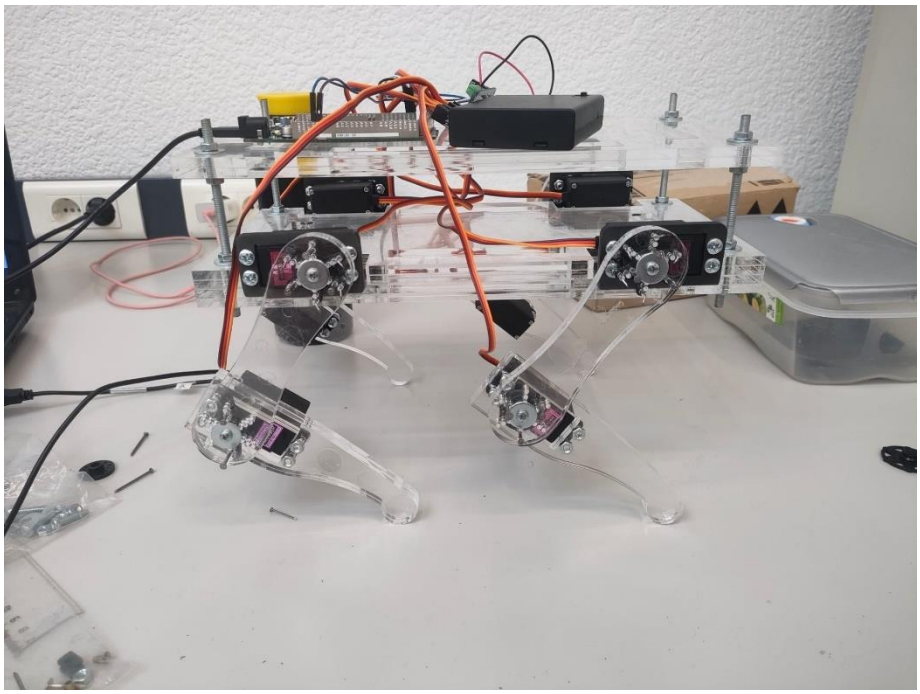


Figura 65: Robot cuadrúpedo una vez montado.

4.4. Programación

Para conseguir que el robot pudiera realizar los movimientos de la simulación al mundo físico se han tenido en cuenta las secuencias empleadas en Simscape de Matlab. Se procederá a explicar por encima el código para entender cómo se controla un robot cuadrúpedo [18][20].

Lo primero es incluir las librerías necesarias para el control de los servomotores, mostradas en la Figura 66. Debido a que se controlan mediante el controlador se ha empleado la librería Wire.h para la comunicación I2C y la Adafruit_PWMServoDriver.h para el propio driver. Se ha declarado el objeto 'servos' y se le ha asignado la dirección 0x40, siendo el valor predeterminado para el controlador PWM de Adafruit.

```
1  #include <Wire.h>
2  #include <Adafruit_PWMServoDriver.h>
3
4  Adafruit_PWMServoDriver servos = Adafruit_PWMServoDriver(0x40);
```

Figura 66: Librerías necesarias para el control de los servos.

A continuación, se definen las variables globales referentes a los anchos de pulso para las posiciones de 0º y 180º de nuestros servos. También se crea el array de posiciones de los servos para almacenar sus posiciones actuales y los array con las posiciones objetivo de estos (Figura 67).

```
6   unsigned int pos0 = 200; // Ancho de pulso en cuentas para posición 0°
7   unsigned int pos180 = 409; // Ancho de pulso en cuentas para la posición 180°
8
9   int pos[8] = {0, 0, 0, 0, 0, 0, 0, 0}; // Array para almacenar las posiciones de los servos
10
11  int targetSet[4][8] = {
12    {-30, -90, 30, 20, 30, 90, -30, -20}, // target 1
13    {0, 10, 0, -5, 0, -10, 0, 5},         // target 2
14    {0, 10, 0, -5, 0, -10, 0, 5},         // target 3
15    {0, 10, 0, -5, 0, -10, 0, 5}          // target 4
16  };
```

Figura 67: Variables globales y arrays de movimientos de los servos.

El siguiente paso es definir las variables de control de tiempo de espera y el cambio entre los diferentes arrays de objetivos de movimientos, así como la declaración de las diferentes funciones (Figura 68).

```
23  int currentTargetSet = 0; // Índice del conjunto de objetivos actual
24  unsigned long previousMillis = 0;
25  unsigned long waitStartMillis = 0;
26  const long interval = 15; // Intervalo de tiempo entre actualizaciones
27  const long waitDurations[4] = {4000, 3000, 3000, 3000}; // Duraciones de espera en milisegundos para cada conjunto de objetivos
28
29  // Declaración de funciones
30  void setServo(uint8_t n_servo, int angulo);
31  int getServoChannel(int index);
32  void moveToPositionSynchronously(int targetAngle);
33  bool moveToTargetSet(int targetSetIndex);
```

Figura 68: Control de tiempos y declaración de funciones.

La Figura 69 muestra la estructura de las diferentes funciones. La función ‘setServo’ configura un servo en un ángulo específico, mapeando el ángulo deseado a un ancho de pulso y configurando el PWM del servo correspondiente. La función ‘getServoChannel’ mapea un índice del servo al número del canal PWM correspondiente, debido a que los canales del servo utilizados son 0 y 1 para la pata delantera derecha, 4 y 5 para la trasera izquierda, 8 y 9 para la delantera derecha y 12 y 13 para la trasera derecha. La función ‘moveToPositionSynchronously’ mueve todos los servos de forma sincronizada a un ángulo calculando los pasos necesarios, ajustando la posición de cada uno de los servos mediante pequeños incrementos. Finalmente, todos los servos se mueven al array objetivo calculando los pasos necesarios gracias a la función ‘moveToTargetSet’.

```cpp
void setServo(uint8_t n_servo, int angulo) {
  int duty = map(angulo, 0, 180, pos0, pos180);
  servos.setPWM(n_servo, 0, duty);
}
```

```cpp
bool moveToTargetSet(int targetSetIndex) {
  bool allServosAtTarget = true;
  int maxSteps = 0;
  int steps[8];

  // Calcular el número máximo de pasos necesario para cualquier servo
  for (int i = 0; i < 8; i++) {
    steps[i] = abs(pos[i] - targetSet[targetSetIndex][i]);
    if (steps[i] > maxSteps) {
      maxSteps = steps[i];
    }
  }

  // Mover todos los servos sincronizadamente
  for (int step = 0; step <= maxSteps; step++) {
    for (int i = 0; i < 8; i++) {
      if (steps[i] > 0) {
        int angle = map(step, 0, maxSteps, pos[i], targetSet[targetSetIndex][i]);
        setServo(getServoChannel(i), angle);
      }
    }
    delay(10); // Ajusta el valor del delay para controlar la velocidad del movim
  }

  // Actualizar las posiciones actuales y verificar si todos los servos están en
  for (int i = 0; i < 8; i++) {
    pos[i] = targetSet[targetSetIndex][i];
    if (pos[i] != targetSet[targetSetIndex][i]) {
      allServosAtTarget = false;
    }
  }
```

```cpp
void moveToPositionSynchronously(int targetAngle) {
  int maxSteps = 0;
  int steps[8];

  // Calcular el número máximo de pasos necesario para cualquier
  for (int i = 0; i < 8; i++) {
    steps[i] = abs(pos[i] - targetAngle);
    if (steps[i] > maxSteps) {
      maxSteps = steps[i];
    }
  }

  // Mover todos los servos sincronizadamente
  for (int step = 0; step <= maxSteps; step++) {
    for (int i = 0; i < 8; i++) {
      if (steps[i] > 0) {
        int angle = map(step, 0, maxSteps, pos[i], targetAngle);
        setServo(getServoChannel(i), angle);
      }
    }
    delay(15); // Ajusta el valor del delay para controlar la vel
  }

  // Actualizar las posiciones actuales
  for (int i = 0; i < 8; i++) {
    pos[i] = targetAngle;
  }
}
```

```cpp
int getServoChannel(int index) {
  switch (index) {
    case 0: return 0;
    case 1: return 1;
    case 2: return 4;
    case 3: return 5;
    case 4: return 8;
    case 5: return 9;
    case 6: return 12;
    case 7: return 13;
    default: return 0;
  }
}
```

Figura 69: Estructura de las funciones utilizadas.

Solamente queda explicar el ‘setup’ y el ‘loop’. En este primero se inicializa el driver y se mueven todos los servos a la posición inicial para que comiencen todos en 0 grados, utilizando una frecuencia PWM de 50 hercios. En el ‘loop’ utiliza la función ‘millis()’ para asegurarse de que el objetivo de los servos se ha alcanzado antes de pasar al siguiente.

```
35  void setup() {
36    servos.begin();
37    servos.setPWMFreq(50); // Frecuencia PWM de 50Hz o T=20ms
38
39    // Mover todos los servos a la posición inicial de forma sincronizada
40    moveToPositionSynchronously(0);
41    delay(5000);
42  }
43
44  void loop() {
45    unsigned long currentMillis = millis();
46
47    if (currentMillis - previousMillis >= interval) {
48      previousMillis = currentMillis;
49
50      if (moveToTargetSet(currentTargetSet)) {
51        // Esperar la duración especificada para el conjunto de objetivos actual
52        if (waitStartMillis == 0) {
53          waitStartMillis = currentMillis;
54        }
55
56        if (currentMillis - waitStartMillis >= waitDurations[currentTargetSet]) {
57          waitStartMillis = 0;
58          currentTargetSet++;
59          if (currentTargetSet >= 4) {
60            currentTargetSet = 0;
61          }
62        }
63      }
64    }
65  }
```

Figura 70: Setup y loop del código.

4.5. Modificaciones del montaje del robot

Una vez terminado el código y el montaje del robot se procedió a comprobar si era capaz de realizar los movimientos correctamente. El robot era capaz de mantenerse en pie, pero al mínimo movimiento que se le programaba, los motores perdían fuerza y el robot caía y quedaba tumbado en el suelo. En primera instancia se pensó que debido a que su cuerpo tenía demasiado peso y los servos no tenían un par lo suficientemente alto como para poder soportarlo.

Es por ello por lo que se procedió a realizar un agujero cuadrado en el centro de ambas partes del cuerpo para, de este modo, quitarle material innecesario y, por ende, peso. Una vez el corte estaba hecho se procedió a montar el robot con únicamente la parte inferior del cuerpo para que tampoco tuviera que soportar su parte superior. El robot una vez montado son estas modificaciones quedó de la siguiente manera pudiéndose observar en la Figura 71 cómo queda de perfil únicamente con la parte inferior del cuerpo y en la Figura 72 cómo quedó el cuerpo con el agujero.

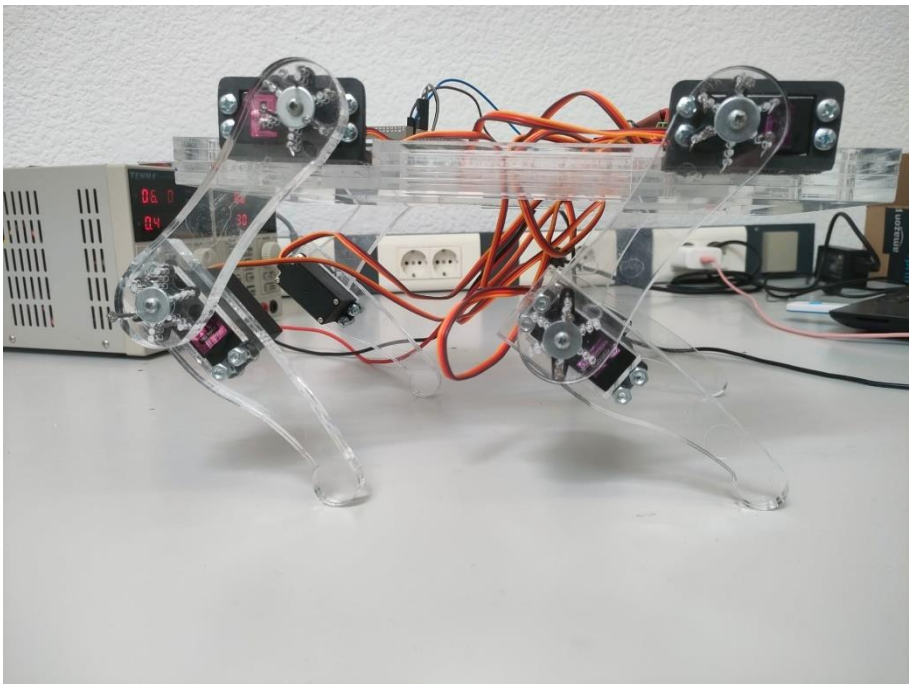


Figura 71; Robot una vez realzadas las modificaciones.

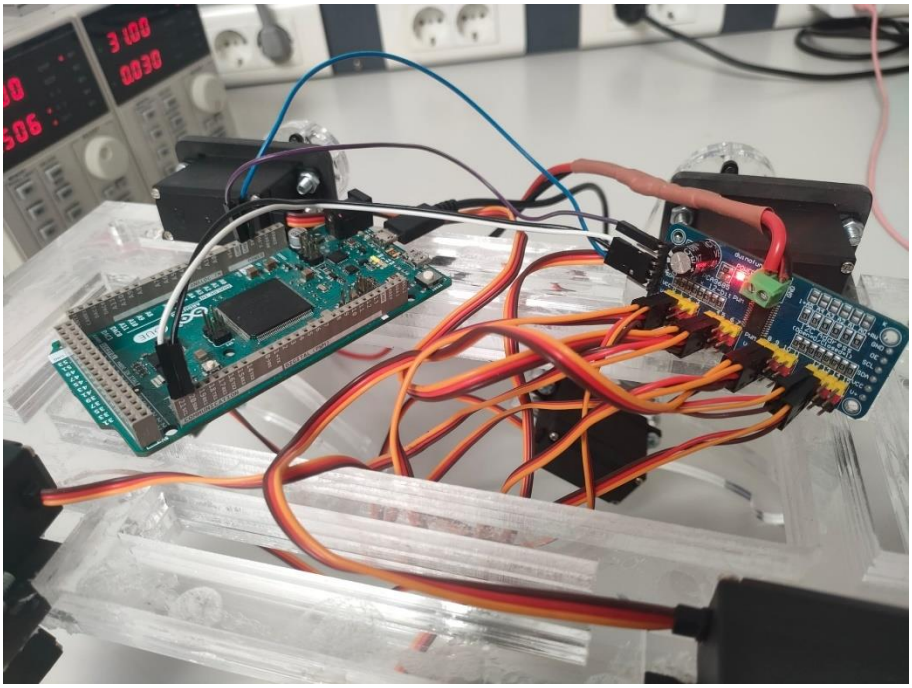


Figura 72: Agujero realizado en el cuerpo para quitar peso.

No obstante, estas modificaciones no sirvieron para que el robot pudiera realizar los movimientos pues, de nuevo, los motores no tenían suficiente fuerza. Una vez llegado a este punto solo quedaba modificar la forma en la que se estaba alimentando el controlador ya que es posible que las pilas de 1.5V no tuvieran suficiente capacidad como para otorgar la corriente necesaria para el movimiento de los servomotores.

La solución que se llevó a cabo es alimentar el driver con una fuente de alimentación. Puede observarse también en la Figura 72, otorgando una intensidad tope de 2 amperios.

Afortunadamente, tras este último cambio el robot ya era capaz de caminar, por lo que el problema principal era la forma en la que se alimentaba el circuito.

5. CONCLUSIONES Y TRABAJO FUTURO

Se han cumplido la mayoría de los objetivos planteados en el proyecto de forma adecuada. Mediante Matlab/Simulink, Simscape Multibody se ha conseguido realizar una simulación del robot cuadrúpedo Spot de Boston Dynamics. Este proceso ha permitido comprender y replicar su comportamiento en un entorno controlado, obteniendo gráficas detalladas que muestran parámetros como sus posiciones y ángulos de cabeceo y balanceo en hasta nueve simulaciones diferentes, realizando movimientos básicos como sentarse o bajar el cuerpo, caminando en sus dos formas de caminar más características, trot y crawl, evaluando su capacidad de sortear obstáculos como rampas, escalones y baches y mostrando su habilidad de caminar de lado, completando todos los tipos de movimientos que se tenían pensados desde el principio.

También se ha completado la implementación real del robot cuadrúpedo, siendo necesario el diseño de un modelo simplificado de Spot, dada la complejidad de su diseño original, realizando la impresión en 3D de las piezas. Durante el montaje, se han enfrentado y superado diversos inconvenientes que ralentizaron la finalización del trabajo. El principal desafío fue el peso del cuerpo del robot, que se resolvió mediante la realización de cortes para eliminar material sobrante, reduciendo el peso de forma significativa.

El otro principal desafío fue la insuficiencia de alimentación con el porta pilas, lo que obligó a utilizar una fuente de alimentación, permitiendo que los servos tuvieran la suficiente potencia para asegurar el correcto funcionamiento del robot durante los movimientos.

Los conocimientos adquiridos a lo largo del máster en las materias de Control Aplicado de Sistemas Mecatrónicos, Robótica y Sistemas de Medición y Actuación han sido imprescindibles para la realización de este proyecto, proporcionando una base sólida para abordar tanto la simulación como la implementación física del robot.

Sin embargo, no pudieron completarse todos los objetivos inicialmente planteados. El control remoto mediante una aplicación Android no se ha podido implementar debido a la falta de tiempo. Estaba planteado como una interfaz sencilla que permitiera elegir la forma de caminar deseada y hacer que el robot avanzara o quedara quieto. Este objetivo podría plantearse como trabajo futuro.

Por otro lado, también sería interesante plantear como próximo objetivo diseñar otro robot más pequeño o ligero, permitiendo explorar la viabilidad de utilizar una forma de alimentación portátil, como el porta pilas que se utilizó inicialmente, ya que la fuente de alimentación limita considerablemente la movilidad del robot.

En conclusión, pese a que el proyecto ha cumplido la gran mayoría de los objetivos planteados y ha superado diversos desafíos, principalmente en la etapa de montaje, también ha dejado la puerta abierta a futuras mejoras, completando aspectos pendientes como el control remoto y permitiendo explicar el potencial de los robots cuadrúpedos a partir de la base obtenida.

6. BIBLIOGRAFÍA

1. *(48) How Boston Dynamics' Spot Robot Works! - YouTube*. (s. f.). Recuperado 24 de junio de 2024, de https://www.youtube.com/watch?v=R-PdPtqw78k
2. *About the Spot robot*. (s. f.). Recuperado 24 de junio de 2024, de https://support.bostondynamics.com/s/article/About-the-Spot-robot
3. *Adafruit PCA9685 16-Channel Servo Driver*. (s. f.). Adafruit Learning System. Recuperado 24 de junio de 2024, de https://learn.adafruit.com/16-channel-pwm-servo-driver/overview
4. *Arduino Due*. (s. f.). Arduino Official Store. Recuperado 24 de junio de 2024, de https://store.arduino.cc/products/arduino-due
5. Boston Dynamics (Director). (2019, septiembre 24). *Spot Launch*. https://www.youtube.com/watch?v=wlkCQXHEgjA
6. Boston Dynamics. (2024). En *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Boston_Dynamics&oldid=1226731014#Spot
7. *Boston Dynamics | Soluciones robóticas para la industria*. (s. f.). Plain Concepts. Recuperado 24 de junio de 2024, de https://www.plainconcepts.com/es/spot-boston-dynamics-robot/
8. *Coefficient of Friction Equation and Table Chart*. (s. f.). Recuperado 24 de junio de 2024, de https://www.engineersedge.com/coeffients_of_friction.htm
9. *El robot Spot Mini | GrupoADD*. (2019, septiembre 15). https://grupoadd.es/el-robot-spot-mini, https://grupoadd.es/el-robot-spot-mini
10. *Get Started with Simscape Multibody—MathWorks España*. (s. f.). Recuperado 24 de junio de 2024, de https://es.mathworks.com/help/sm/getting-started-with-simmechanics.html

11. Invelon. (s. f.). *ROBOTS CUADRÚPEDOS | Invelon Technologies*. invelon. Recuperado 24 de junio de 2024, de https://www.invelon.com/robots-cuadrupedos/
12. Lesics (Director). (2022, diciembre 23). *Boston Dynamics Spot Robot | All of its Engineering SECRETS!* https://www.youtube.com/watch?v=tfWbE_1eCZk
13. Llamas, L. (2016, noviembre 24). *Controlar 16 servos o 16 salidas PWM en Arduino con PCA9685*. Luis Llamas. https://www.luisllamas.es/controlar-16-servos-o-16-salidas-pwm-en-arduino-con-pca9685/
14. PCA9685 16 Canales PWM I2C 12-Bit. (s. f.). *UNIT Electronics*. Recuperado 24 de junio de 2024, de https://uelectronics.com/producto/pca9685-16-canales-pwm-i2c-12-bit/
15. *Quadruped Robot Dog | 3D CAD Model Library | GrabCAD*. (s. f.). Recuperado 24 de junio de 2024, de https://grabcad.com/library/quadruped-robot-dog-2
16. *Servo MG996R, un motor de alta potencia y calidad superior*. (s. f.). ElectroHobby. Recuperado 24 de junio de 2024, de https://www.electrohobby.es/servo/370-servo-mg996r.html
17. *Spot BostonDynamics | 3D CAD Model Library | GrabCAD*. (s. f.). Recuperado 24 de junio de 2024, de https://grabcad.com/library/spot-bostondynamics-1
18. *Tutorial Módulo Controlador de servos PCA9685 con Arduino*. (s. f.). Naylamp Mechatronics - Perú. Recuperado 24 de junio de 2024, de https://naylampmechatronics.com/blog/41_tutorial-modulo-controlador-de-servos-pca9685-con-arduino.html
19. VentureX - Future Tech (Director). (2021, mayo 29). *Robot Dog Spot: What Futuristic Things Can it ACTUALLY Do? (Boston Dynamics)*. https://www.youtube.com/watch?v=mqDncPrTI2w
20. Zacetrex Technologies (Director). (2019, octubre 26). *Arduino + Driver PCA9685*. https://www.youtube.com/watch?v=DBjzcNvAG_4

7. ANEXOS

ANEXO I. RELACIÓN DEL TRABAJO CON LOS OBJETIVOS DE DESARROLLO SOSTENIBLE DE LA AGENDA 2030

Anexo al Trabajo de Fin de Grado y Trabajo de Fin de Máster: Relación del trabajo con los Objetivos de Desarrollo Sostenible de la agenda 2030

En la siguiente tabla se muestra la relación de este trabajo con los Objetivos de Desarrollo Sostenible (ODS) de la agenda 2030.

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

Tabla 2: Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS)

El proyecto se relaciona principalmente con el ODS 9: “Industria, Innovación e Infraestructura” debido a la creación de un modelo de simulación avanzado e implementación física de un robot cuadrúpedo, promoviendo la innovación en robótica, así como el uso de Matlab/Simscape Multibody y componentes accesibles como servomotores y placas Arduino.

También contribuye, en menor medida, en el ODS 4: “Educación de Calidad” al proporcionar un proyecto educativo avanzado y en el ODS 8: “Trabajo Decente y Crecimiento Económico” al promover empleo de calidad, fomentar el emprendimiento, y mejorar la eficiencia en sectores industriales, impulsando el crecimiento económico.

ANEXO II. CÓDIGO EN ARDUINO

1. El robot se sienta

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

Adafruit_PWMServoDriver servos = Adafruit_PWMServoDriver(0x40);

unsigned int pos0 = 200; // Ancho de pulso en cuentas para posición 0°
unsigned int pos180 = 409; // Ancho de pulso en cuentas para la posición 180°

int pos[8] = {0, 0, 0, 0, 0, 0, 0, 0}; // Array para almacenar las posiciones de los servos

int targetSet[4][8] = {
  {-30, -90, 30, 20, 30, 90, -30, -20}, // target 1
  {0, 10, 0, -5, 0, -10, 0, 5},         // target 2
  {0, 10, 0, -5, 0, -10, 0, 5},         // target 3
  {0, 10, 0, -5, 0, -10, 0, 5}          // target 4
};

  // PATA DELANTERA IZQUIERDA: Servos 0 y 1
  // PATA TRASERA IZQUIERDA: Servos 4 y 5
  // PATA DELANTERA DERECHA: Servos 8 y 9
  // PATA TRASERA DERECHA: Servos 12 y 13

int currentTargetSet = 0; // Índice del conjunto de objetivos actual
unsigned long previousMillis = 0;
unsigned long waitStartMillis = 0;
const long interval = 15; // Intervalo de tiempo entre actualizaciones
const long waitDurations[4] = {4000, 3000, 3000, 3000}; // Duraciones de espera en milisegundos para cada conjunto de objetivos

// Declaración de funciones
void setServo(uint8_t n_servo, int angulo);
int getServoChannel(int index);
void moveToPositionSynchronously(int targetAngle);
bool moveToTargetSet(int targetSetIndex);

void setup() {
  servos.begin();
```

```
38.  servos.setPWMFreq(50); // Frecuencia PWM de 50Hz o T=20ms
39.
40.  // Mover todos los servos a la posición inicial de forma
   sincronizada
41.  moveToPositionSynchronously(0);
42.  delay(5000);
43.}
44.
45.void loop() {
46.  unsigned long currentMillis = millis();
47.
48.  if (currentMillis - previousMillis >= interval) {
49.    previousMillis = currentMillis;
50.
51.    if (moveToTargetSet(currentTargetSet)) {
52.      // Esperar la duración especificada para el conjunto de
   objetivos actual
53.      if (waitStartMillis == 0) {
54.        waitStartMillis = currentMillis;
55.      }
56.
57.      if (currentMillis - waitStartMillis >=
   waitDurations[currentTargetSet]) {
58.        waitStartMillis = 0;
59.        currentTargetSet++;
60.        if (currentTargetSet >= 4) {
61.          currentTargetSet = 0;
62.        }
63.      }
64.    }
65.  }
66.}
67.
68.void setServo(uint8_t n_servo, int angulo) {
69.  int duty = map(angulo, 0, 180, pos0, pos180);
70.  servos.setPWM(n_servo, 0, duty);
71.}
72.
73.bool moveToTargetSet(int targetSetIndex) {
74.  bool allServosAtTarget = true;
75.  int maxSteps = 0;
76.  int steps[8];
77.
78.  // Calcular el número máximo de pasos necesario para cualquier
   servo
79.  for (int i = 0; i < 8; i++) {
80.    steps[i] = abs(pos[i] - targetSet[targetSetIndex][i]);
81.    if (steps[i] > maxSteps) {
82.      maxSteps = steps[i];
```

```
    }
  }

  // Mover todos los servos sincronizadamente
  for (int step = 0; step <= maxSteps; step++) {
    for (int i = 0; i < 8; i++) {
      if (steps[i] > 0) {
        int angle = map(step, 0, maxSteps, pos[i],
targetSet[targetSetIndex][i]);
        setServo(getServoChannel(i), angle);
      }
    }
    delay(10); // Ajusta el valor del delay para controlar la
velocidad del movimiento
  }

  // Actualizar las posiciones actuales y verificar si todos los
servos están en sus objetivos
  for (int i = 0; i < 8; i++) {
    pos[i] = targetSet[targetSetIndex][i];
          if (pos[i] != targetSet[targetSetIndex][i]) {
            allServosAtTarget = false;
          }
        }

        return allServosAtTarget;
      }

      void moveToPositionSynchronously(int targetAngle) {
        int maxSteps = 0;
        int steps[8];

        // Calcular el número máximo de pasos necesario para
cualquier servo
        for (int i = 0; i < 8; i++) {
          steps[i] = abs(pos[i] - targetAngle);
          if (steps[i] > maxSteps) {
            maxSteps = steps[i];
          }
        }

        // Mover todos los servos sincronizadamente
        for (int step = 0; step <= maxSteps; step++) {
          for (int i = 0; i < 8; i++) {
            if (steps[i] > 0) {
              int angle = map(step, 0, maxSteps, pos[i],
targetAngle);
              setServo(getServoChannel(i), angle);
            }
```

```
127.          }
128.          delay(15); // Ajusta el valor del delay para controlar la
   velocidad del movimiento
129.        }
130.
131.        // Actualizar las posiciones actuales
132.        for (int i = 0; i < 8; i++) {
133.          pos[i] = targetAngle;
134.        }
135.      }
136.
137.      int getServoChannel(int index) {
138.        switch (index) {
139.          case 0: return 0;
140.          case 1: return 1;
141.          case 2: return 4;
142.          case 3: return 5;
143.          case 4: return 8;
144.          case 5: return 9;
145.          case 6: return 12;
146.          case 7: return 13;
147.          default: return 0;
148.        }
149.      }
150.
```

2. Crawl

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

Adafruit_PWMServoDriver servos = Adafruit_PWMServoDriver(0x40);

unsigned int pos0 = 200; // Ancho de pulso en cuentas para posición 0°
unsigned int pos180 = 409; // Ancho de pulso en cuentas para la posición
180°

int pos[8] = {0, 0, 0, 0, 0, 0, 0, 0}; // Array para almacenar las
posiciones de los servos

int targetSet[5][8] = {
  {10,  -8,  20,   -5,   -10,    15,   0,   20}, // target 0
  {0,  -12,  10, -8,    -20,    5,   -10,   40}, // target 1
  {10, -15,   0,   -12,   -10,    8,  -10,   40},   // target 2
```

```
  {10, -15,  10,   -40,    0,    12,  -20,   15},    // target 3
  {20, -5,  10,   -40,   -10,    15,  -10,  15}   // target 4
};
int currentTargetSet = 0; // Índice del conjunto de objetivos actual
unsigned long previousMillis = 0;
unsigned long waitStartMillis = 0;
const long interval = 15; // Intervalo de tiempo entre actualizaciones
const long waitDurations[5] = {0, 0, 0, 0, 0}; // Duraciones de espera en
milisegundos para cada conjunto de objetivos

// Declaración de funciones
void setServo(uint8_t n_servo, int angulo);
int getServoChannel(int index);
void moveToPositionSynchronously(int targetAngle);
bool moveToTargetSet(int targetSetIndex);

void setup() {
  servos.begin();
  servos.setPWMFreq(50); // Frecuencia PWM de 50Hz o T=20ms

  // Mover todos los servos a la posición inicial de forma sincronizada
  moveToPositionSynchronously(0);
  delay(5000);
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    if (moveToTargetSet(currentTargetSet)) {
      // Esperar la duración especificada para el conjunto de objetivos
actual
      if (waitStartMillis == 0) {
        waitStartMillis = currentMillis;
      }

      if (currentMillis - waitStartMillis >=
waitDurations[currentTargetSet]) {
        waitStartMillis = 0;
        currentTargetSet++;
        if (currentTargetSet >= 4) {
          currentTargetSet = 0;
        }
      }
    }
  }
}
```

```
void setServo(uint8_t n_servo, int angulo) {
  int duty = map(angulo, 0, 180, pos0, pos180);
  servos.setPWM(n_servo, 0, duty);
}

bool moveToTargetSet(int targetSetIndex) {
  bool allServosAtTarget = true;
  int maxSteps = 0;
  int steps[8];

  // Calcular el número máximo de pasos necesario para cualquier servo
  for (int i = 0; i < 8; i++) {
    steps[i] = abs(pos[i] - targetSet[targetSetIndex][i]);
    if (steps[i] > maxSteps) {
      maxSteps = steps[i];
    }
  }

  // Mover todos los servos sincronizadamente
  for (int step = 0; step <= maxSteps; step++) {
    for (int i = 0; i < 8; i++) {
      if (steps[i] > 0) {
        int angle = map(step, 0, maxSteps, pos[i],
targetSet[targetSetIndex][i]);
        setServo(getServoChannel(i), angle);
      }
    }
    delay(5); // Ajusta el valor del delay para controlar la velocidad
del movimiento
  }

  // Actualizar las posiciones actuales y verificar si todos los servos
están en sus objetivos
  for (int i = 0; i < 8; i++) {
    pos[i] = targetSet[targetSetIndex][i];
    if (pos[i] != targetSet[targetSetIndex][i]) {
      allServosAtTarget = false;
    }
  }

  return allServosAtTarget;
}

void moveToPositionSynchronously(int targetAngle) {
  int maxSteps = 0;
  int steps[8];

  // Calcular el número máximo de pasos necesario para cualquier servo
```

```
  for (int i = 0; i < 8; i++) {
    steps[i] = abs(pos[i] - targetAngle);
    if (steps[i] > maxSteps) {
      maxSteps = steps[i];
    }
  }

  // Mover todos los servos sincronizadamente
  for (int step = 0; step <= maxSteps; step++) {
    for (int i = 0; i < 8; i++) {
      if (steps[i] > 0) {
        int angle = map(step, 0, maxSteps, pos[i], targetAngle);
        setServo(getServoChannel(i), angle);
      }
    }
    delay(5); // Ajusta el valor del delay para controlar la velocidad
del movimiento
  }

  // Actualizar las posiciones actuales
  for (int i = 0; i < 8; i++) {
    pos[i] = targetAngle;
  }
}

int getServoChannel(int index) {
  switch (index) {
    case 0: return 0;
    case 1: return 1;
    case 2: return 4;
    case 3: return 5;
    case 4: return 8;
    case 5: return 9;
    case 6: return 12;
    case 7: return 13;
    default: return 0;
  }
}
```

3. Trot

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

Adafruit_PWMServoDriver servos = Adafruit_PWMServoDriver(0x40);

unsigned int pos0 = 200; // Ancho de pulso en cuentas para posición 0°
unsigned int pos180 = 409; // Ancho de pulso en cuentas para la posición 180°

int pos[8] = {0, 0, 0, 0, 0, 0, 0, 0}; // Array para almacenar las posiciones de los servos

int targetSet[4][8] = {
  {5,   -12.5, 10,  -20, -10, -10,  -5,   55}, // target 1
  {10,  -25,  -20,  -40,  10,  5,  -10,  60},   // target 2
  {10,   -5,  -10,  -55,  0,  15, -10,  30},    // target 3
  {-10, -10,   10, -60, -10,  25,  20,   45}   // target 4
};
int currentTargetSet = 0; // Índice del conjunto de objetivos actual
unsigned long previousMillis = 0;
unsigned long waitStartMillis = 0;
const long interval = 15; // Intervalo de tiempo entre actualizaciones
const long waitDurations[4] = {0, 0, 0, 0}; // Duraciones de espera en milisegundos para cada conjunto de objetivos

// Declaración de funciones
void setServo(uint8_t n_servo, int angulo);
int getServoChannel(int index);
void moveToPositionSynchronously(int targetAngle);
bool moveToTargetSet(int targetSetIndex);

void setup() {
  servos.begin();
  servos.setPWMFreq(50); // Frecuencia PWM de 50Hz o T=20ms

  // Mover todos los servos a la posición inicial de forma sincronizada
  moveToPositionSynchronously(0);
  delay(5000);
}

void loop() {
  unsigned long currentMillis = millis();

```

```
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    if (moveToTargetSet(currentTargetSet)) {
      // Esperar la duración especificada para el conjunto de objetivos actual
      if (waitStartMillis == 0) {
        waitStartMillis = currentMillis;
      }

      if (currentMillis - waitStartMillis >= waitDurations[currentTargetSet]) {
        waitStartMillis = 0;
        currentTargetSet++;
        if (currentTargetSet >= 4) {
          currentTargetSet = 0;
        }
      }
    }
  }
}

void setServo(uint8_t n_servo, int angulo) {
  int duty = map(angulo, 0, 180, pos0, pos180);
  servos.setPWM(n_servo, 0, duty);
}

bool moveToTargetSet(int targetSetIndex) {
  bool allServosAtTarget = true;
  int maxSteps = 0;
  int steps[8];

  // Calcular el número máximo de pasos necesario para cualquier servo
  for (int i = 0; i < 8; i++) {
    steps[i] = abs(pos[i] - targetSet[targetSetIndex][i]);
    if (steps[i] > maxSteps) {
      maxSteps = steps[i];
    }
  }

  // Mover todos los servos sincronizadamente
  for (int step = 0; step <= maxSteps; step++) {
    for (int i = 0; i < 8; i++) {
      if (steps[i] > 0) {
        int angle = map(step, 0, maxSteps, pos[i], targetSet[targetSetIndex][i]);
        setServo(getServoChannel(i), angle);
      }
```

```
89.     }
90.     delay(2); // Ajusta el valor del delay para controlar la
   velocidad del movimiento
91.   }
92.
93.   // Actualizar las posiciones actuales y verificar si todos los
   servos están en sus objetivos
94.   for (int i = 0; i < 8; i++) {
95.     pos[i] = targetSet[targetSetIndex][i];
96.     if (pos[i] != targetSet[targetSetIndex][i]) {
97.       allServosAtTarget = false;
98.     }
99.   }
100.
101.       return allServosAtTarget;
102.     }
103.
104.     void moveToPositionSynchronously(int targetAngle) {
105.       int maxSteps = 0;
106.       int steps[8];
107.
108.       // Calcular el número máximo de pasos necesario para
   cualquier servo
109.       for (int i = 0; i < 8; i++) {
110.         steps[i] = abs(pos[i] - targetAngle);
111.         if (steps[i] > maxSteps) {
112.           maxSteps = steps[i];
113.         }
114.       }
115.
116.       // Mover todos los servos sincronizadamente
117.       for (int step = 0; step <= maxSteps; step++) {
118.         for (int i = 0; i < 8; i++) {
119.           if (steps[i] > 0) {
120.             int angle = map(step, 0, maxSteps, pos[i],
   targetAngle);
121.             setServo(getServoChannel(i), angle);
122.           }
123.         }
124.         delay(2); // Ajusta el valor del delay para controlar la
   velocidad del movimiento
125.       }
126.
127.       // Actualizar las posiciones actuales
128.       for (int i = 0; i < 8; i++) {
129.         pos[i] = targetAngle;
130.       }
131.     }
132.
```

```
133.     int getServoChannel(int index) {
134.       switch (index) {
135.         case 0: return 0;
136.         case 1: return 1;
137.         case 2: return 4;
138.         case 3: return 5;
139.         case 4: return 8;
140.         case 5: return 9;
141.         case 6: return 12;
142.         case 7: return 13;
143.         default: return 0;
144.       }
145.     }
146.
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

SIMULACIÓN E IMPLEMENTACIÓN DEL ROBOT CUADRÚPEDO SPOT

2. PLANOS

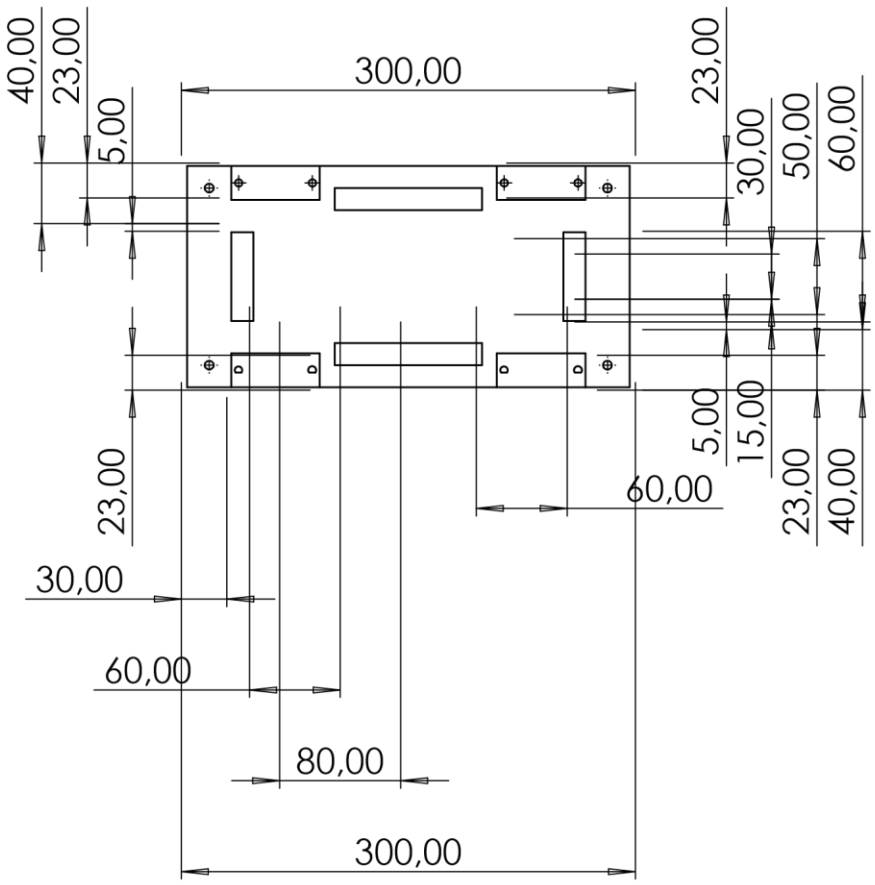
Trabajo Fin de Máster

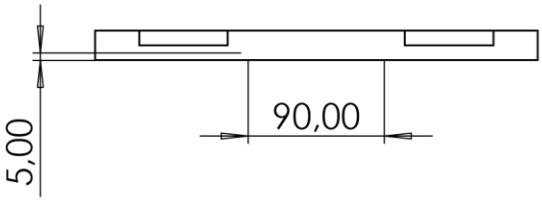
Máster en Ingeniería Mecatrónica

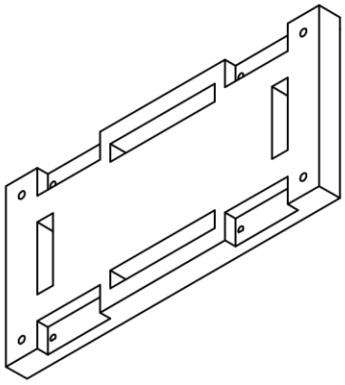
Realizado por: Diego Vicente Camarena Morant

Tutorizado por: Vicente Casanova

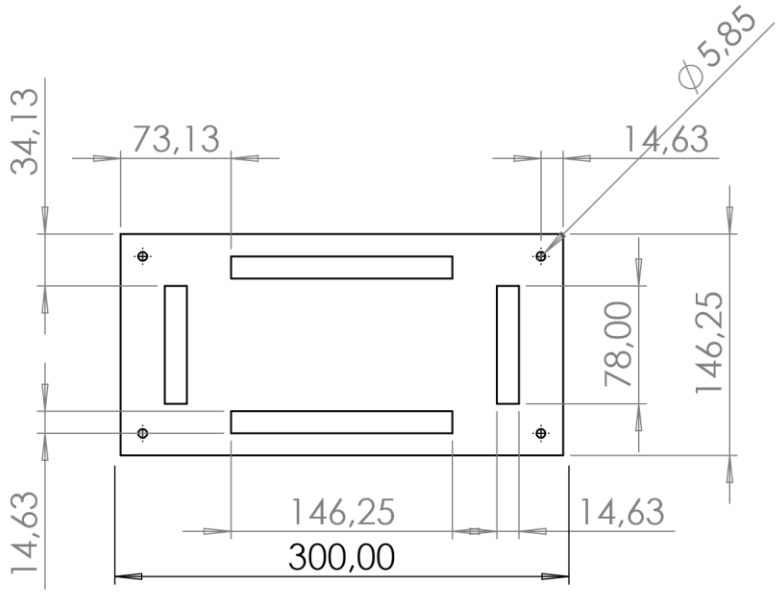
Curso Académico: 2023/2024

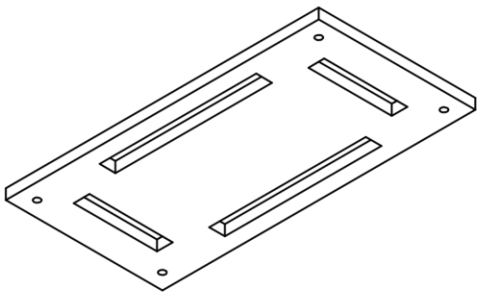




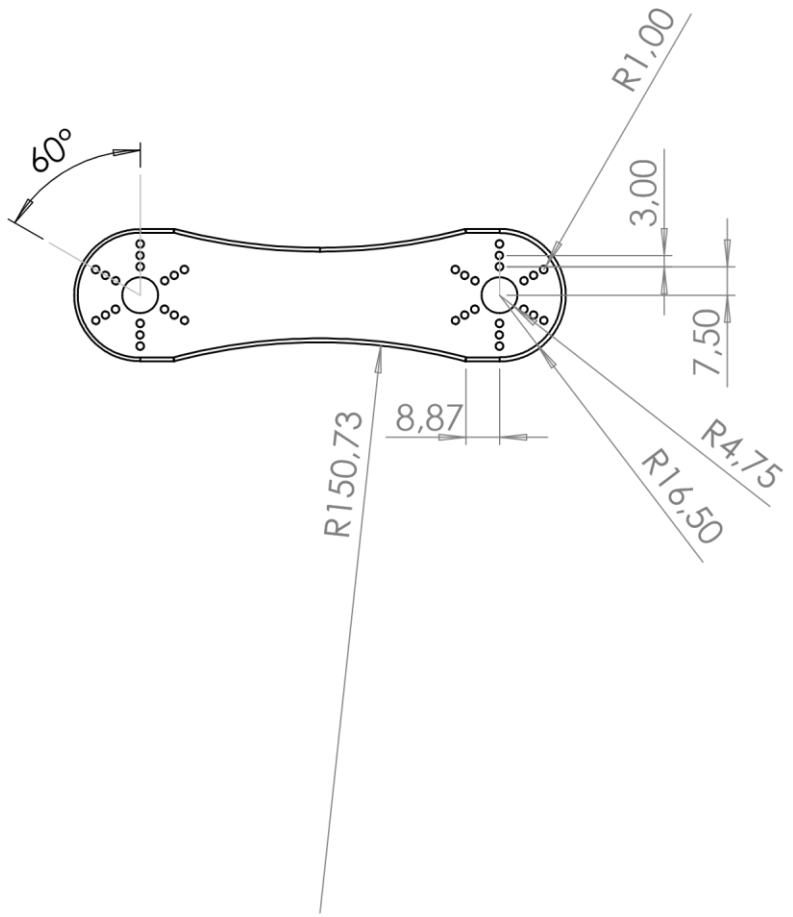


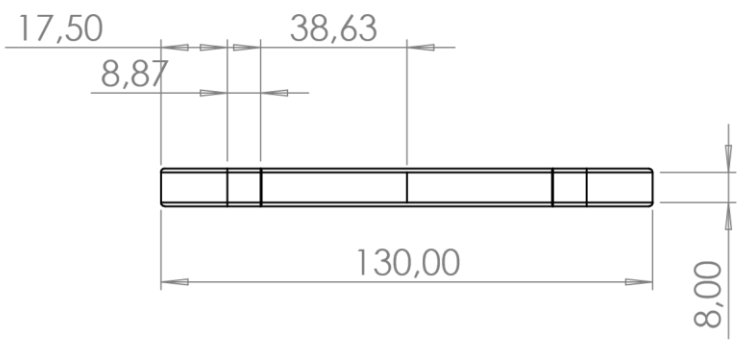
SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MM	REBARBAR Y ROMPER ARISTAS VIVAS	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AEROESPACIAL Y DISEÑO INDUSTRIAL	
	NOMBRE	FECHA	TÍTULO: SIMULACIÓN E IMPLEMENTACIÓN DEL ROBOT CUADRÚPEDO SPOT
PROYECTADO			
DIBUJADO	Diego Vicente Camarena Morant	25 de Junio del 2024	
CONFORMADO			
MATERIAL: Metacrilato	90	Denominación del plano: Plano de la parte inferior del cuerpo	A4
		ESCALA:1:5	PLANO 1

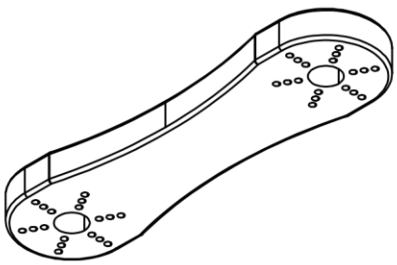




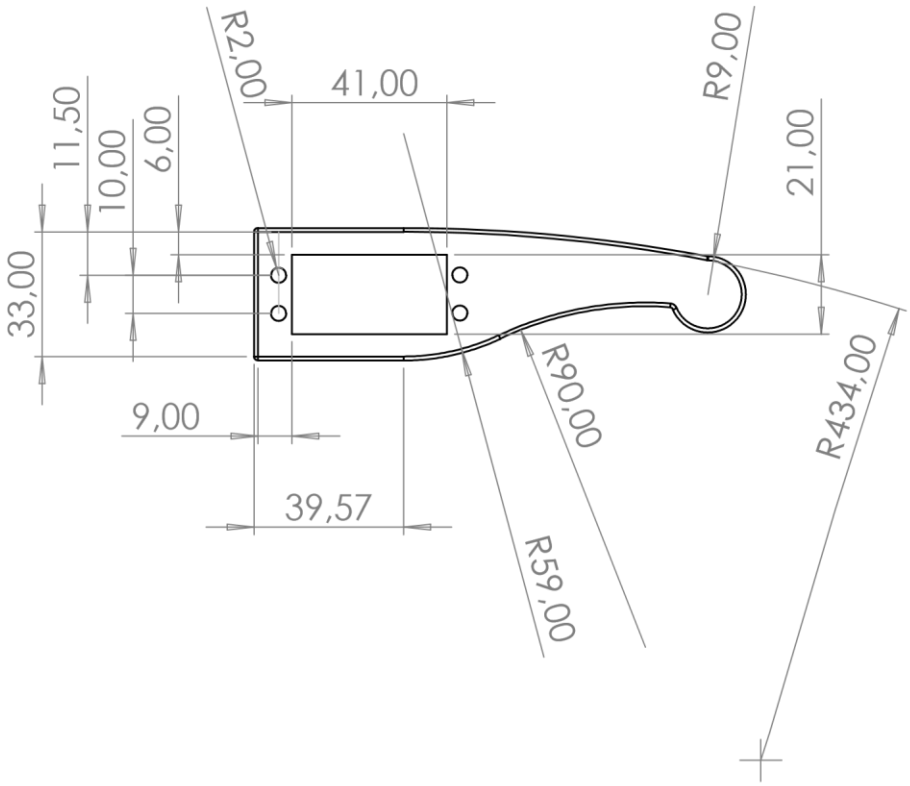
SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MM	REBARBAR Y ROMPER ARISTAS VIVAS	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AEROESPACIAL Y DISEÑO INDUSTRIAL	
	NOMBRE	FECHA	TÍTULO: SIMULACIÓN E IMPLEMENTACIÓN DEL ROBOT CUADRÚPEDO SPOT
PROYECTADO			
DIBUJADO	Diego Vicente Camarena Morant	25 de Junio del 2024	
CONFORMADO			Denominación del plano: Plano de la parte superior del cuerpo — A4
MATERIAL: Metacrilato	91		ESCALA:1:5 — PLANO 2

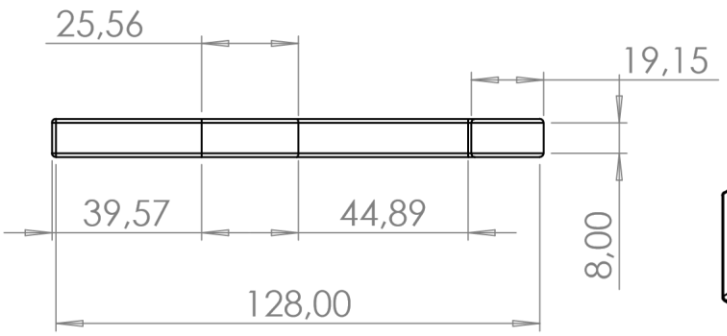


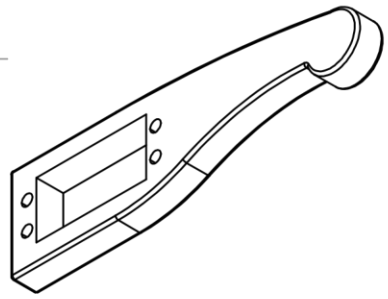




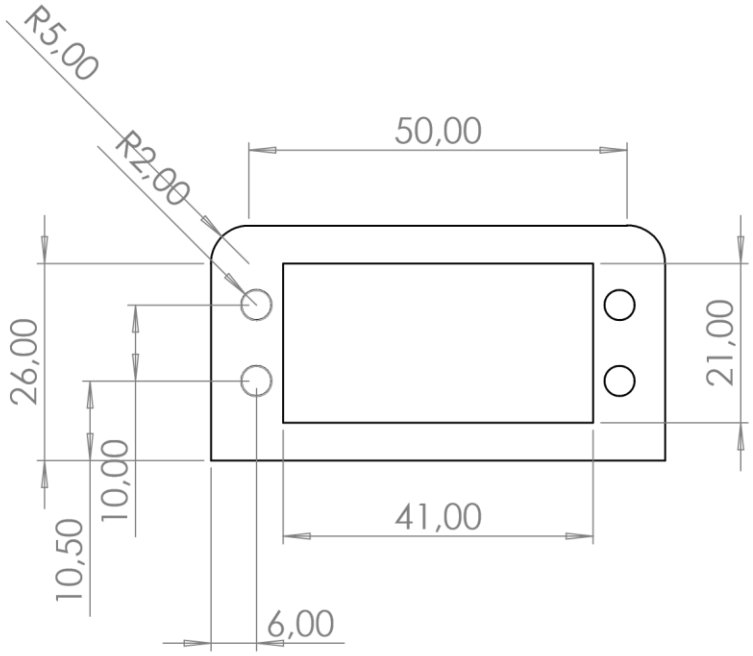
SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MM	REBARBAR Y ROMPER ARISTAS VIVAS	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AEROESPACIAL Y DISEÑO INDUSTRIAL	
	NOMBRE	FECHA	TÍTULO: SIMULACIÓN E IMPLEMENTACIÓN DEL ROBOT CUADRÚPEDO SPOT
PROYECTADO			
DIBUJADO	Diego Vicente Camarena Morant	25 de Junio del 2024	
CONFORMADO			
MATERIAL: Metacrilato	92		Denominación del plano: Plano de la parte superior de cada pata — A4
			ESCALA:1:2 — PLANO 3

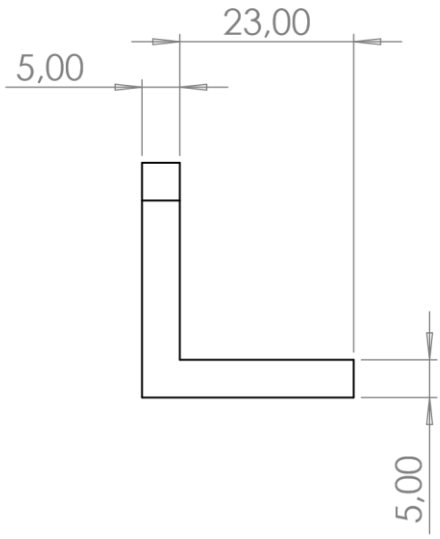


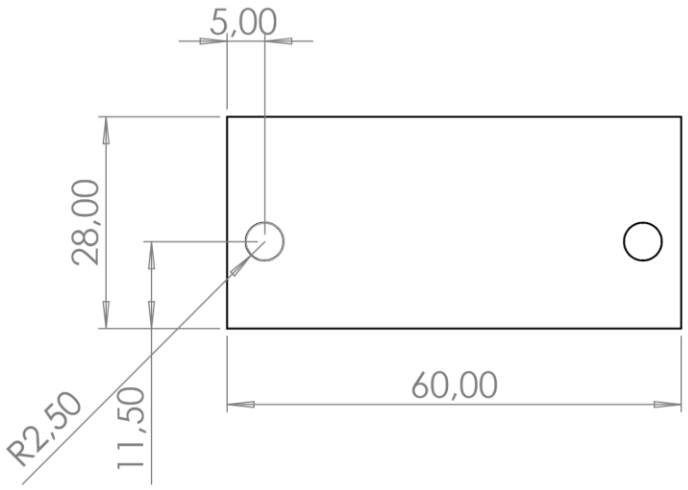


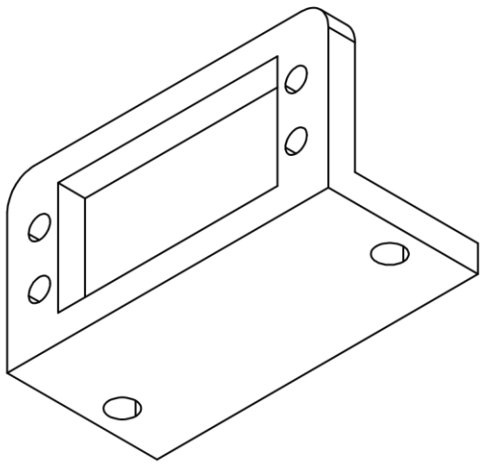


SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MM	REBARBAR Y ROMPER ARISTAS VIVAS	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AEROESPACIAL Y DISEÑO INDUSTRIAL	
	NOMBRE	FECHA	TÍTULO: SIMULACIÓN E IMPLEMENTACIÓN DEL ROBOT CUADRÚPEDO SPOT
PROYECTADO			
DIBUJADO	Diego Vicente Camarena Morant	25 de Junio del 2024	
CONFORMADO			
MATERIAL: Metacrilato	93		Denominación del plano: Plano de la parte inferior de cada pata — A4
			ESCALA:1:2 — PLANO 4









SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MM	REBARBAR Y ROMPER ARISTAS VIVAS	ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AEROESPACIAL Y DISEÑO INDUSTRIAL
	NOMBRE	FECHA
PROYECTADO		
DIBUJADO	Diego Vicente Camarena Morant	25 de Junio del 2024
CONFORMADO		
MATERIAL: Nylon	94	

TÍTULO:

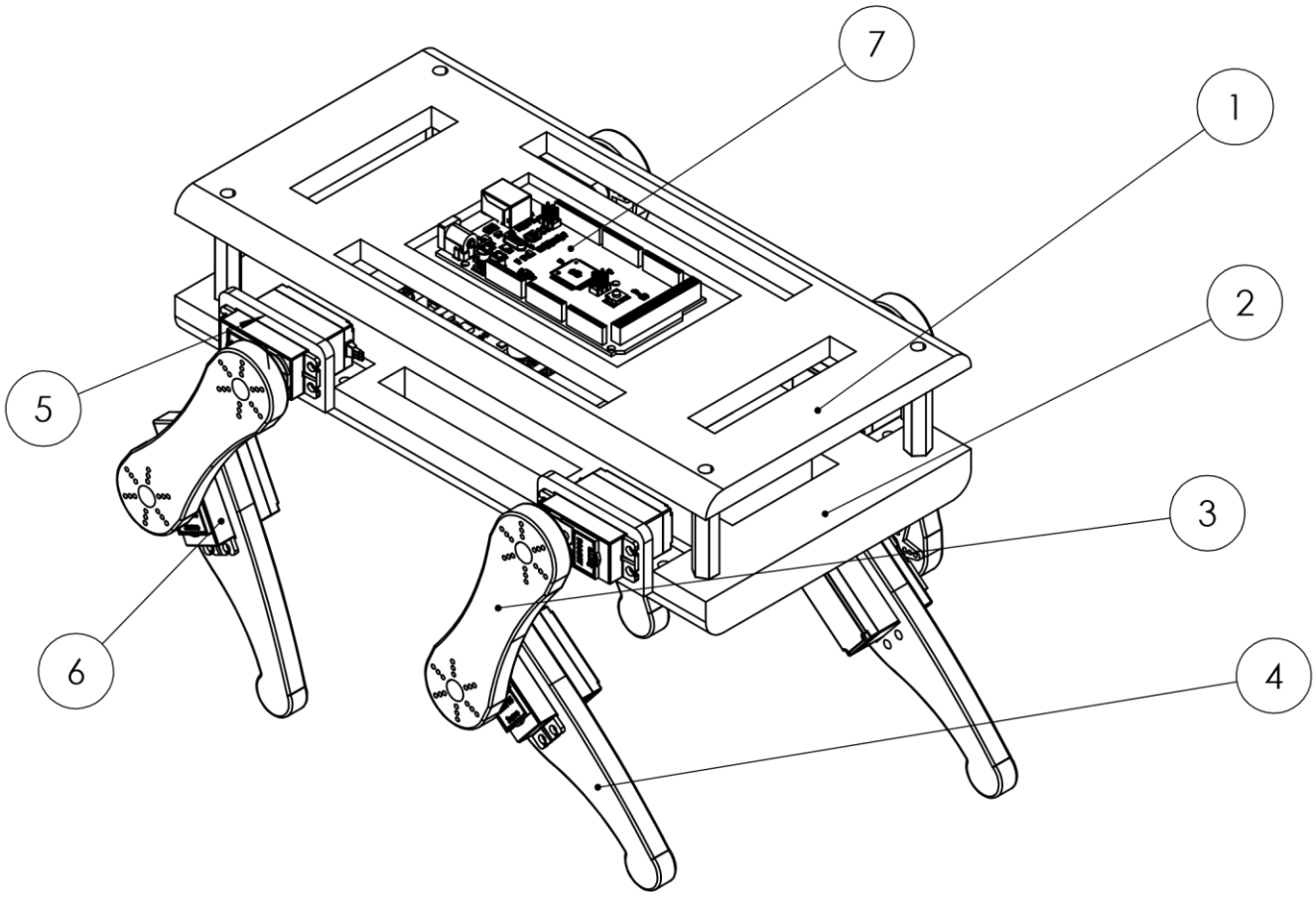
SIMULACIÓN E IMPLEMENTACIÓN DEL ROBOT CUADRÚPEDO SPOT

Denominación del plano: Plano de la base de los servos

A4

ESCALA:1:1

PLANO 5



N.º DE ELEMENTO	N.º DE PIEZA	DESCRIPCIÓN	CANTIDAD
1	Cuerpo inferior		1
2	Pata superior		4
1	Cuerpo superior		1
4	Pata inferior		4
5	Base servos		4
6	Servomotor		8
7	Placa Arduino		1

SI NO SE INDICA LO CONTRARIO LAS COTAS SE EXPRESAN EN MM

REBARBAR Y ROMPER ARISTAS VIVAS

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AEROESPACIAL Y DISEÑO INDUSTRIAL

	NOMBRE	FECHA
PROYECTADO		
DIBUJADO	Diego Vicente Camarena Morant	25 de Junio del 2024
CONFORMADO		

95

TÍTULO:

SIMULACIÓN E IMPLEMENTACIÓN DEL ROBOT CUADRÚPEDO SPOT

Denominación del plano: Despiece del ensamblaje del robot

A4

ESCALA:1:5

PLANO 6

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

SIMULACIÓN E IMPLEMENTACIÓN DEL ROBOT CUADRÚPEDO SPOT

3. PLIEGO DE CONDICIONES

Trabajo Fin de Máster

Máster en Ingeniería Mecatrónica

Realizado por: Diego Vicente Camarena Morant

Tutorizado por: Vicente Casanova

Curso Académico: 2023/2024

Índice del pliego de condiciones

1. OBJETO

El presente documento tiene como objeto recoger las especificaciones técnicas y legales para llevar a cabo la ejecución del presente proyecto referido al robot cuadrúpedo tipo Spot.

Se explicará a detalle los materiales utilizados, así como los procesos de montaje y control de calidad que se han de seguir para el correcto funcionamiento del robot. En caso de que se realice cualquier modificación, no se garantiza que su funcionamiento sea el correcto.

2. CONDICIONES Y NORMATIVA

En este apartado se muestran las normativas que deberán de seguirse en todo momento durante la ejecución del proyecto:

- **ISO 8373:2021**. Define los términos y conceptos fundamentales relacionados con la robótica y los dispositivos robóticos.
- **UNE-EN ISO 10218:2012**. Referente a los requisitos de seguridad para robots industriales. Parte 1: Robots.
- **UNE-EN ISO 10218-2:2011**. Requisitos de seguridad para robots industriales. Parte 2: Sistemas robot e integración.
- **ISO/IEC 19510:2013**. Estándar para el modelado de procesos que incluyen simulación de sistemas robóticos.
- **UNE-EN ISO 13482:2014**. Robots y dispositivos robóticos. Requisitos de seguridad para robots no industriales.
- **ISO 9283:1998**. Define los criterios de rendimiento y métodos de prueba para robots y dispositivos robóticos.
- **ISO 9409-1:2004.** Trata sobre las interfaces mecánicas en robots industriales.
- **UNE-EN ISO 14644-18:2024.** Salas limpias y locales anexos controlados.
- **UNE-EN ISO 9001:2015**. Sistemas de gestión de calidad.
- **ISO 12100:2010**. Seguridad de maquinaria – Principios generales para el diseño – Evaluación y reducción del riesgo.
- **ISO/TR 20218-1:2018**. Robótica: Diseño y seguridad para sistemas de robots industriales – Parte 1: Dispositivos finales.
- **ISO 9241-210:2019**. Ergonomía de la interacción hombre – Sistema – Parte 210: Diseño centrado en el operados humano para los sistemas interactivos.
- **Real Decreto 208/2005**. Medidas para prevenir la generación de residuos procedentes de aparatos eléctricos y electrónicos.
- **Ley 12/1986, de 1 de abril**. Regula las atribuciones profesionales de los Arquitectos e Ingenieros técnicos.
- **ISO 19649**. Define términos relacionados con robots móviles que viajan sobre una superficie sólida y que operan tanto en aplicaciones de robots industriales como de robots de servicio.

- **ISO DIN 13.** Tamaños y métricas normalizadas para tornillos y tuercas.
- **ISO 898-1**. Requisitos mecánicos para varillas roscadas de acero.
- **DIN 125**. Dimensiones y características de las arandelas planas.

3. CONDICIONES DE LOS MATERIALES

3.1. Piezas de las patas y el cuerpo

Estas piezas se fabricarán mediante corte láser con la ayuda de una impresora 3D en metacrilato.

En cuanto al control de calidad se tendrá en cuenta que las placas no presenten fisuras o imperfecciones.

El metacrilato estará estandarizado según la norma ISO 7823-1:2003, que especifica los tipos, dimensiones y características de las láminas de metacrilato (PMMA).

3.2. Piezas de las bases de los servomotores

Estas piezas, debido a su complejidad para ser impresas en metacrilato, serán de nylon.

En cuanto al control de calidad se tendrá en cuenta que las piezas no presenten grietas ni imperfecciones.

El nylon estará estandarizado según la norma ISO 1031-1, que especifica el sistema de designación para los materiales termoplásticos, incluyendo el nylon, basado en su composición y características de uso.

3.3. Tuercas, tornillos, varilla roscada y arandelas

Para el montaje del robot se han utilizado diversos elementos de fijación como arandelas, tuercas, tornillos y una varilla roscada. A continuación, se detallarán las especificaciones y control de calidad de cada uno de los componentes.

3.3.1. Tuercas

Las tuercas utilizadas están estandarizadas según la normativa ISO DIN-13 y serán del tipo M3 y M5 de acero inoxidable para garantizar su resistencia y durabilidad. Serán tuercas autoblocantes para tener una mayor fijación y evitar aflojamientos por vibraciones.

Su norma de estandarización son la ISO 4032 referida a las dimensiones y características de las tuercas hexagonales con rosca métrica.

3.3.2. Arandelas

Las arandelas empleadas serían de acero inoxidable y serían del tamaño M3 y M5. Se colocarán entre las tuercas y la superficie de la pieza para distribuir la carga de apriete y evitar daños en el material. También se colocarán entre el tornillo y el eje del motor para garantizar una mayor fijación de este a las patas.

Su norma de estandarización son la DIN 125, referida a las dimensiones y características de las arandelas planas.

3.3.3. Tornillos

Se utilizarán tornillos M3 X 20 mm y tornillos pasantes M5 X 20 mm. Los primeros se utilizarán para fijar el eje del servomotor en la articulación con la pata, y los segundos para sujetas los servomotores en sus bases.

La normativa DIN 125 se encargará de establecer sus dimensiones y características estandarizadas.

3.3.4. Varilla roscada

Para la unión de las dos partes del cuerpo del robot se utilizará una varilla roscad M5, la cual se dividirá en 4 trozos de 100 mm cada uno, fabricada de acero inoxidable.

Su norma de estandarización es la ISO 898-1, especificando los requisitos mecánicos para varillas roscadas de acero.

3.3.5. Control de calidad

Para garantizar su integridad y funcionalidad se realizarán inspecciones visuales y mecánicas, verificando que no presenten deformaciones, roturas, corrosión u otros defectos. También se comprobará la fijación de los elementos, así como que se cumplan las dimensiones con respecto a las especificaciones de las normas aplicadas.

Estos elementos se encargarán de fijar los elementos del robot, garantizando una estructura robusta y confiable, permitiendo un montaje preciso y un mantenimiento eficiente.

4. CONDICIONES DE EJECUCIÓN

En este apartado se detallará como se ha realizado el montaje del robot cuadrúpedo para garantizar su correcto ensamblaje y funcionamiento. Es importante haber conectado el Arduino DUE con el diver PCA9685 según se ha explicado en el apartado 4.2 del documento de la memoria del presente proyecto. También es importante que, previo al montaje e implementación del robot, se haya probado la secuencia de movimiento de los motores y comprobado que es correcta.

4.1. Descripción del montaje

Se empezará estableciendo todos los servomotores en su posición 0 para que, cuando se monte la estructura del robot, están todas las articulaciones en la posición inicial. A continuación, se colocarán las piezas impresas en nylon en su respectivo hueco en la parte inferior del cuerpo, lijándolas previamente para que puedan introducirse con mayor facilidad. Se fijarán con cola potente, a poder ser Loctite – Super Glue -3, pues estas bases sujetarán el peso de toda la estructura y tendrán que estar todo lo fijadas que se pueda.

El siguiente paso es montar juntar la otra parte del cuerpo, de forma que se deje una separación entre capas de unos 4 cm, dejando unos pocos centímetros fuera en cada lado para poder fijarlas con tuercas. Se colocarán tuercas M5 tanto arriba como debajo de cada parte del cuerpo con sus respectivas arandelas M5, quedando la parte del cuerpo muy robusta.

Con la estructura del cuerpo completada, se procederá a colocar los 4 motores que unirán el cuerpo con las patas. Para ello, se colocarán en las bases de nylon y se fijarán con tornillos pasantes M5 y tuercas autobloqueantes M5. A continuación, se colocarán los horns en forma de estrella en el eje del motor y se alinearán con el extremo de la pieza superior de las patas en la posición inicial de estas (aproximadamente 110º con respecto a la horizontal). Para una correcta fijación, se colocarán tres clavos 1.5 mm x 20 mm en las filas de agujeros de la estrella, pasando por el horn de cada eje del motor, para mantenerlo alineado con la pata. También se colocarán clavos 1 mm x 20 mm, doblando sus puntas para evitar que se salgan. Finalmente, se hará pasar un tornillo M3 x 20 mm por el eje del motor junto a una arandela para que, al apretarlo, quede la pieza completamente fijada con el servomotor. De esta forma se consigue que la pieza no deslice con respecto al movimiento del eje.

Ahora queda unir la parte de bajo de las patas. Para ello, se colocan cada uno de los servomotores en las articulaciones y las patas a un ángulo de unos 20º con respecto a la horizontal. La forma de unión es la misma explicada anteriormente, con los clavos, tornillos y arandelas. También será necesario unir el motor a su base con tuercas y tornillos M5.

4.2. Control de calidad

Para garantizar el correcto funcionamiento del robot se realizará un control de calidad tras su montaje, consistiendo en comprobar su robustez. Para ello se le aplicará un programa que consista en mantener los servomotores en su posición inicial de forma permanente y comprobar que el robot es capaz de mantenerse en pie y de soportar ligeros golpes, garantizando de esta forma su estabilidad.

Importante comentar que, debido a que la alimentación de los motores no era lo suficientemente potente, se realizaron cambios en la estructura del robot y su forma de alimentarlo. Puede leerse con mayor detalle en el apartado 4.5 del documento de la memoria del presente proyecto.

5. PRUEBA DE SERVICIO

Para la prueba en servicio se cargarán los códigos desarrollados en Arduino en la placa Arduino Due. Se dispone de tres códigos, siendo cada uno un tipo de movimiento diferente: sentarse, crawl y trot.

Tras volver a revisar las conexiones del circuito, se comprobará que el robot es capaz de realizar el movimiento de sentarse, siendo el más sencillo. Como se ha comentado, al principio los motores no tenían fuerza para poder realizarlo, pero tras modificar el peso de la estructura y, principalmente, la forma de alimentar los motores ya era capaz. Posteriormente se comprobaron los otros programas, modificando, en caso de que sea necesario, el valor de los ángulos de la secuencia que sigue el robot para que vaya de la forma más fluida posible.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

SIMULACIÓN E IMPLEMENTACIÓN DEL ROBOT CUADRÚPEDO SPOT

4. PRESUPUESTO

Trabajo Fin de Máster

Máster en Ingeniería Mecatrónica

Realizado por: Diego Vicente Camarena Morant

Tutorizado por: Vicente Casanova

Curso Académico: 2023/2024

Índice del presupuesto

En este documento se procederá a calcular el presupuesto total del proyecto.

1. PRECIO DE LOS MATERIALES

El precio de los materiales empleados para la fabricación del proyecto se divide en:

- Materiales que se han comprado para el montaje del robot
- Piezas impresas en 3D

1.1. Materiales comprados

Descripción del material	Coste unitario (€/ud)	Cantidad (ud)	Coste total (€)
Placa Arduino Due	42	1	42
Servomotor MG996R	6,5	8	52
Controlador PCA9685	3,38	1	3,38
Varilla roscada M5 x 100 mm	0,2	4	0,8
Tuerca M5	0,05	50	2,5
Arandela M5	0,03	20	0,6
Arandela M3	0,03	12	0,36
Clavo 1.5 mm x 20 mm	0,05	30	1,5
Clavo 1 mm x 20 mm	0,005	200	1
Tornillo pasante M5 x 20 mm	0,1	40	4
Lija	2	1	2
Cola de secado lento	5,5	2	11
Cola extrafuerte de secado rápido	10,95	1	10,95
Sierra	11,50	1	11,5
Pila AA 1,5V	0,2	4	0,8
Cinta adhesiva	1	1	1
Porta pilas para 4 pilas	2,5	1	2,5
		TOTAL	147,89

Tabla 3: Coste de los materiales comprados.

1.2. Coste de fabricación de las piezas

Para la fabricación de las piezas se ha empleado una cortadora láser para las piezas del cuerpo y las patas en metacrilato y una impresora 3D para las bases de los servomotores en nylon. Se ha estimado el precio de cada servicio por hora y la cantidad de horas que han sido necesarias para la impresión de las piezas incluyendo los cortes que tuvieron que hacerse posteriormente para reducir el peso del robot. Para el corte láser se ha estimado un coste de 4.2 euros por hora y para la impresión 3D en nylon 5 euros por hora.

Descripción de la pieza	Coste unitario (€/h)	Cantidad (h)	Coste total (€)
Cuerpo del robot	2,5	0,7	1,75
Patas del robot	2,5	0,3	0,75
Base de los servomotores	5	10	50
		TOTAL	52,5

Tabla 4: Costes de la fabricación de las piezas.

1.3. Coste total de los materiales

Tipo de coste	Coste (€)
Materiales comprados	147,89
Fabricación de las piezas	52,5
TOTAL	200,39

Tabla 5: Coste total de los materiales.

2. MANO DE OBRA

Para calcular el coste de la mano de obra se ha tenido en cuenta el tiempo aproximado que se ha invertido en las diferentes fases del proyecto, estimando un salario de 30 euros por hora.

Descripción del trabajo	Coste unitario (€/h)	Cantidad (h)	Coste total (€)
Simulaciones	30	90	2700
Diseño CAD	30	10	300
Montaje	30	30	900
Pruebas	30	30	900
Redacción	30	50	1500
		TOTAL	6300

Tabla 6: Coste de la mano de obra.

3. LICENCIAS Y EQUIPO

Para calcular el coste del uso del equipo y las licencias que se han empleado para la realización del trabajo se tendrá en cuenta su coste unitario, tiempo de vida útil y el tiempo que se ha utilizado durante el proyecto, obteniendo el coste repercutido en este.

Descripción del recurso	Coste unitario (€)	Vida útil (años)	Tiempo utilizado (años)	Coste repercutido (€)
Licencia de Matlab/Simscape	900	1	0.5	450
Licencia de SolidWorks	3495	1	0,5	1747,5
Licencia de Windows 10	12,5	5	0,5	1,25
Licencia de Microsoft Office	817	1	0,5	408,5
Ordenador portátil	300	5	0,5	30
Ordenador de sobre mesa	500	5	0,5	50
			TOTAL	2267,25

Tabla 7: Coste de las licencias y equipos utilizados.

4. VALORACIÓN

En este apartado se procederá a calcular el presupuesto total del proyecto, empezando con su presupuesto de ejecución:

Tipo de coste	Coste (€)	%
Materiales	200,39	2,28
Mano de obra	6300	71,86
Licencia y equipos	2267,25	25,86
PRESUPUESTO DE EJECUCION	8767,64	100

Tabla 8: Coste del presupuesto de ejecución del proyecto.

Una vez obtenido el presupuesto de ejecución hay que aplicar los gastos generales asociados con la operación y administración general de la empresa durante la ejecución del proyecto y el beneficio industrial, referente al porcentaje adicional agregado al coste total del proyecto que representa el beneficio neto que la empresa espera obtener después de cubrir los gastos directos e indirectos. Por supuesto, también se le aplicará el 21% de IVA.

Gastos	Coste (€)
12% Gastos generales	1052,12
6% de Beneficio Industrial	526,06
PRESUPUESTO	10345,82
21% de IVA	2172,62
PRESUPUESTO + IVA	12518,44

Tabla 9: Presupuesto final del proyecto.

El presupuesto final asciende a un total de doce mil quinientos dieciocho euros con cuarenta y cuatro céntimos.

Valencia, 28 de junio de 2024.