Escuela
Politécnica
Superior

Impacto gravitatorio en la planificación y seguimiento de trayectorias en un robot cuadrúpedo

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:
Sergio Arias Hernández
Tutor/es:
Jorge Pomares Baeza
Carlos Alberto Jara Bravo

Julio 2023

Universitat d'Alacant
Universidad de Alicante

Impacto gravitatorio en la planificación y seguimiento de trayectorias en un robot cuadrúpedo

Autor
Sergio Arias Hernández

Tutor/es
Jorge Pomares Baeza
Departamento de física, ingeniería de sistemas y teoría de la señal
Carlos Alberto Jara Bravo
Departamento de física, ingeniería de sistemas y teoría de la señal

Grado en Ingeniería Robótica

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2023

Preámbulo

Desde mi infancia, la robótica y el espacio han sido fuentes constantes de fascinación para mí. Este trabajo surge de mi deseo por combinar ambos intereses y explorar las posibilidades de la robótica en el espacio.

Durante el desarrollo de este proyecto se realiza un estudio de *towr*, un *framework* de optimización de trayectorias. También, se lleva a cabo un estudio de la cinemática y dinámica del robot Go1 de *Unitree Robotics*. Mediante *towr* se generan trayectorias para este robot con tres gravedades distintas: terrestre, marciana y una intermedia. Posteriormente, se realiza un seguimiento de las trayectorias utilizando *Gazebo*. Y finalmente, se analizan las diferencias en las trayectorias y en las respuestas de los robots ante las variaciones gravitacionales.

Además, en el caso de la gravedad terrestre, se realizan pruebas con el robot real.

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor, Jorge Pomares Baeza, quien ha sido capaz de soportar y contestar a las cientos de preguntas y dudas que he me han ido surgiendo, y cuya guía ha sido de gran utilidad en cada etapa del proyecto. Agradecer también a Carlos Alberto Jara Bravo y a todo el grupo de investigación de Human Robotics, con quienes cursé la asignatura de Prácticas Externas y me ofrecieron un ambiente de trabajo increíble y cómodo donde puede realizar un proyecto sobre otro campo de la robótica: la robótica de rehabilitación.

Gracias también a Jonathan Mortes, propietario del robot Go1, quien ha podido hacer tiempo para ayudarme y realizar las pruebas en el robot real, pese a los posibles peligros que esto podía conllevar.

Agradezco de todo corazón a mis amigos del grado, quienes han sido unos compañeros increíbles en esta etapa de mi vida. Juntos, hemos compartido momentos maravillosos y hemos aprendido y crecido, tanto en el ámbito académico como en lo personal.

No puedo olvidar mencionar a mis amigos de toda la vida, quienes han estado a mi lado desde siempre, apoyándome y compartiendo grandes momentos juntos. A pesar de las distancias y las idas y venidas, hemos seguido creando recuerdos inolvidables. Han sido y serán, un pilar fundamental en mi vida, y estoy infinitamente agradecido por su presencia.

Este trabajo no habría sido posible sin la educación y los valores inculcados por mis padres. Su dedicación, apoyo incondicional y ejemplo a lo largo de mi vida han sido fundamentales en mi desarrollo académico y personal. Agradezco profundamente su esfuerzo y sacrificio para brindarme las oportunidades necesarias para seguir mis pasiones y perseguir mis metas. Su amor, sabiduría y guía han sido un faro en mi camino, inspirándome a superar obstáculos y a nunca dejar de aprender. Les estoy infinitamente agradecido por su constante apoyo y por ser mis mayores impulsores en todos los aspectos de mi vida. Os quiero. Y finalmente agradecer a mi hermana, quien, pese a todas las discusiones, ha sido una fuente invaluable de apoyo y distracción en los momentos más difíciles.

A mi familia y amigos,
sin los cuales habría sido todo mucho más difícil.

Cualquier tecnología
lo suficientemente avanzada
es indistinguible de la magia

Tercera ley de Clarke; Arthur C. Clarke.

Índice general

Índice de figuras

Índice de Códigos

1 Introducción

La exploración espacial ha sido una de las mayores hazañas de la humanidad. Desde los primeros pasos del ser humano en la Luna hasta las misiones espaciales en otros planetas, el interés por conocer y comprender el universo ha sido una cosa continua en la historia humana. Y en esta búsqueda de conocimiento, la robótica actualmente desempeña y desempeñará un papel fundamental.

La robótica espacial se ha convertido en una disciplina esencial para la exploración y el estudio de otros planetas, satélites o asteroides. Los robots espaciales han demostrado su valía al ser capaces de adentrarse en diferentes entornos y realizar tareas complejas en lugares donde los seres humanos no pueden llegar. Desde la recopilación de muestras en Marte hasta la reparación de satélites en órbita, los robots espaciales han ampliado nuestras capacidades y han sido nuestros ojos y manos en el espacio.

Esta área de la robótica plantea desafíos únicos, especialmente debido a los entornos con diferente gravedad o microgravedad. Estos desafíos son especialmente relevantes en el caso de los robots con patas, ya que deben adaptarse y ajustar su locomoción para mantener el equilibrio y realizar movimientos precisos en condiciones de gravedad alterada.

Esta necesidad de los robots con patas de moverse de manera efectiva y estable en otras condiciones de gravedad, es crucial para su éxito en las misiones espaciales. La adaptación de su locomoción no solo garantiza su propia supervivencia, sino que también les permite realizar tareas de exploración, recolección de datos y construcción.

En este contexto, el presente proyecto tiene como objetivo adentrarse en el mundo de la robótica espacial. A través del estudio y desarrollo de trayectorias para un robot en diferentes gravedades, se pretende explorar cómo afecta la variación gravitacional en el movimiento y las respuestas del robot. Esto permitirá comprender mejor los desafíos y las oportunidades que surgen al enfrentarse a entornos con una gravedad distinta.

1.1 Motivación

La curiosidad ha sido siempre mi principal motor para explorar cosas nuevas. Por eso, cuando mi tutor Jorge planteó la idea de este proyecto no dudé en aceptarlo directamente.

La propuesta de trabajar con un robot cuadrúpedo, un robot diferente de los que había utilizado antes, y con un tema relacionado al espacio despertó mi entusiasmo y mi sed de conocimiento. Desde pequeño, he sentido una profunda fascinación por el espacio y siempre he soñado con formar parte de los avances en la exploración espacial. La idea de poder aportar

mi granito de arena y poder ayudar en un futuro en el desarrollo de nuevas tecnologías en este campo es algo que me motiva y emociona.

1.2 Estructura

La memoria presenta una estructura clara y cómoda para que cualquier persona interesada en el tema pueda leerla sin complicación.

En el Capítulo 2, se establecen los objetivos específicos que se deben alcanzar para lograr el objetivo general de este proyecto de fin de grado. En el Capítulo 3, se aborda el tema de los robots cuadrúpedos, su potencial uso en la robótica espacial y las diferencias clave con respecto a los robots móviles. Además, se exploran diferentes métodos de optimización y optimización de trayectorias.

El Capítulo 4 se centra en la descripción detallada de las herramientas y técnicas utilizadas en el desarrollo de este proyecto. Se presentan los aspectos metodológicos que guían la implementación práctica, incluyendo el uso de los *frameworks* y software específicos.

En el Capítulo 5, se lleva a cabo la implementación práctica paso por paso del proyecto, utilizando las herramientas y metodologías descritas anteriormente.

Finalmente, en el Capítulo 6 se muestran los resultados obtenidos y se realizan las justificaciones, mientras que en el último capítulo, el Capítulo 7, se realiza una conclusión final que resume lo realizado, los hallazgos y resultados.

2 Objetivos

El objetivo general de este Trabajo Final de Grado (TFG) es investigar y analizar el impacto de diferentes gravedades en la planificación y seguimiento de trayectorias en un robot cuadrúpedo, concretamente en robot Go1 de *Unitree Robotics*. Para ello, se introduce el Go1 en el *framework* de optimización de trayectorias llamado *towr* en el entorno de Robot Operating System (ROS). Mediante esta herramienta, se planifican trayectorias en tres distintas condiciones de gravedad: terrestre, marciana e intermedia. Una vez generadas, mediante los paquetes proporcionados por *Unitree Robotics* y los *rosbag* con las trayectorias generadas por *towr*, se realiza el seguimiento de dichas trayectorias en *Gazebo*. Los datos obtenidos tanto en la planificación como en el seguimiento se recogen con *Plotjuggler*.

Para lograr el objetivo general se establecen los siguientes objetivos específicos:

- Estudio del funcionamiento de *towr*.
- Estudio de la cinemática directa e inversa del robot Go1, así como del modelo dinámico utilizado.
- Estudio de los posibles motores de físicas a utilizar en *Gazebo*.
- Estudio de los paquetes de *Unitree Robotics* para realizar el seguimiento.
- Incluir el robot Go1 y su cinemática inversa en *towr*.
- Generar la trayectoria para las tres gravedades y obtener los datos.
- Realizar el seguimiento de cada trayectoria y obtener los datos.
- Analizar el impacto de las gravedades en la planificación y el seguimiento.
- Pruebas en gravedad terrestre con el robot real.

3 Marco Teórico

3.1 Robótica espacial

La robótica espacial es un ámbito especializado de la robótica que se centra en el uso de robots para asistir o reemplazar a los seres humanos en diversas actividades espaciales. Estos robots desempeñan un papel crucial en experimentos científicos y otras tareas realizadas en el espacio, lo que genera un impacto cada vez mayor en los métodos tradicionales de transporte espacial, construcción en órbita, mantenimiento en el espacio y exploración planetaria. En este sentido, los robots espaciales se establecen como herramientas clave que permitirán realizar futuras misiones espaciales, ya sean tripuladas o no tripuladas.

El desarrollo de robots espaciales implica enfrentar desafíos significativos en cuanto a diseño, fabricación y control, debido a que operan en un entorno espacial considerablemente diferente de las condiciones terrestres. Para abordar estos desafíos, se requiere una estrecha colaboración entre expertos de distintas disciplinas. Estos profesionales tienen el objetivo común de avanzar en el desarrollo y la implementación de robots espaciales cada vez más eficientes y sofisticados. (Jiang y cols., 2022)

3.1.1 Propiedades y características esenciales de los robots espaciales

En el libro "Space Robotics" escrito por Yoshida y cols. (2014) se establece que la autonomía es un aspecto fundamental en la robótica, y que realmente, cualquier nave espacial no tripulada bajo control secuencial automatizado, podría ser considerada como un satélite robótico. También indica que, el uso del término "robot espacial", implica un sistema mecánico de mayor capacidad que facilite las diferentes tareas o amplíe las áreas y capacidades de exploración en otros planetas como sustituto de los humanos. Esto implica un nivel más avanzado de inteligencia y habilidades en el sistema robótico, permitiendo una mayor independencia en la toma de decisiones y la ejecución de tareas complejas en entornos extraterrestres.

Por ello se establecen las siguientes propiedades como las características clave de los robots espaciales, así como sus principales problemáticas:

- **Manipulación**: A pesar de que la manipulación es una tecnología fundamental en la robótica, la falta de gravedad en el entorno orbital exige una consideración especial de la dinámica del movimiento de los brazos manipuladores y los objetos que se manejan. Este asunto debe tener en cuenta: la dinámica de las reacciones que afectan al cuerpo principal, la dinámica de impacto cuando la mano robótica entra en contacto con un objeto a ser manipulado y la dinámica de vibración generada por la flexibilidad estructural.
- **Movilidad**: La movilidad es especialmente crucial en los robots exploradores (*rovers*) que se desplazan por la superficie de una luna o planeta. Estas superficies son complejas

y muy parecidas, lo que representa un desafío considerable durante su reconocimiento y movimiento. La percepción y el reconocimiento del entorno, la mecánica y la dinámica del vehículo, el control y la navegación son todas tecnologías de la robótica móvil que deben realizarse en un entorno totalmente inexplorado. Sin embargo, estas características también deberían ser aplicable a robots con patas que deban moverse por el mismo entorno que los *rovers*.

- **Teleoperación y Autonomía**: Debido a la existencia un retraso temporal no despreciable entre los robots en el espacio y un operador humano en la Tierra, el cual puede llegar a ser de varios minutos, e incluso horas, en misiones planetarias la tecnología de tele-robótica es indispensable en la exploración espacial, y la introducción de la autonomía es una opción razonable.
- **Entornos extremos**: Existen muchos problemas relacionados con los entornos extremos del espacio y que deben resolverse para permitir aplicaciones prácticas. Estos problemas incluyen temperaturas extremadamente altas o bajas, alto vacío o alta presión, atmósferas corrosivas….
- **Versatilidad**: Esta característica establece el objetivo final de los robots espaciales. Indica que un robot espacial debe ser adaptable a los entornos extremos del espacio mencionados anteriormente y poseer la versatilidad para manejar muchas situaciones y escenarios diferentes con los recursos que posee, incluidos aquellos que surjan de manera inesperada.

3.1.2 Prototipos de robots espaciales con patas

En general, los robots con patas se pueden dividir en dos categorías: robot bípedos y robots con múltiples patas. Los robots bípedos surgen de la necesidad del ser humano de imitarse así mismo, sin embargo, los robots actuales están lejos de igualar al humano. Según los resultados obtenidos en Hutter y cols. (2016) al construir el robot cuadrúpedo *ANYmal*, se logra un mejor rendimiento de locomoción en términos de velocidad, eficiencia energética y habilidades de superación de obstáculos con sistemas de múltiples patas.

En estado estático, el peso del cuerpo de un robot con múltiples patas debe ser soportado por al menos tres de ellas, manteniendo al robot estable. Cuando el robot esta en movimiento, el sistema puede estar en contacto con menos de tres patas, lo que permite que el sistema se mueva mucho más rápido y de manera más eficiente, sin embargo, se sacrifica estabilidad y se ocasionan mayores demandas en la actuación y el control.

Algunos ejemplos de robots con múltiples patas creados para funcionar en el espacio son:

- ***SpaceClimber*** (Bartsch, 2014): es un robot de seis patas bioinspirado específicamente creado para la exploración en la Luna con actuadores y electrónica adecuados para el espacio. El robot es capaz de mantenerse estable y de moverse por superficies inclinadas de 25º. Además, consigue realizar el movimiento manteniendo un bajo consumo de energía y un bajo costo computacional. También cuenta con un conjunto de sensores que incluye: sensores táctiles en los pies, cámaras, IMUs y sensores de posición absoluta en las articulaciones.

Figura 3.1: SpaceClimber

- ***Scorpion*** (Dirk y Kirchner, 2007): es un robot de ocho patas muy robusto desarrollado para atravesar terrenos muy empinados y sin estructura, sin la necesidad de planificación a alto nivel ni el uso de sensores exteroceptivos complejos. *Scorpion* es un proyecto iniciado en 1997 y que se ha ido mejorando con el tiempo. Dadas sus características, se ha propuesto como candidato a ser un posible explorador espacial.

Figura 3.2: Scorpion

- ***ALoF*** (Remy y cols., 2011): es un robot de cuatro patas con tres grados de libertad cada una. Las tres articulaciones están configuradas con las dos rodillas enfrentándose entre sí generando fuerzas de contacto con el suelo más simétricas, lo que permite reducir la carga en las articulaciones en ciertas configuraciones y una amplia gama de movimientos. Fue desarrollado por ETH Zurich y probado en la ESA Lunar Robotics Challenge 2008.

  Tiene motores de corriente continua no deformables conjunto a una alta reducción que están protegidos contra el polvo. Además, es capaz de recuperarse tras caer y de gatear, lo que permite subir pendientes empinadas y con suelo suelto.

Figura 3.3: ALoF

3.1.3 Robots de múltiples patas vs robots con ruedas en el espacio

El uso de robots con ruedas está ampliamente utilizado en las misiones de exploración espacial a la Luna y Marte, además, cuenta con una mayor experiencia a diferencia de su contra-parte. Su principal ventaja se debe a su constante desarrollo, su bajo consumo, su menor complejidad, su redundancia y que incluso con solo tres ruedas son capaces de funcionar (Seeni y cols., 2008).

Por otro lado, los robots con múltiples patas pueden ser más complejos y costosos computacionalmente (Seeni y cols., 2008). Sin embargo, presentan ciertas ventajas que los robots con ruedas no tienen y que superan sus limitaciones. En algunas ocasiones, debido al terreno, el robot no puede acceder a una área determinada o las ruedas pueden quedarse enganchadas. La capacidad de colocar sus pies libremente, minimizando el deslizamiento y maximizando la estabilidad, superar obstáculos y lograr la redundancia (Inoue y Kaminogo, 2015) de los robots con múltiples patas los hacen una opción muy estable a considerar para la exploración espacial.

Sin embargo, la comparación es complicada ya que se comparan robots que están actualmente en el espacio con prototipos de robots con múltiples patas utilizados solo en entornos de laboratorio.

3.2 Optimización

En general, un problema de optimización matemático tiene la siguiente forma (Boyd y Vandenberghe, 2004):

$$\begin{aligned} &\textit{miminiza} && f_0(x) && && (3.1a)\\ &\textit{tal que} && f_i \leq b_i x, && i = 1, \ldots, m && (3.1b) \end{aligned}$$

Donde $x = (x_i, ..., x_n)$ es la variable de optimización, f_0: $\mathbb{R}^n \to \mathbb{R}$ es la función objetivo, f_i: $\mathbb{R}^n \to \mathbb{R}$ son las restricciones de desigualdad y b_i son los límites de dichas restricciones.

Lo que se busca es encontrar un vector óptimo x^* que contenga el menor valor objetivo que el resto de variables y que cumplan las restricciones del problema. Sin embargo, el problema anterior es una generalización de cualquier problema de optimización en el que se busca la mejor solución para unas restricciones conocidas.

Dada la diversidad de objetivos y problemas que puede haber, es fundamental categorizar los distintos problemas de optimización, ya que cada uno requiere un enfoque de solución único. Cada categoría se aborda con un método de resolución específico. Por lo tanto, si se puede formular un problema como un problema de optimización perteneciente a una categoría específica, es posible utilizar un optimizador diseñado para resolver problemas similares y obtener una solución adecuada. La efectividad de estos optimizadores depende de muchos factores y su clase viene dada por sus funciones objetivo y sus restricciones.

Por ello, a continuación se presentan las categorías más comunes de problemas de optimización convexa.

3.2.1 Optimización convexa

En este tipo de problemas las funciones $f_0, \ldots, f_m$ deben cumplir la siguiente propiedad:

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y), \forall x, y \in \mathbb{R}^n, \forall \alpha, \beta \in \mathbb{R} \tag{3.2}$$

Donde $\alpha + \beta = 1$, $\alpha \leq 0$, $\beta \leq 0$. A partir de la definición 3.2, se identifican diferentes tipos de problemas de optimización convexa según las formas de las funciones.

- **Mínimos cuadrados**
  Este es un método de optimización que no tiene restricciones, que busca miminizar la siguiente función:

  $$f_0(x) = ||Ax - b||^2 = \sum_{i=1}^{k} (a_i^T x - b_i)^2 \tag{3.3}$$

  Siendo $A \in \mathbb{R}^{kxn}$ (k $\leq$ n), a_i^T las filas de A y $x \in \mathbb{R}^n$ la variable de optimización. Además, debido a que es una función cuadrática (existe un único mínimo global) puede reducirse a una serie de ecuaciones lineales, siendo posible encontrar una solución analítica tal que:

  $$x = (A^T A)^{-1} A^T b \tag{3.4}$$

- **Programación lineal**
  En estos problemas tanto las funciones objetivo como las restricciones son funciones

lineales, por lo que cumplen:

$$f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y), \forall x, y \in \mathbb{R}^n, \forall \alpha, \beta \in \mathbb{R} \tag{3.5}$$

Por lo que el problema queda como:

$$\begin{aligned} &miminiza && c^T x && (3.6a)\\ &tal\ que && Ax \leq b, \quad i = 1, \ldots, m && (3.6b) \end{aligned}$$

Aquí los vectores $c, a_1, \ldots, a_m \in \mathbb{R}^n$ y los escalares $b_1, \ldots, b_m$ son los parámetros del problema que especifican las funciones objetivo y restricciones.

A diferencia de la categoría anterior, este método no se puede resolver de forma analítica, pero existen diferentes algoritmos que los resuelven relativamente rápido con un orden de $O(n^2m)$. Estos algoritmos son bastante confiables, aunque tal vez no tanto como los métodos para mínimos cuadrados. Sin embargo, es posible resolver problemas con cientos de variables y miles de restricciones fácilmente, en segundos.

- **Programación cuadrática**
  Según "Quadratic Programming" (2006) un problema de optimización con una función objetivo cuadrática y restricciones lineales se trata de un programa cuadrático, que se define como:

$$\begin{aligned} &miminiza && q(x) = \frac{1}{2} x^T G x + x^T c && (3.7a)\\ &tal\ que && a_i^T b = b_i, \quad i \in \varepsilon && (3.7b)\\ &y && a_i^T b \leq b_i, \quad i \in \xi && (3.7c) \end{aligned}$$

  Donde G es una matriz simétrica nxn , ε y ξ son un conjunto de índices finitos y donde c,x y a_i son vectores en $\mathbb{R}^n$. Estos problemas pueden resolverse en una cantidad finita de cálculos, pero el esfuerzo requerido para encontrar una solución depende de las características de la función objetivo y del número de restricciones.

  Si la matriz G es semidefinida positiva, se trata de un problema cuadrático convexo, y el problema es similar en dificultad a un programa lineal. Si G es una matriz indefinida, estos problemas son más complicados ya que pueden tener varios puntos estacionarios y mínimos locales.

  Al contrario que en los mínimos cuadrados y al igual que en la programación lineal, no existen soluciones analíticas para estos problemas. Existen diferentes algoritmos que se han ido desarrollando para solucionar estos problemas. Principalmente se dividen en tres: *Métodos del conjunto activo*, *Métodos de primer orden* y *Métodos de los puntos interiores.*

3.2.2 Optimización no lineal

Afín con Boyd y Vandenberghe (2004), Nonlinear Programming (NLP) es el término utilizado para describir un problema de optimización donde las funciones objetivo o de restricción no son lineales y no se conoce si son convexas. Debido a la complejidad que esto atribuye no

existen métodos efectivos para resolver estos problemas, ya que incluso con pocas variables, puede tratarse de problemas muy complicados o imposibles de resolver. Principalmente, se diferencian dos métodos de aproximación para este tipo de programación: local y global.

3.2.2.1 Optimización local

El punto de la optimización local es abandonar la idea de encontrar la variable objetivo que minimice todos y cada uno de los puntos factibles. En su lugar, se busca encontrar un óptimo local que satisfaga los puntos más cercanos. Estos métodos pueden ser rápidos, manejar problemas muy grandes y ser ampliamente aplicables, ya que solo requieren que las funciones objetivo y de restricción sean diferenciables.

Sin embargo, a parte de no encontrar la solución óptima global, existen varias desventajas de estos métodos. Requieren una suposición inicial para la variable de optimización, afectando de gran manera al valor objetivo de la solución local obtenida y además, se proporciona poca información sobre la distancia de la solución local a la óptima global.

Estos métodos son más complejos que los explicados en el apartado 3.2.1, ya que es necesario ajustar los parámetros del algoritmo y encontrar una suposición inicial lo suficientemente buena.

Algunos de los optimizadores locales más utilizados en la industria son SNOPT (Gill y cols., 2002) e IPOPT (Wächter y Biegler, 2006).

3.2.2.2 Optimización global

A coste de la eficiencia, el objetivo final de la optimización global es encontrar la solución global al problema de optimización 3.1. En el peor de los casos, estos problemas pueden llegar a crecer a un orden exponencial tanto para el número de variables como para las restricciones. Debido a estas limitaciones, este método se utiliza en problemas con un bajo número de variables.

3.3 Optimización de trayectorias

Según se describe en Kelly (2017) el concepto de "*optimización de trayectorias*" se refiere al conjunto de métodos utilizados para determinar la mejor trayectoria, seleccionando las entradas del sistema como funciones temporales.

En estos métodos se busca ajustar una serie de variables, denominadas comúnmente como "*variables de decisión*" que minimizan la función objetivo. Normalmente, estas variables son: el tiempo inicial y final y el estado y control de la trayectoria. Generalmente, la función objetivo tiene la siguiente forma (Kelly, 2017):

$$\min_{t_0,,t_f,x(t_0),x(t_f)} J(t_0,t_f,x(t_0),x(t_f)) + \int_{t0}^{tf} (w\tau, x(\tau), u(\tau))d\tau \tag{3.8}$$

Además de las funciones objetivo, existen una serie de restricciones que el sistema debe cumplir. Algunas de las más importantes son las siguientes:

- **Dinámica del sistema**: determina como cambia el sistema en función del tiempo.

$$\dot{x}(t) = f(t, x(t), u(t)) \tag{3.9}$$

- **Restricción de ruta**: limita el movimiento de la trayectoria. Este tipo de restricción se utiliza, por ejemplo, para mantener la pata de un robot por encima del suelo.

$$h(t, x(t), u(t)) \leq 0 \tag{3.10}$$

- **La restricción de límites no lineal**: impone restricciones en los estados inicial y final del sistema, que se pueden utilizar para garantizar que la marcha de un robot caminante sea periódica.

$$g(t_0, t_F, x(t_0), x(t_F)) \leq 0 \tag{3.11}$$

- **Límites constantes en el estado o control**: como los límites de una articulación de un robot o el límite de torque que se le puede aplicar.

$$x_{low} \leq x(t) \leq x_{upp} \tag{3.12a}$$
$$u_{low} \leq u(t) \leq u_{upp} \tag{3.12b}$$

- **Límites en el tiempo y estado inicial y final**: se incluyen cuando se quiere que la solución al problema alcance su objetivo en un tiempo determinado o que alcance un estado concreto del espacio del estados.

$$t_{low} \leq t_0 < t_F \leq t_{upp} \tag{3.13a}$$
$$x_{0,low} \leq x(t_0) \leq x_{0,upp} \tag{3.13b}$$
$$x_{F,low} \leq x(t_F) \leq x_{F,upp} \tag{3.13c}$$

Para solucionar estos problemas existen dos tipos de aproximaciones: *Open-Loop* y *Closed-Loop* (Kelly, 2017).

3.3.1 Open-Loop

En estos casos, la solución de la optimización es encontrar una secuencia de controles como una función temporal que permita llevar al sistema desde un estado inicial hasta un estado final. Normalmente, para poder aplicar estas soluciones a una aplicación real es necesario combinarla con un controlador. Se utilizan normalmente en problemas muy grandes y de alta dimensionalidad. Dentro de esta aproximación se diferencian dos principales métodos: métodos directos e indirectos.

- **Métodos directos**: estos métodos consisten en discretizar el problema de optimización de trayectorias en un problema no lineal. A este proceso se le llama "transcripción", y consiste en discretizar el problema de optimización continuo aproximando todas las

funciones continuas del enunciado como *splines* polinómicos. Algunos de estos métodos son: *Colocación Trapezoidal*, *Disparo Único Directo* y *Disparo Múltiple Directo*.

- **Métodos indirectos**: consisten en construir las condiciones necesarias y suficientes para la optimabilidad, tras ello se discretizan estas condiciones y se resuelven numéricamente. Una ventaja de estos métodos es que, generalmente, tienen una precisión mayor y una estimación de error más confiable. Sin embargo, necesitan una inicialización muy buena y construir las condiciones, procesos muy complicados de realizar.

Normalmente, estos métodos se distinguen porque el directo discretiza y después optimiza, mientras que los indirectos realizan el proceso contrario.

3.3.2 Closed-Loop

También denominadas como "Programación dinámica". Estas técnicas, en lugar de encontrar la trayectoria óptima buscan la política óptima, es decir, el control óptimo para cada punto en el espacio. Se utilizan normalmente en problemas de dimensiones bajas pero con espacios complejos. A diferencia del otro método, estos siempre encuentran la solución global, sin embargo, el coste es exponencial.

Figura 3.4: Comparación entre un problema *Open-Loop* y otro *Closed-Loop* (Kelly, 2017)

4 Metodología

4.1 Software

4.1.1 ROS

ROS (Quigley y cols., 2009) es una colección de herramientas, bibliotecas y convenciones que se utilizan para desarrollar software en el campo de la robótica. Pese a su nombre, se trata de un *middleware* que se ejecuta sobre un sistema operativo real, como Linux. Además, es un sistema de código abierto bajo la licencia BSD, permitiendo libertad en uso comercial e investigación y facilitando su uso a todo tipo de usuarios, tanto individuales como empresas o instituciones.

Una de las características clave es su arquitectura de nodos. Los nodos de ROS son procesos independientes que se comunican entre sí a través de un sistema de mensajería mediante *topics* o mediante llamadas a servicios. Esto permite una gran flexibilidad y modularidad en el desarrollo de aplicaciones robóticas, ya que los nodos pueden ser escritos en diferentes lenguajes de programación y ejecutarse en diferentes computadoras.

Es ampliamente utilizado en la comunidad de robótica y ha sido adoptado como un estándar en la industria. Las herramientas y bibliotecas que proporciona permite una realizar una amplia gama de tareas como percepción, planificación de movimiento, navegación y manipulación.

Alguna de estas herramientas son, por ejemplo, Rviz (Kam y cols., 2015), que permite la visualización y representación 3D de robots, nubes de puntos, datos de sensores…. Y bibliotecas como *rosbag* para guardar los diferentes mensajes que se envían por los *topics*.

4.1.2 Pinocchio

Pinocchio (Carpentier y cols., 2019) es una biblioteca que implementa los algoritmos del libro *Rigid Body Dynamics Algorithms* (Featherstone, 2007) capaz de proporcionar de forma automática las derivadas analíticas de sus algoritmos. Además, adopta un enfoque modular al separar el modelo del robot y los datos específicos de una instancia particular del robot. El modelo del robot se define utilizando la clase **pinocchio::Model**, que contiene toda la información estática del robot, como la estructura cinemática, los nombres de las articulaciones…. Por otro lado, los datos del robot se almacenan en una instancia **pinocchio::Data**. Esta clase contiene la información dinámica del robot en un instante dado, como las posiciones y velocidades de las articulaciones, los momentos, la matriz de inercias….

4.1.3 Plotjuggler

Plotjuggler (s.f.) es una herramienta rápida e intuitiva para la visualización de datos en tiempo real. Es capaz de conectarse a una aplicación externa utilizando cualquier comunicación entre procesos y mostrar los datos. Además, es totalmente compatible con ROS lo que

significa que puede aprovechar sus capacidades para cargar archivos de registro (*rosbags*), suscribirse a *topics* y republicar mensajes para visualizarlos en Rviz.

A parte de de sus funciones básicas, *Plotjuggler* ofrece características avanzadas que mejoran el análisis de datos. Se pueden filtrar y manipular los datos para destacar la información relevante. Además, la herramienta permite sincronizar y comparar múltiples conjuntos de datos, lo que facilita la visualización y comprensión de la interacción entre diferentes variables en los sistemas robóticos en funcionamiento.

4.2 Planificación de trayectorias

Para realizar la planificación de las trayectorias se utiliza *towr* (Trayectory Optimizer for Walking Robots) creado por Winkler y cols. (2018). *towr* es una biblioteca de código abierto escrita en C++ creada principalmente para la optimización de trayectorias de robots con patas como el robot *HyQ* (Semini y cols., 2011) o ANYmal (Hutter y cols., 2016). Esta biblioteca permite implementar un conjunto de variables, costos y restricciones que se pueden usar para representar un problema de locomoción no lineal. Dicho problema puede resolverse utilizando optimizadores como IPOPT o SNOPT implementados en *ifopt* (Winkler, 2018), una interfaz basada en Eigen (Guennebaud y cols., 2010) para optimizadores de problemas no lineales creada por el mismo autor que *towr*. Finalmente, para visualizar las trayectorias se utilizada un paquete llamado *xpp* (Winkler, 2017). Dicho paquete utiliza *Rviz* para mostrar todos los datos y los propios robots incluidos en *towr* mediante su fichero URDF.

El principal motivo de su uso en el proyecto es debido a que sus desarrolladores han implementado una interfaz que permite utilizarlo de forma sencilla en ROS, junto a la posibilidad de poder descargar estas herramientas en forma de paquetes.

Debido a esta sencillez de uso se ha utilizado en diversos proyectos como *POPI* (2021).

4.2.1 Objetivos

towr tiene como objetivo obtener la posición y orientación del centro de masas y los movimientos y fuerzas de contacto para cada pata. Para llevarlo a cabo se proporciona el estado inicial y final deseado del sistema, la duración total y la cantidad de pasos. Sin embargo, no es necesario proveer el intervalo de tiempo de cada paso, ya que es el controlador el que lo determina automáticamente.

Utiliza polinomios de cuarto orden de duración fija y concatenados para crear un *spline* continuo y optimizar los coeficientes del polinomio. Para el movimiento de cada pata, se utilizan varios polinomios de tercer orden por cada fase de vuelo, y un valor constante para la fase de apoyo. Mientras que, para el perfil de fuerza de cada pata, se usan varios polinomios que representan cada una de las fases de apoyo mientras que se establece una fuerza nula durante la fase de vuelo. Así pues, la duración de cada fase, y por lo tanto la duración de cada polinomio de cada pata, se modifica en función de la duración de las fases optimizadas.

4.2.2 Restricciones

Como se ha comentado anteriormente se deben definir una serie de restricciones que deben cumplir las trayectorias que se generan en *towr*. A continuación se detallan dichas restricciones:

4.2.2.1 Restricciones de la cinemática

En lugar de imponer restricciones directas sobre los ángulos de las articulaciones, *towr* utiliza un enfoque que se basa en restringir las posiciones cartesianas que las patas pueden alcanzar. Para lograr esto, se utiliza un prisma rectangular que tiene una posición relativa con respecto a la base del robot. Este prisma define un espacio de trabajo válido para las extremidades, y las posiciones finales de cada pata deben mantenerse dentro de este espacio en todo momento. De esta manera, se garantiza que las patas puedan alcanzar las posiciones deseadas sin violar las limitaciones impuestas por las articulaciones. Esta condición se debe cumplir de manera regular cada cierto periodo de tiempo.

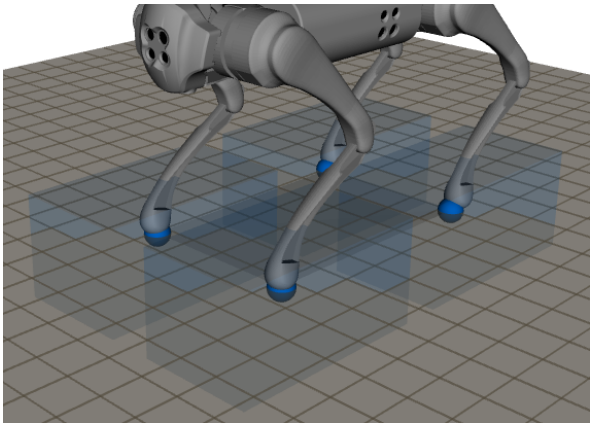


Figura 4.1: Prismas rectangulares de cada pie

4.2.2.2 Restricciones del modelo dinámico

Tal y como se explica más adelante en el apartado 4.4 se elige trabajar con el modelo del sólido rígido único. Debido a las suposiciones que establece y que solo se trabaja con los parámetros físicos de inercia y masa constantes permite simplificar significativamente la optimización y la generación de las trayectorias.

La restricción consiste en hacer cumplir las ecuaciones de Newton-Euler 4.6a y 4.6b cada cierto tiempo y además, se añade la restricción de que las aceleraciones entre dos *splines* sean iguales, ya que si no fuera así esto implicaría saltos en las fuerzas o en las posiciones de las patas, es decir, discontinuidades.

4.2.2.3 Mapas de altura

Para que una pata del robot se considere en contacto con el suelo, es necesario que la altura de la pata en esa posición bidimensional coincida con la altura del terreno en ese mismo punto. Además, es importante garantizar que en ningún momento las patas del robot se encuentren por debajo de la superficie.

Para representar los terrenos en los cuales se desarrolla la locomoción, se utilizan mapas de altura. Estos mapas permiten generar diferentes tipos de escenarios y configuraciones de terreno, lo que a su vez resulta en la generación de trayectorias específicas para el robot.

Esto permite evaluar el desempeño del robot en diferentes entornos y analizar cómo se adaptan sus trayectorias y movimientos a las condiciones del terreno.

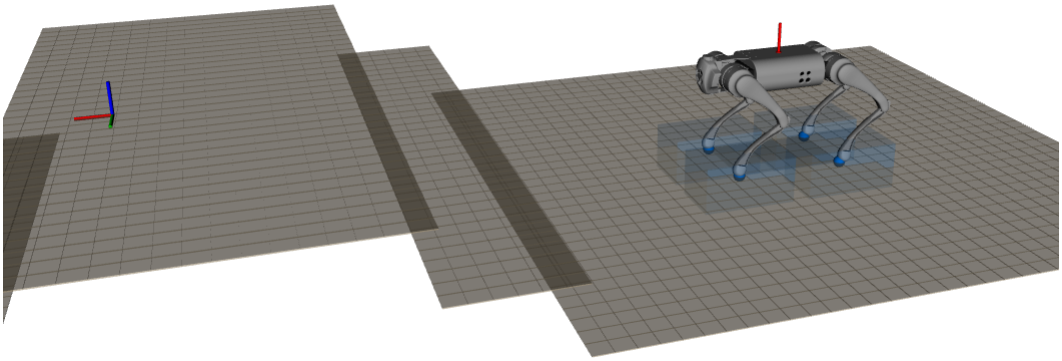


Figura 4.2: Mapa de altura con escalones

4.2.2.4 Contactos con el terreno

El modelo dinámico comentado es general y puede aplicarse tanto a drones voladores como a sistemas con patas. Sin embargo, en los sistemas con patas, las restricciones sobre las fuerzas y posiciones de los pies cambian abruptamente cuando una pata está en contacto con el entorno. Estas transiciones discretas y las restricciones asociadas son difíciles de manejar en formulaciones tradicionales de optimización. Para abordar esto, *towr* trata cada pata de forma individual y simplifica los conceptos de fases y configuraciones de contacto a una representación binaria (contacto o no contacto). Esto le permite manejar las discontinuidades en el movimiento y las fuerzas de los pies de manera más efectiva.

4.2.2.5 Restricciones en la fase de apoyo

Con el fin de lograr una locomoción físicamente correcta, es fundamental que las fuerzas generadas por las patas del robot sean exclusivamente de empuje y nunca de tracción sobre la superficie de contacto. Esta restricción es esencial para garantizar la estabilidad de los contactos y prevenir caídas. Para mantener la estabilidad, se establece que las fuerzas tangenciales deben permanecer dentro de un cono de fricción en todo momento. Al asegurar el cumplimiento de esta restricción, se garantiza que las fuerzas se mantengan dentro de los límites de fricción y se preserve la estabilidad del contacto entre las patas y la superficie.

4.2.3 Formulación del problema

Una vez explicados los objetivos de *towr* y sus restricciones se muestra la formulación del problema de trayectorias a optimizar:

$$
\begin{aligned}
\text{find} \quad & \mathbf{r}(t) \in \mathbb{R}^3 && \text{(CoM linear position)} \\
& && \texttt{duration\_base\_polynomial\_} \\
& \boldsymbol{\theta}(t) \in \mathbb{R}^3 && \text{(base euler angles)} \\
& && \texttt{duration\_base\_polynomial\_} \\
& \text{for every foot } i: \\
& \quad \Delta T_{i,1} \ldots, \Delta T_{i,2n_{s,i}} \in \mathbb{R} && \text{(phase durations)} \\
& && \texttt{ee\_phase\_durations\_} \\
& \quad \mathbf{p}_i(t, \Delta T_{i,1}, \ldots) \in \mathbb{R}^3 && \text{(foot position)} \\
& && \texttt{ee\_polynomials\_per\_swing\_phase\_} \\
& \quad \mathbf{f}_i(t, \Delta T_{i,1}, \ldots) \in \mathbb{R}^3 && \text{(force at foot)} \\
& && \texttt{force\_polynomials\_per\_stance\_phase\_} \\
\text{s.t.} \quad & [\mathbf{r}, \boldsymbol{\theta}](t{=}0) = [\mathbf{r}_0, \boldsymbol{\theta}_0] && \text{(initial state)} \\
& \mathbf{r}(t{=}T) = \mathbf{r}_g && \text{(desired goal)} \\
& [\ddot{\mathbf{r}}, \dot{\boldsymbol{\omega}}]^T = \mathbf{f}_d(\mathbf{r}, \mathbf{p}_1, \ldots, \mathbf{f}_1, \ldots) && \text{(dynamic model)} \\
& && \texttt{dt\_constraint\_dynamic\_} \\
& \text{for every foot } i: \\
& \quad \mathbf{p}_i(t) \in \mathcal{R}_i(\mathbf{r}, \boldsymbol{\theta}), && \text{(kinematic model)} \\
& && \texttt{dt\_constraint\_range\_of\_motion\_} \\
& \quad \text{if foot } i \text{ in contact}: \\
& \quad\quad \dot{\mathbf{p}}_i(t \in \mathcal{C}_i) = \mathbf{0} && \text{(no slip)} \\
& \quad\quad p_i^z(t \in \mathcal{C}_i) = h_{terrain}(\mathbf{p}_i^{xy}) && \text{(terrain height)} \\
& \quad\quad \mathbf{f}_i(t \in \mathcal{C}_i) \cdot \mathbf{n}(\mathbf{p}_i^{xy}) \geq 0 && \text{(pushing force)} \\
& \quad\quad \mathbf{f}_i(t \in \mathcal{C}_i) \in \mathcal{F}(\mu, \mathbf{n}, \mathbf{p}_i^{xy}) && \text{(friction cone)} \\
& \quad \text{if foot } i \text{ in air}: \\
& \quad\quad \mathbf{f}_i(t \notin \mathcal{C}_i) = \mathbf{0} && \text{(no force in air)} \\
& \quad \textstyle\sum_{j=1}^{2n_{s,i}} \Delta T_{i,j} = T && \text{(total duration)} \\
& && \texttt{ee\_phase\_durations\_}
\end{aligned}
$$

Figura 4.3: Problema a resolver (Winkler y cols., 2018)

4.3 Estudio de la cinemática

Antes de proceder a cualquier tarea relacionada con la programación o control de un robot, resulta crucial realizar un análisis de la cinemática del mismo. La cinemática es una disciplina mecánica que se encarga del estudio de los movimientos y posiciones de los cuerpos sin considerar las fuerzas que los producen. En el contexto de la robótica, la cinemática se refiere a la descripción matemática precisa de los movimientos de las articulaciones y extremidades del robot.

El estudio de la cinemática es primordial para entender la estructura y el movimiento del robot, y es esencial para la programación, control, diseño y optimización del mismo. Por ejemplo, el análisis de la cinemática permite determinar el espacio de trabajo del robot, la cantidad de movimientos y grados de libertad disponibles, así como las posibles limitaciones físicas y mecánicas. Además, proporciona información sobre la capacidad del robot para realizar tareas específicas.

De esta manera y teniendo esto en cuenta, a continuación se realiza el estudio cinemático del robot utilizado en este proyecto, el Go1 de Unitree Robotics.

4.3.1 Cinemática directa

La cinemática directa se refiere al cálculo de la transformación geométrica entre el sistema de coordenadas de la base y el sistema de coordenadas de la herramienta final del robot. Este cálculo se basa en la geometría y la mecánica del robot, y se utiliza para determinar la posición y orientación del efector final en función de las posiciones de las articulaciones del robot.

En este proyecto se procede a realizar el estudio de la cinemática directa de la pata delantera izquierda del robot Go1 mediante el método de DH. Esto es debido a la simetría del diseño del robot, por lo que las otras patas son idénticas en términos de su geometría y configuración. Una vez que se haya realizado el estudio de la cinemática directa de dicha pata, es posible extrapolar esta información a las otras patas del robot.

El primer paso del método DH es establecer un sistema de coordenadas para cada articulación del robot. Una vez establecidos, cada sistema se define en función del sistema de coordenadas del sistema anterior, utilizando una serie de parámetros que describen la relación entre las articulaciones. Estos parámetros incluyen:

- **El ángulo de rotación**: ángulo que habría que girar en torno a z_{i-1} para que x_{i-1} y x_i queden paralelos
- **La longitud del enlace**: distancia medida sobre z_{i-1} que habría que desplazar S_{i-1} para alinear x_{i-1} y x_i
- **La distancia de desplazamiento**: distancia medida sobre x_i (que ahora coincidiría con x_{i-1}) que habría que desplazar el nuevo S_{i-1} para que su origen coincidiese con Si.
- **El ángulo de desviación**: ángulo que habría que girar en torno a x_{i-1} (que ahora coincidiría con x_i) para que el nuevo S_{i-1} coincidiese totalmente con Si.

Tras obtener los parámetros del DH se calculan las matrices de transformación entre un sistema y el anterior.

$$T_i^{i-1} = \begin{bmatrix} cos\theta_i & -cos\alpha_i * sen\theta_i & sen\alpha_i * cos\theta_i & a_i * cos\theta_i \\ sen\theta_i & cos\alpha_i * cos\theta_i & -sen\alpha_i * cos\theta_i & a_i * sen\theta_i \\ 0 & sen\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

Finalmente, se multiplican estas matrices para obtener la matriz de transformación que relaciona el sistema de la base con el del extremo del robot.

$$T = T_1^0 T_1^2 \dots T_n^{n-1} \tag{4.2}$$

De esta manera, la matriz resultante define la posición y orientación del extremo del robot respecto a la base en función de las n coordenadas articulares.

A continuación se muestra la tabla DH del Go1:

θ_i	d_i	a_i	α_i
0.243602	0	0.193823	0
-0.243602	0	0	0
$-\pi/2$	0	0	$-\pi/2$
$q_1+\pi$	0	0.08	$-\pi/2$
$-\pi/2$	0	0	0
0	0	0	$-\pi/2$
$q_2+\pi$	0	0	$-\pi/2$
π	-0.213	0	0
0	0	0	$-\pi/2$
$q_3+\pi$	0	0	$-\pi/2$
π	-0.213	0	0

Figura 4.4: Tabla DH del Go1 a partir de su URDF (Byeonggi y cols., 2023)

4.3.2 Cinemática inversa

La cinemática inversa es el proceso de calcular las posiciones de las articulaciones para que el extremo final del robot alcance una posición y orientación específica en el espacio tridimensional.

La cinemática inversa es un problema complejo debido a la no linealidad de las ecuaciones que relacionan las posiciones de las articulaciones con la posición y orientación del extremo del robot y cuya solución en ocasiones no es única. Esto se debe a que la geometría del robot es compleja y varía dependiendo de la configuración de las articulaciones.

En muchos casos, no existen soluciones analíticas exactas para la cinemática inversa, lo que implica que es necesario recurrir a métodos numéricos y algoritmos de optimización para encontrar soluciones aproximadas. Los métodos numéricos más comunes para resolver la cinemática inversa incluyen el método de Newton-Raphson y el método de iteración inversa.

Sin embargo, en el caso del robot Go1 no es necesario el uso de estos algoritmos complejos. Al igual que en la cinemática directa, se va a realizar un estudio de la cinemática inversa de la pata delantera izquierda, ya que al ser simétrico, el mismo proceso se puede extrapolar al resto de patas.

El método que se utiliza para resolver el problema es la trigonometría. Esto se debe a que la pata esta formada por tres articulaciones rotacionales, lo que significa que su geometría es relativamente simple y puede ser modelada de manera precisa utilizando funciones trigonométricas.

En un inicio el sistema de coordenadas cartesianas del robot GO1 está referenciado a la base que se sitúa en el centro del cuerpo del robot. Sin embargo, para realizar la cinemática inversa, es necesario traducir las coordenadas al sistema local de la pata.

(a) Coordenadas cartesianas del extremo referenciadas a la base

(b) Coordenadas cartesianas del extremo referenciadas a la cadera

Figura 4.5: Cambio del sistema de coordenadas de base a cadera

Al estar el sistema situado sobre la articulación de la cadera y conocer las posiciones del efector final, es posible obtener el ángulo de la primera articulación (q1) mediante el arcotangente.

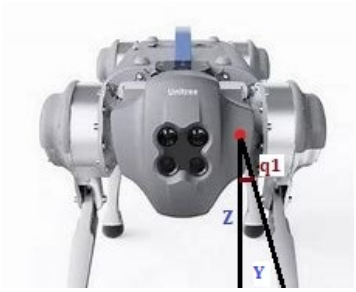


Figura 4.6: Obtención del ángulo de la cadera

$$q1 = -atan\frac{Y_c}{-Z_c} \tag{4.3}$$

Una vez obtenido, se rota el vector de posición al sistema de coordenadas del muslo utilizando una matriz de rotación construida mediante el ángulo calculado anteriormente y se traslada al sistema de coordenadas del muslo para ajustarlo a la distancia entre la articulación de la cadera y este.

De esta forma, las posiciones cartesianas quedan referenciadas al sistema de coordenadas del muslo. Así, conocidas las longitudes de ambas partes de la pata y las coordenadas de la posición final del extremo de esta, se pueden obtener tanto el ángulo de la articulación del muslo como el del tobillo.

La suma de los ángulos α y β que se observan en la imagen 4.8 da lugar al ángulo del muslo (q2). Es posible obtener α mediante el triángulo rectángulo formado por las posiciones del

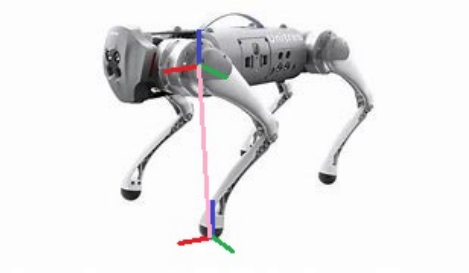


Figura 4.7: Cambio del sistema de coordenadas al muslo con q1=0

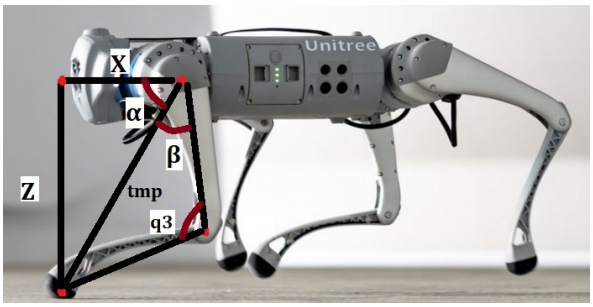


Figura 4.8: Cálculo de los dos ángulos restantes

extremo del robot y β mediante el teorema del coseno.

$$\alpha = atan\frac{-Z_m}{X_m} - 0.5 * \pi \tag{4.4a}$$

$$\beta = acos(\frac{lu^2 + tmp^2 - ll^2}{2 * lu * tmp}) \tag{4.4b}$$

$$q2 = \alpha + \beta \tag{4.4c}$$

donde: $X_m \rightarrow$ Es la distancia en X desde el muslo al efector final.
$Z_m \rightarrow$ Es la distancia en Z desde el muslo al efector final.
$lu \rightarrow$ Es la longitud de la parte superior de la pata.
$ll \rightarrow$ Es la longitud de la parte inferior de la pata.
$tmp \rightarrow$ Es la hipotenusa del triángulo rectángulo.

Finalmente, el cálculo del último ángulo, el del tobillo (q3) es posible calcularlo también mediante el teorema del coseno.

$$q3 = acos(\frac{ll^2 + lu^2 - tmp^2}{2 * ll * lu}) - \pi \tag{4.5}$$

4.4 Modelo dinámico

La dinámica se ocupa de la relación entre las fuerzas que actúan sobre un cuerpo y el movimiento que éstas generan. Comprende el análisis de cómo las fuerzas externas y las

fuerzas internas generadas por las articulaciones y actuadores afectan al movimiento y al equilibrio del robot.

En el caso del modelo dinámico de un robot, se trata de conocer la relación entre el movimiento y las fuerzas que se originan. En el proyecto propuesto se decide trabajar con el modelo del sólido rígido único debido a las suposiciones que establece:

- **Cuerpo rígido**: Se supone que el robot es un cuerpo rígido, lo que significa que no se deforma al aplicar fuerzas externas.
- **Articulaciones sin fricción ni momento**: El modelo asume que las articulaciones del robot no tienen fricción y que los momentos producidos por las velocidades articulares son despreciables.
- **Inercia constante**: Se supone que la masa y los momentos de inercia del cuerpo rígido son constantes y no cambian durante el movimiento siendo equivalente a las obtenidas al dejar al robot en su estado articular de reposo.
- **Interacción con el entorno**: Se considera que el cuerpo rígido solo interactúa con su entorno a través de fuerzas externas específicas, como la gravedad u otras fuerzas aplicadas externamente.

Debido a todas estas suposiciones este modelo se utiliza en robots cuya masa se encuentra centrada mayoritariamente en la base y cuyas extremidades prácticamente no tengan masa, como es el caso del robot Go1. Es por eso que el modelo dinámico de un cuerpo rígido simple y único es apropiado en este caso ya que, al tener poca masa en las patas, los momentos generados como resultado de las fuerzas aplicadas pueden ser considerados despreciables.

Además, la inercia del robot no cambia significativamente durante su movimiento debido al mismo motivo, lo que simplifica aún más el modelo dinámico, por lo que se puede asumir que la masa y los momentos de inercia se mantienen constantes.

Finalmente, debido a que el robot se considera un cuerpo rígido al cual solo se le aplican fuerzas externas, su movimiento se puede definir mediante las ecuaciones de Newton-Euler:

$$m\ddot{r} = \sum_{i=1}^{n_i} f_i(t) - mg \tag{4.6a}$$

$$I\dot{w}(t) + w(t) \times Iw(t) = \sum_{i=1}^{n_i} f_i(t) \times (r(t) - p_i(t)) \tag{4.6b}$$

donde:
- m → Es la masa del robot.
- r → Es la posición del centro de masas.
- n_i → Es el número de patas.
- g → Es el valor de la gravedad.
- w → Es la velocidad angular de la base.
- $\mathrm{I} \in \mathbb{R}^{3\times3}$ → Es la inercia constante del cuerpo completo del robot con respecto al centro de masas con sus articulaciones en posición de reposo.

4.5 Simulación

Para validar las trayectorias generadas por *towr* con diferentes gravedades y la aplicación de la cinemática inversa a estas, se decide realizar su comprobación de manera virtual mediante un simulador de físicas antes de proceder a aplicar algunas de ellas al robot real. Debido a que hasta el momento la programación y todo el trabajo anterior se ha realizado mediante ROS se elige *Gazebo* (Koenig y Howard, 2004) para realizar las simulaciones, siendo este el simulador por excelencia de ROS (Ivaldi y cols., 2014).

Gazebo es un simulador capaz de funcionar principalmente con cuatro motores de físicas diferentes: Open Dynamics Engine (ODE), *Bullet*, Dynamic Animation and Robotics Toolkit (DART) y *Simbody*. Por defecto, *Gazebo* puede trabajar con ODE, *Bullet* o *Simbody* mientras que DART tiene que ser compilado por separado.

- **ODE** (*ODE*, s.f.): es una librería de código abierto escrita en C/C++ especializada en simulación de cuerpos rígidos. Esto incluye el cálculo de colisiones, restricciones y movimientos de cuerpos rígidos bajo la influencia de fuerzas y momentos. Es el motor principal de *Gazebo* y uno de los más utilizados.

- ***Bullet*** (Coumans y Bai, 2016–2021): es una librería de código abierto escrita en C/C++ que permite el cálculo y la simulación del comportamiento dinámico de sistemas robóticos, capaz de manejar colisiones entre múltiples elementos en un entorno virtual y capaz simular objetos deformables y líquidos.

- ***Simbody*** (*Simbody*, s.f.): es otra biblioteca de código abierto escrita en C++ de alto rendimiento para la simulación de mecanismos articulados, sistemas mecánicos como robots, vehículos y máquinas, y cualquier otro sistema que pueda describirse como un conjunto de cuerpos rígidos interconectados por articulaciones, influenciados por fuerzas y movimientos, y restricciones. Se le conoce a veces como un motor de física estilo Featherstone ya que tiene un enfoque basado en la formulación desarrollada por el Dr. Roy Featherstone para la simulación eficiente de cuerpos rígidos interconectados (Featherstone, 2007).

- **DART** (Lee y cols., 2018): al igual que el resto, también se trata de una biblioteca de código abierto. Ha sido desarrollada por el Instituto Tecnológico de Georgia y proporciona estructuras de datos y algoritmos para aplicaciones cinemáticas y dinámicas en robótica y animación por computadora. Se destaca por su precisión y estabilidad, que se logran mediante el uso de coordenadas generalizadas para representar sistemas de cuerpos rígidos articulados y la aplicación del Algoritmo de Cuerpo Articulado de Featherstone (Featherstone, 2007) para calcular la dinámica del movimiento.

Para realizar las simulaciones de las trayectorias se ha elegido ODE como el motor de físicas ha utilizar. Esto se debe a su prominencia y amplio uso en *Gazebo* donde se ha establecido como el motor de físicas por excelencia, tal y como se ha comentado anteriormente.

Además, esta elección se ve respaldada aún más por el hecho de que los mundos creados por *Unitree* en su paquete de simulación están específicamente diseñados y configurados para funcionar con este motor de físicas. Esto significa que los modelos y escenarios proporcionados

están optimizados y ajustados para aprovechar al máximo las capacidades y características de ODE.

Por tanto, al utilizar dicho motor de físicas, se garantiza la compatibilidad y coherencia entre el entorno de simulación proporcionado por la empresa y el motor subyacente en *Gazebo* asegurando un rendimiento óptimo y resultados realistas al simular el robot y su interacción con el entorno.

4.6 Seguimiento de trayectorias

Una vez realizada la trayectoria en *towr*, este guarda la información resultante en un archivo *rosbag* para su posterior análisis y visualización. Este archivo contiene datos importantes sobre la base del robot y las diferentes patas durante la trayectoria generada y esta compuesto por los siguientes elementos:

- **base**: almacena la información de la posición, aceleración y giro de la base del robot a lo largo de la trayectoria. Estos datos permiten comprender cómo se mueve y se orienta el cuerpo del robot durante la ejecución.
- **ee_contact**: un vector de booleanos que indica si cada pata del robot está en contacto con el suelo o no en cada instante de tiempo.
- **ee_forces**: vector 3D que contiene la información sobre las fuerzas ejercidas por cada pata del robot en formato cartesiano.
- **ee_motion**: guarda información sobre la posición, velocidad y aceleración de cada pata en coordenadas cartesianas.

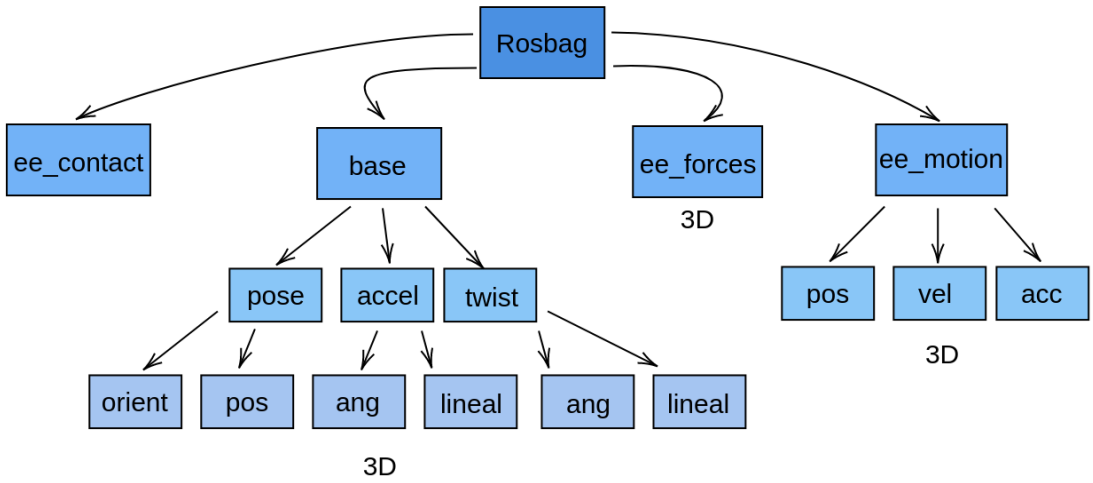


Figura 4.9: Diagrama del rosbag

Para lograr un seguimiento preciso de la trayectoria generada por *towr* tanto en simulación como en el robot real, se utilizan los paquetes desarrollados por *Unitree Robotics*. Estos

paquetes proporcionan un controlador virtual para poder simular trayectorias en *Gazebo* y como establecer una conexión con el robot para enviar los datos en tiempo real a través de User Datagram Protocol (UDP).

Teniendo todo esto en cuenta se crea un archivo en C++ que lee cada uno de los elementos del rosbag generado por *towr*. Para cada una de las iteraciones y para cada pata se realiza la cinemática inversa de las posiciones cartesianas y se transforman las fuerzas cartesianas a pares articulares. Tras ello, el programa publica los datos transformados en el *topic* correspondiente a cada articulación. Finalmente, estos datos son recogidos por el controlador para realizar el movimiento en la simulación.

En el caso del seguimiento de la trayectoria en el robot real, los datos, en lugar de ser publicados en el *topic* correspondiente a su articulación, se publican todos en un único *topic* que recibe otro archivo. Este archivo envía los datos recibidos por el *topic* al robot mediante UDP, donde el controlador del Go1 se encarga de ejecutar los movimientos.

5 Desarrollo

En el capítulo anterior, se ha realizado un estudio de los elementos fundamentales para realizar este proyecto. Se ha abordado el funcionamiento de la biblioteca *towr*, que desempeña un papel clave en la consecución de los objetivos planteados. Además, se ha explorado la cinemática directa e inversa del robot Go1, lo cual resulta esencial para el desarrollo de las trayectorias deseadas. Asimismo, se ha establecido el modelo dinámico que se utilizará en el proyecto y finalmente, el método que se va a utilizar para realizar el seguimiento de las trayectorias y su simulación.

En este capítulo, se realiza la implementación práctica del proyecto, aprovechando las herramientas y conceptos discutidos anteriormente. De esta manera, se pueden aplicar de manera efectiva los conocimientos teóricos adquiridos en la etapa de estudio y llevarlos así a la práctica.

5.1 Planificación de trayectorias

5.1.1 Modelo cinemático y dinámico en towr

Para que *towr* pueda planificar las trayectorias para el Go1 es necesario incluir dicho robot en la biblioteca. El primer paso se trata de definir el modelo cinemática y dinámico del robot para que *towr* sea capaz de generar las trayectorias cartesianas. Para ello, se crea un archivo llamado **go1_model.h** en la carpeta **towr/towr/include/towr/model/examples** cuyo contenido es el siguiente:

Código 5.1: Definición del modelo cinématico y dinámico del Go1 en towr

```
class Go1KinematicModel : public KinematicModel {
public:
  Go1KinematicModel () : KinematicModel(4)
  {
    const double x_nominal_b = 0.19;
    const double y_nominal_b = 0.125;
    const double z_nominal_b = -0.32;

    nominal_stance_.at(LF) << x_nominal_b, y_nominal_b, z_nominal_b;
    nominal_stance_.at(RF) << x_nominal_b, -y_nominal_b, z_nominal_b;
    nominal_stance_.at(LH) << -x_nominal_b, y_nominal_b, z_nominal_b;
    nominal_stance_.at(RH) << -x_nominal_b, -y_nominal_b, z_nominal_b;

    max_dev_from_nominal_ << 0.16, 0.09, 0.08;
  }
};

class Go1DynamicModel : public SingleRigidBodyDynamics {
public:
  Go1DynamicModel() : SingleRigidBodyDynamics(13,
                        0.1996, 0.451246, 0.409456, -0.000377827, -0.00371351, -0.00012186,
                        4) {}
```

```
};
```

Se definen dos clases llamadas *Go1KinematicModel* y *Go1DynamicModel* que representan el modelo cinemático y dinámico respectivamente.

La primera hereda de la clase base *KinematicModel* y en ella se definen las posiciones de los efectores finales de cada pata de la posición de reposo, la posición nominal de la base del robot y el tamaño del prisma de restricción del cual los efectores no pueden salir (4.2.2.1).

La segunda hereda de la clase *SingleRigidBodyDynamics* y se encarga de implementar la dinámica de un sólido rígido único (4.2.2.2). En esta clase se definen los parámetros físicos del robot, como la masa del robot, la matriz de inercias en la posición nominal y el número de patas del robot.

Para obtener dichos parámetros en la posición nominal se ha creado el siguiente *script*:

Código 5.2: Obtención de la masa y la matriz de inercia mediante Pinocchio

```
import pinocchio
import os
import numpy

script_dir = os.path.dirname(os.path.realpath(__file__))
urdf_filename = script_dir + "/go1."urdf

model = pinocchio.buildModelFromUrdf(urdf_filename, pinocchio.JointModelFreeFlyer())
data = model.createData()
print('model name: ' + model.name)

q = pinocchio.neutral(model)
q[7:] = numpy.array([0.0, 0.67, −1.3, −0.0, 0.67, −1.3, 0.0, 0.67, −1.3, −0.0, 0.67, −1.3])
v = pinocchio.utils.zero(model.nv)
pinocchio.ccrba(model, data, q, v)
print(data.Ig)

print(pinocchio.centerOfMass(model, data, q))
```

Este *script* utiliza la biblioteca *Pinocchio* para obtener el centro de masas, la matriz de inercia y la masa del robot.

En primer lugar, mediante *Pinocchio*, se crea un modelo del robot Go1 a partir de su archivo URDF. Luego, se obtienen los datos del robot utilizando la instancia **pinocchio::Data** y finalmente, se aplica el algoritmo Centroidal Composite Rigid Body Algorithm (CCRBA) para calcular la masa y la matriz de inercia del robot en función de la posición del centro de masas en la posición nominal. Obteniendo:

$$m = 13kg \tag{5.1a}$$

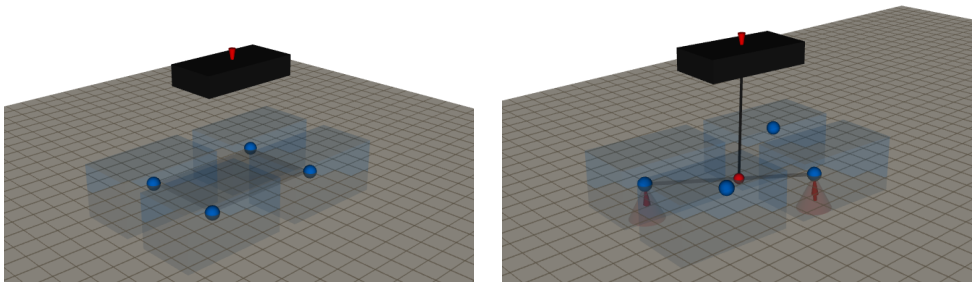
$$I = \begin{bmatrix} 0.15579 & -0.000426763 & -0.0163273 \\ -0.000426763 & 0.420183 & -0.000099 \\ -0.0163273 & -0.000099 & 0.422203 \end{bmatrix} \tag{5.1b}$$

El siguiente paso es incluir el robot en el archivo **robot_model.h** ubicado en **towr/towr/include/towr/model**. En dicho archivo se define la clase base *RobotModel*, que representa el modelo cinemático y dinámico de un robot en el contexto del problema abordado por *towr*. Esta clase se utiliza como una interfaz para definir y acceder a los modelos específicos de

cada robot. Para incluir al robot, es necesario introducirlo en el enumerador *Robot*, el cual enumera los diferentes robots para los cuales se implementan los modelos, y en el mapa llamado *robot_names*, que mapea los enumerados de *Robot* a sus nombres correspondientes.

Finalmente, para terminar de incluir los modelos, es necesario modificar el fichero **robot_model.cc** ubicado en **towr/towr/src/**. Este *script* implementa el constructor de la clase *RobotModel* mencionada anteriormente. Es responsable de crear y asignar los modelos cinemáticos y dinámicos adecuados al objeto *RobotModel* en función del robot especificado. Dentro del constructor, se utiliza un *switch* para determinar el robot específico y luego crear la instancia apropiada con los modelos correspondientes. Por ello, se debe incluir el archivo **go1_model.h** e incluir al robot Go1 en el *switch* asignándole sus modelos: *Go1DynamicModel* y *Go1KinematicModel*.

De esta manera, los modelos del robot Go1 se incluyen en *towr* y este es capaz de generar las trayectorias cartesianas en función de su dinámica y cinemática.



(a) Representación en reposo **(b)** Representación en movimiento

Figura 5.1: Representación simple del Go1 generado en Rviz por *towr*

En las imágenes, se pueden observar las esferas azules que representan los extremos de las patas del robot, los prismas azules que son las áreas que delimitan los efectores finales y el prisma negro que representa la base del robot.

5.1.2 Visualización y cinemática inversa del robot

Como se puede observar en la Figura 5.1, al incluir los modelos, *towr* genera una representación básica del robot Go1 en función de las especificaciones y restricciones. Sin embargo, para poder realizar el seguimiento de las trayectorias más adelante, es necesario realizar la cinemática inversa del robot para obtener las posiciones de cada articulación en cada instante de tiempo, y visualizar y comprobar si se realizan los movimientos correctamente.

El primer paso, es hacer uso del paquete del robot Go1 desarrollado por Unitree Robotics llamado *go1_description*. Dicho paquete contiene el fichero URDF, los ficheros xacro, las mallas del robot, archivos *launch* para visualizarlo en *Rviz* y un archivo *yaml* para utilizarlo en su controlador virtual.

El paquete se coloca en la ubicación **xpp/robots** dentro del paquete *xpp*. Para poder visualizar el robot mediante su URDF y realizar la cinemática inversa se deben crear cinco archivos:

El primero se denomina **inverse_kinematics_go1.h** el cual se debe ubicar dentro del

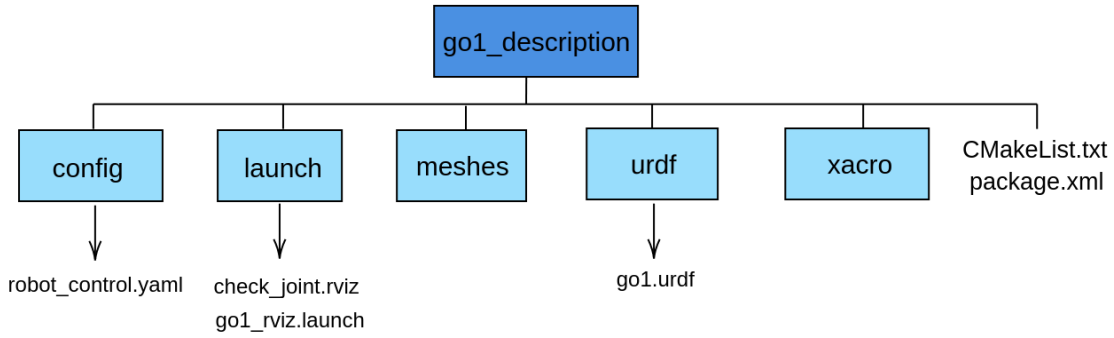


Figura 5.2: Jerarquía del paquete *go1_description*

paquete del robot en **go1_description/include/go1_description**. Este código define la implementación de la cinemática inversa del robot Go1. En él, se crea una clase llamada *InverseKinematicsGo1* que hereda de una clase base *InverseKinematics*. En dicha clase se definen dos funciones principales.

La función llamada *GetAllJointAngles* que toma como parámetro la posición tridimensional de un pie expresada en el marco de referencia base y devuelve los ángulos de las articulaciones necesarios para alcanzar dicha posición. Y la otra función, denominada *GetEECount*, que devuelve el número de efectores finales del robot, en este caso, cuatro.

Además, se incluyen variables que representan las distancias desde la base del robot hasta las articulaciones de las caderas y un objeto *Go1legInverseKinematics* llamado *leg* para implementar la cinemática inversa específica para una pata.

Código 5.3: Definición de la clase InverseKinematicsGo1

```
class InverseKinematicsGo1 : public InverseKinematics {
public:
  InverseKinematicsGo1() = default;
  virtual ~InverseKinematicsGo1() = default;

  Joints GetAllJointAngles(const EndeffectorsPos& pos_b) const override;

  int GetEECount() const override { return 4; };

private:
  Vector3d base2hip_LF_ = Vector3d(0.18, 0.125, 0.0);
  Vector3d base2hip_LH_ = Vector3d(-0.18, 0.125, 0.0);
  Go1legInverseKinematics leg;
};
```

El siguiente código se ubica en **go1_description/src** y se llama **inverse_kinematics_go1.cc**. Este *script* implementa las función *GetAllJointAngles* definida de la clase anterior. La función toma como parámetro las posiciones tridimensionales de los efectores finales del robot. Dentro de la función, se declaran algunas variables locales, como *ee_pos_H* que representa la posición del efector final en el marco de referencia de la cadera, y *q_vec* que es un vector que almacena los ángulos de las articulaciones para cada efector final.

Tras ello, se itera sobre cada efector final y, según cual sea, se realizan los cálculos correspondientes para trasformar las posiciones del marco de la base al de la cadera. Por ejemplo,

en el caso de la pata delantera izquierda, se resta la posición del efector final en el marco de referencia base a la posición de la cadera izquierda (*base2hip_LF_)* y luego, se calculan los ángulos de las articulaciones correspondientes utilizando la función *GetJointAngles* (función que se explica más adelante) del objeto *leg* cuyo resultado se almacenan en el vector *q_vec*.

Finalmente, se devuelve un objeto de tipo *Joints* que encapsula los vectores de ángulos de las articulaciones de todas las patas.

Como se puede observar en el código, para calcular la cinemática inversa de un robot cuadrúpedo, solo es necesario definir la inversa de una pata. Esto se logra extrayendo las posiciones de los efectores finales y extrapolándolas desde el marco de referencia base al marco de referencia de cada cadera, tal y como se explica en el apartado 4.3.2.

Código 5.4: Transformaciones de la base a la cadera para cada pata

```
for (int ee=0; ee<pos_B.size(); ++ee) {
    using namespace quad;
    ee_pos_H = pos_B.at(ee);
    switch (ee) {
      case LF:
        ee_pos_H -= base2hip_LF_;
        break;
      case RF:
        ee_pos_H = pos_B.at(ee).cwiseProduct(Eigen::Vector3d(1,-1,1));
          ee_pos_H -= base2hip_LF_;
        break;
      case LH:
        ee_pos_H -= base2hip_LH_;
        break;
      case RH:
        ee_pos_H = pos_B.at(ee).cwiseProduct(Eigen::Vector3d(1,-1,1));
          ee_pos_H -= base2hip_LH_;
        break;
      default: // joint angles for this foot do not exist
        break;
    }
```

A continuación, se crea un nuevo archivo en **go1_description/include/go1_description** llamado **go1leg_inverse_kinematics.h**. En dicho archivo se define la clase llamada *Go1legInverseKinematics* que se utiliza para transformar las posiciones cartesianas de una pata del robot a los ángulos de cada articulación.

Para poder realizar su función se define un enumerador denominado *Go1JointID* que define un identificador para cada articulación de la pata. Siendo los identificadores: HAA (articulación del muslo con respecto a la cadera), HFE (articulación de la tobillo con respecto al muslo) y KFE (articulación del pie con respecto del tobillo).

Después, se define la función *GetJointAngles* la cual toma una posición tridimensional de la pata expresada en el marco de referencia de la cadera y devuelve los ángulos de las articulaciones necesarios para alcanzar esa posición.

También se define la función *EnforceLimits* que restringe los ángulos a una posición factible si se sobrepasan los límites especificados. Como parámetros de entrada recibe el ángulo de la articulación y el identificador de la articulación correspondiente.

Finalmente, se definen algunas variables que representan distancias y longitudes para realizar la inversa de la pata del robot, como la distancia entre las articulaciones del muslo y la cadera en la dirección Z y las longitudes de las dos partes que conforman la pata.

Código 5.5: Definición de la clase *Go1legInverseKinematics*

```
enum Go1JointID {HAA=0, HFE, KFE, Go1legJointCount};
class Go1legInverseKinematics {
public:
  using Vector3d = Eigen::Vector3d;

  Go1legInverseKinematics () = default;
  virtual ~Go1legInverseKinematics () = default;

  Vector3d GetJointAngles(const Vector3d& ee_pos_H) const;

  void EnforceLimits(double& q, Go1JointID joint) const;

private:
  Vector3d hfe_to_haa_z = Vector3d(0.0, 0.0, 0.048);
  double length_thigh = 0.190; // length of upper leg
  double length_shank = 0.198; // length of lower leg
};
```

Tras definir la clase anterior es necesario crear un archivo que implemente ambas funciones, dicho *script* se crea en la ubicación **go1_description/src** y se le da el nombre de **go1leg_inverse_kinematics.cc**. Pese a que el método se explica en el apartado 4.3.2, a continuación, se explica como se ha llevado dicho método a código paso a paso. En primer lugar se explica la función *GetJointAngles*:

1. Se crear las variables *q_HAA*, *q_HFE* y *q_KFE* para almacenar los ángulos de las articulaciones de la cadera, muslo y tobillo respectivamente.

2. A continuación, se crea un vector tipo *Eigen::Vector3d* llamado *xr* que almacena la posición de la pata en el sistema de referencia de la cadera.

3. Tras esto, es posible calcular el ángulo de la primera articulación (*q_HAA*) mediante 4.3.

4. El siguiente paso es crear una matriz de rotación R que se utiliza para rotar el vector *xr* al sistema de coordenadas del muslo (giro en X del ángulo calculado anteriormente).

5. Después, se realiza una traslación del vector al sistema de coordenadas del muslo añadiendo la distancia *hfe_to_haa_z* que representa la distancia entre las articulaciones del muslo y la cadera en la dirección Z. Tras realizar esto, la posición cartesiana de la pata se encuentra en función del sistema de coordenadas del muslo.

6. A continuación, se realiza el calculo de la distancia del muslo al pie mediante el teorema de Pitágoras y se almacena en *tmp*.

7. Se calcula el ángulo *alpha* restando *PI/2* al cálculo de la arco-tangente de las coordenadas -Z y X del vector *xr* (4.4a). Esta diferencia se hace para realizar un ajuste y coincidir con la definición de las articulaciones del robot Go1. Si no se realizara, el robot queda como en la Figura 5.3.

8. Se definen las variables *ll* y *lu* que determinan la longitud de la parte superior e inferior de las patas respectivamente.

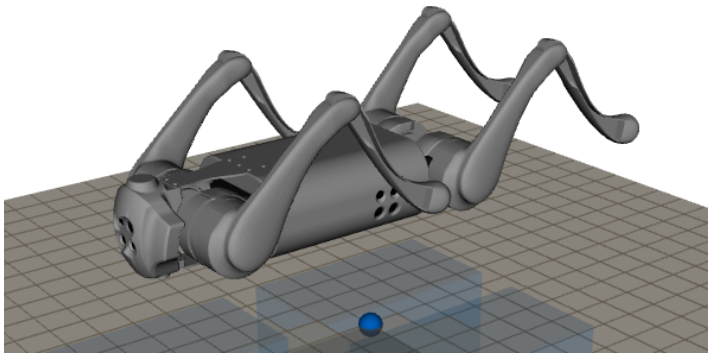


Figura 5.3: Visualización del Go1 sin restar PI/2 al calcular *alpha*

9. Se calcula un valor temporal *some_random_value_for_beta*. Dicho valor representa el coseno del ángulo beta, es decir, la ecuación 4.4b sin realizar el arco-coseno, y se realiza una comprobación de que esta dentro del rango [-1,1]. Si se sobrepasa del rango, el valor se limita a -1 o 1, según el caso.

10. Se calcula *beta* aplicando el arco-coseno al valor anterior obtenido.

11. De esta forma, es posible obtener el ángulo de la segunda articulación (*q_HFE*) mediante la suma de *alpha* y *beta* (4.4c).

12. Para calcular el último ángulo, es necesario calcular otro valor temporal *some_random_value_for_gam*
Al igual que la variable temporal anterior, esta representa el coseno pero del ángulo *gamma*. Tras calcularlo, se comprueba que este dentro del rango.

13. Se obtiene el valor del ángulo *gamma* tras aplicar el arco-coseno y se obtiene el ángulo de la última articulación (*q_KFE*) restando PI a *gamma*. Esta diferencia se hace para tener en cuenta la flexión del tobillo. Si no se realizara, el robot queda de la siguiente forma:

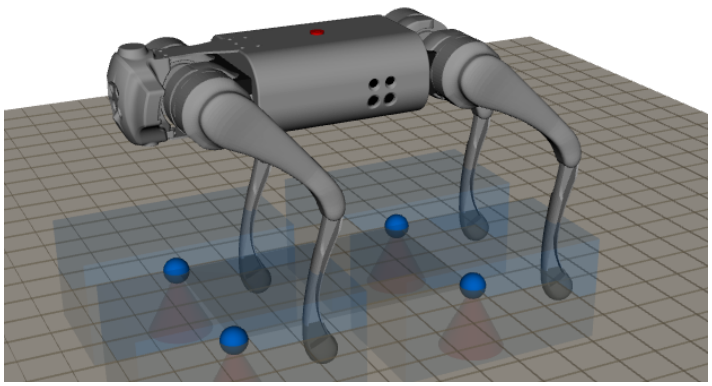


Figura 5.4: Visualización del Go1 sin restar PI a *gamma*

14. Antes de devolver los ángulos de las articulaciones se les aplican restricciones utilizando la función *EnforceLimits*. De esta manera, es posible asegurarse de que estén dentro del rango permitido.

15. Finalmente, se devuelve un vector que contiene los ángulos de las articulaciones calculados: *q_HAA*, *q_HFE* y *q_KFE*.

Como se ha comentado anteriormente, para asegurar que las articulaciones se encuentren dentro del rango permitido se utiliza la segunda función de la clase *Go1legInverseKinematics* llamada *EnforceLimits*. Esta función define los límites máximos y mínimos de cada articulación. Después, utiliza dos mapas, uno que asocia cada articulación con su valor máximo permitido en radianes y otro que realiza la misma función pero para los valores mínimos.

Finalmente, se compara el valor actual de la articulación especificada con el valor máximo y el valor mínimo. Si el valor actual es superior al máximo, se limita y se actualiza el valor a dicho máximo, sino, se mantiene. Si el valor actual es inferior al mínimo, se limita y se actualiza el valor a dicho mínimo, y sino, al igual que antes, se mantiene.

Código 5.6: Función que limita los ángulos de las articulaciones

```
void Go1legInverseKinematics::EnforceLimits (double& val, Go1JointID joint) const {
  const static double haa_min = -50;
  const static double haa_max = 50;
  const static double hfe_min = -40;
  const static double hfe_max = 258;
  const static double kfe_min = -161.5;
  const static double kfe_max = -50;

  static const std::map<Go1JointID, double> max_range {
    {HAA, haa_max/180.0*M_PI},
    {HFE, hfe_max/180.0*M_PI},
    {KFE, kfe_max/180.0*M_PI}
  };
  static const std::map<Go1JointID, double> min_range {
    {HAA, haa_min/180.0*M_PI},
    {HFE, hfe_min/180.0*M_PI},
    {KFE, kfe_min/180.0*M_PI}
  };

  double max = max_range.at(joint);
  val = val>max? max : val;
  double min = min_range.at(joint);
  val = val<min? min : val;
}
```

Tras explicar como realizar la cinemática inversa para que el robot generado en Rviz pueda seguir la trayectoria cartesiana de *towr*, es necesario crear un último archivo. Este *script* llamado **urdf_visualizer_go1.cc** está ubicado en **go1_description/src/exe**. Se trata de un nodo de ROS que crea los objetos de las clases *InverseKinematicsGo1*, *CartesianJointConverter* y *UrdfVisualizer* para permitir la visualización en RViz del modelo URDF a partir de los estados de las articulaciones deseadas del robot.

En primer lugar se define el nombre del *topic* donde se publican los valores de las articulaciones. Tras ello, se crea un objeto *InverseKinematicsGo1*. Mediante este objeto, el *topic* de las posiciones cartesianas y el *topic* de los estados de las articulaciones, se crea un objeto *CartesianJointConverter*.

A continuación, se almacena en un vector los nombres de las articulaciones tal y como se definen en el URDF del robot. Finalmente, se crea un objeto *UrdfVisualizer* que toma el nombre del archivo URDF, el vector con los nombres de las articulaciones, el nombre de la articulación base en el URDF, el marco fijo en RViz, el *topic* de los estados de las articulaciones y el prefijo para los marcos de transformación.

Al crear los anteriores objetos, en sus constructores, se forman diferentes *publishers* y *subscribers*. Cuando se publica un mensaje en el *topic* de las posiciones cartesianas el archivo **cartesian_joint_converter.cc** lo recibe y realiza la inversa mediante el objeto *InverseKinematicsGo1*. Una vez realizada la inversa utilizando los métodos anteriores, se publican los ángulos de las articulaciones en su *topic* correspondiente. Este mensaje lo recibe el archivo **urdf_visualizer.cc** que asigna cada valor a su articulación correspondiente y publica en el *robot_state_publisher* permitiendo visualizar el movimiento del robot.

Después de haber creado todos los archivos mencionados, el paquete *go1_descrition* debe acabar teniendo la consecuente jerarquía:

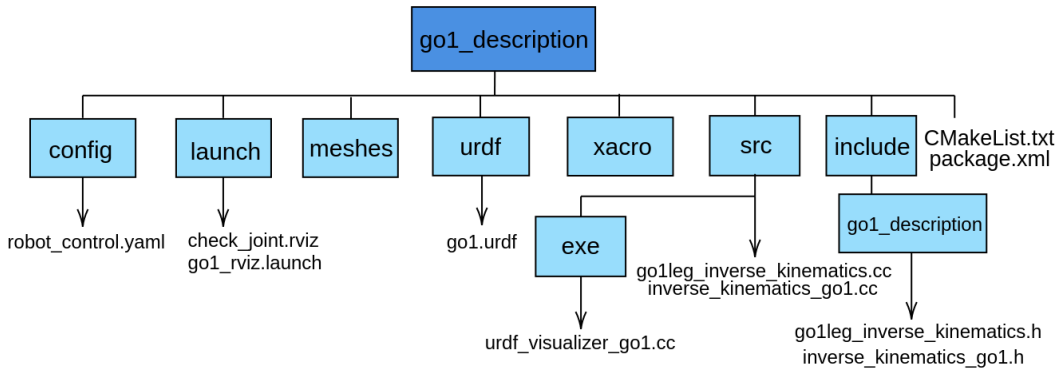


Figura 5.5: Jerarquía final del paquete *go1_description*

Sin embargo, para poder compilar los archivos es necesario modificar el fichero **CMakeList.txt** del paquete añadiendo los archivos **go1leg_inverse_kinematics.cc** y **inverse_kinematics_go1.cc** como librerías y el archivo **urdf_visualizer_go1** como un ejecutable.

Código 5.7: Cambios en el CMakeList.txt del paquete del Go1 en towr

```
add_library(${PROJECT_NAME}
  src/go1leg_inverse_kinematics.cc
  src/inverse_kinematics_go1.cc
)
add_executable(urdf_visualizer_go1 src/exe/urdf_visualizer_go1.cc)
target_link_libraries(urdf_visualizer_go1
  ${PROJECT_NAME}
  ${catkin_LIBRARIES}
)
```

Finalmente, se deben modificar los archivos **go1_rviz.launch** del paquete del Go1 y **towr_ros.launch** ubicado en **towr/towr_ros/launch** para realizar las llamadas corres-

pondientes y así poder visualizar el robot en Rviz.

En el caso del primer archivo, es necesario eliminar la llamada a los nodos *joint_state_publisher* y *robot_state_publisher* y sustituirlos por una llamada al nodo *urdf_visualizer_go1*.

Código 5.8: Cambios en el go1_rviz.launch

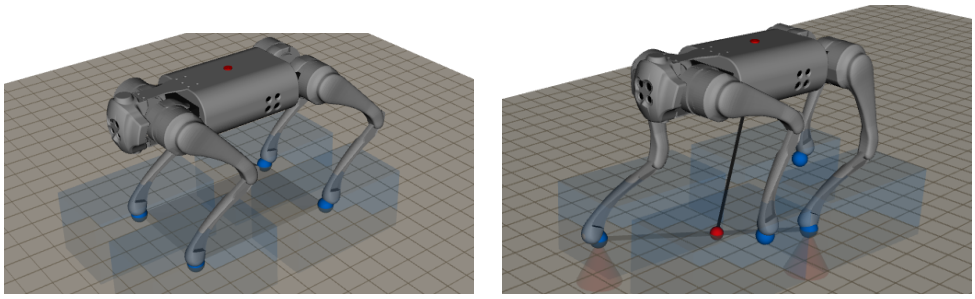
```
<launch>
    <arg name="user_debug" default="false"/>
    <param name="go1_description" command="$(find xacro)/xacro --inorder '$(find go1_description)/xacro/robot.xacro' DEBUG:=$(arg user_debug)"/>
    <node name="urdf_visualizer_go1" pkg="go1_description" type="urdf_visualizer_go1" output="screen" />
</launch>
```

En el caso del segundo, se debe incluir una llamada al fichero *launch* anterior.

Código 5.9: Incluir llamada al archivo go1_rviz.launch

```
<include file="$(find go1_description)/launch/go1_rviz.launch"></include>
```

De esta manera, al lanzar *towr* y añadir un nuevo *RobotModel* con sus parámetros definidos *Robot Description* y *TF Prefix* permite la visualización del robot Go1.



(a) Visualización en reposo **(b)** Visualización en movimiento

Figura 5.6: Visualización del Go1 en Rviz

5.1.3 Modificación de la gravedad

towr, por defecto, planifica las trayectorias utilizando la gravedad terrestre. Sin embargo, para cumplir los objetivos de este proyecto se deben realizar modificaciones en algunos archivos para que genere trayectorias con una gravedad diferente.

Para ello, es necesario modificar dos *scripts*. El primero se llama *dynamic_model.cc* ubicado en la ruta **towr/towr/src**. Este archivo contiene la implementación del constructor de la clase *DynamicModel*, donde se establecen las variables de gravedad y masa que se utilizan para calcular el modelo dinámico. Por tanto, para generar las trayectorias en una gravedad diferente, se debe modificar la variable que constituye la aceleración gravitatoria, estableciendo el valor deseado.

EL segundo fichero que hay que modificar se denomina **rviz_robot_builder.cc** ubicado en la dirección **xpp/xpp_vis/src**. Este archivo es el encargado de crear y generar la representación simple del robot Go1 en Rviz (5.1). En dicho archivo existe una variable a la que

se asigna el valor de la gravedad y se utiliza para crear una flecha que representa la fuerza de la gravedad ejercida sobre el centro de masas del robot. Para que la flecha se visualice correctamente en Rviz, es necesario modificar el parámetro al mismo valor que se ha asignado en el primer archivo.

Al realizar estas modificaciones en ambos archivos, es posible generar trayectorias en diferentes gravedades y asegurar que la visualización en Rviz sea coherente con los cambios realizados.

5.1.4 Modificación de la frecuencia de escritura en el rosbag

La frecuencia de publicación en el *rosbag* es un factor crucial a tener en cuenta, ya que determina la cantidad de datos capturados y almacenados para su posterior reproducción. Al incrementar la frecuencia de publicación, se logra una mayor cantidad de puntos de datos en el tiempo, lo que a su vez permite una representación más precisa y suave de la trayectoria seguida por el robot.

Al generar trayectorias más suaves, se busca mejorar la calidad del movimiento del robot Go1 y obtener los valores a una frecuencia compatible con la frecuencia de funcionamiento del robot (500Hz).

Este cambio de se realiza en el archivo **towr_ros_interface.cc** ubicado en **towr/towr_ros/src** donde se debe cambiar la variable *visualization_dt_* para que guarde los valores a la frecuencia correcta, aproximadamente cada 0.002 segundos.

5.2 Seguimiento de la trayectoria

Una vez que comprendido el proceso de como incluir lo necesario para generar trayectorias utilizando *towr*, es importante entender cómo se pueden seguir esas trayectorias tanto en simulación como en el robot real. En esta sección, se explora el proceso para lograr un seguimiento preciso de las trayectorias generadas.

En la simulación, se van ha utilizar herramientas y entornos virtuales para verificar el comportamiento del robot y evaluar su desempeño en diferentes movimientos. Una vez que hayamos validado y refinado las trayectorias en simulación, el siguiente paso es implementar el seguimiento de trayectorias en el robot real. Esto implica traducir las trayectorias generadas en comandos y acciones ejecutables por los actuadores del robot.

Para llevar a cabo un seguimiento preciso de las trayectorias, es necesario contar con los paquetes proporcionados por *Unitree Robotics*. Estos paquetes son esenciales para controlar y comunicarse eficazmente con sus robots cuadrúpedos. El primer paso para utilizar sus paquetes es descargarlos e instalarlos desde su repositorio oficial *Unitree Robotics github* (s.f.).

Sus paquetes constan de varios repartidos en tres carpetas principales:

- **unitree_ros**: esta carpeta contiene los paquetes (controlador virtual, mundos, robots…) que se utilizan para realizar la simulación de los robots en *Gazebo*.

- **unitree_ros_to_real** y **unitree_legged_sdk**: contiene las herramientas y dependencias necesarias para enviar los comandos al robot por UDP. Concretamente en la

carpeta **unitree_ros_to_real** contiene el paquete que define los mensajes que se utilizan para enviar los comandos.

5.2.1 Simulación

Tal y como se indica en el apartado 4.6, se ha desarrollado un archivo con el objetivo de leer y utilizar los valores almacenados en el *rosbag* generado por *towr*. El *rosbag* captura y almacena la información, que incluye datos cruciales como la posición, velocidad y aceleración de las articulaciones del robot a lo largo del tiempo. Estos valores son fundamentales para comprender y replicar las trayectorias. La figura 4.9 muestra un ejemplo visual de los datos contenidos en el *rosbag*.

A continuación se realiza la explicación del contenido de dicho *script* y su funcionamiento:

1. El primer paso es crear el archivo al que se le ha denominado como **bag_reader.cc** y se ha ubicado en la dirección **unitree_ros/unitree_controller/src**.

2. Para que el código funcione correctamente es necesario incluir las librerías necesarias, por ello se incluyen las librerías para trabajar con *Pinocchio*, las librerías de ROS y otras librerías estándar de C++. También se incluyen los tipos de mensaje definidos en los paquetes que se van a utilizar.

3. Tras ello se definen dos funciones auxiliares: *EnforceLimits* y *inverse_kinematics*, que son las mismas funciones del archivo **go1leg_inverse_kinematics.cc** explicadas en el apartado 5.1.2.

4. A continuación se define la función principal, la función *main*, que es el punto de entrada del programa y donde se realiza la mayor parte del trabajo.

5. El primer paso de la función *main* es inicializar ROS y darle un nombre al nodo. Después, se crea un objeto *ros::NodeHandle* llamado *n*. Un *ros::NodeHandle* es un objeto que permite interactuar con el sistema de comunicación de ROS y proporciona métodos para publicar mensajes, suscribirse a tópicos y realizar otras operaciones. Además, se establece la frecuencia de publicación a 500Hz.

6. Se crean los objeto *ros::Publisher* para publicar mensajes de tipo *unitree_legged_msgs::MotorCmd* para cada *topic* asociado a su articulación. Cada objeto se asigna a un elemento del vector llamado *servo_pub*.

Código 5.10: Creación de los Publishers

```
ros::Publisher servo_pub[12];

servo_pub[0] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/FL_hip_controller/↩
    ↪ command", 1);
servo_pub[1] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/FL_thigh_controller/↩
    ↪ command", 1);
servo_pub[2] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/FL_calf_controller/↩
    ↪ command", 1);
servo_pub[3] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/FR_hip_controller/↩
    ↪ command", 1);
servo_pub[4] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/FR_thigh_controller/↩
    ↪ command", 1);
```

```
servo_pub[5] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/FR_calf_controller/command", 1);
servo_pub[6] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/RL_hip_controller/command", 1);
servo_pub[7] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/RL_thigh_controller/command", 1);
servo_pub[8] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/RL_calf_controller/command", 1);
servo_pub[9] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/RR_hip_controller/command", 1);
servo_pub[10] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/RR_thigh_controller/command", 1);
servo_pub[11] = n.advertise<unitree_legged_msgs::MotorCmd>("/go1_gazebo/RR_calf_controller/command", 1);
```

7. Debido a que el *rosbag* contiene los valores de fuerza de cada pata en cartesiano, es necesario transformar dichos los valores a articulares. Para transformar los valores, se utiliza *Pinocchio*. En primer lugar, se obtiene el modelo completo del robot utilizando su archivo URDF.

   A continuación, se obtienen modelos reducidos para cada pata. Esto implica bloquear las articulaciones y segmentos no necesarios para el cálculo de la cinemática y dinámica de cada pata. Al tener modelos reducidos, se simplifica el cálculo de las jacobianas y se reduce la complejidad computacional.

   Para bloquear las articulaciones se crea un vector que contiene los nombres de las articulaciones que no se van a utilizar. Tras ello, se obtienen los IDs que asocian esas articulaciones con las del modelo completo para poder crea su modelo reducido y obtener sus datos.

   Código 5.11: Modelo reducido de la pata izquierda

```
pinocchio::Model model_complete;
pinocchio::urdf::buildModel(urdf_filename,model_complete);

std::vector<std::string> articulaciones_bloqueadas_FL_name{"FR_hip_joint", "FR_thigh_joint", "FR_calf_joint", "RL_hip_joint", "RL_thigh_joint", "RL_calf_joint", "RR_hip_joint", "RR_thigh_joint", "RR_calf_joint"};

std::vector<pinocchio::JointIndex> articulaciones_bloqueadas_FL_id;
articulaciones_bloqueadas_FL_id.reserve(articulaciones_bloqueadas_FL_name.size());
for (const auto& str : articulaciones_bloqueadas_FL_name){
    articulaciones_bloqueadas_FL_id.emplace_back(model_complete.getJointId(str));
}

auto qc = pinocchio::neutral(model_complete);
pinocchio::Model model_FL = pinocchio::buildReducedModel(model_complete,articulaciones_bloqueadas_FL_id, qc);

pinocchio::Data data_FL(model_FL);
```

8. El siguiente paso es crear una matriz jacobiana para cada modelo reducido y crear las variables que van a ir conteniendo los valores de posición y par. Además, se definen otras variables para realizar la cinemática inversa, concretamente las generadas en los archivos **inverse_kinematics_go1.h** y **go1leg_inverse_kinematics.h** explicados anteriormente, que son: distancias tridimensionales de la base a cada cadera, distancia de la cadera al muslo en la dirección Z y longitud de las dos partes de la pata.

9. Cuando el robot se genera en Gazebo, inicialmente aparece totalmente tumbado. Sin embargo, las trayectorias generadas para el robot comienzan desde una posición inicial estando ya levantado. Para lograr una transición suave entre la posición tumbada y la posición levantada, es necesario realizar una interpolación de valores.

   La interpolación implica calcular valores intermedios entre dichas posiciones. Al realizar esta interpolación, se logra una transición gradual y suave desde la posición tumbada a una posición levantada, evitando movimientos bruscos y permitiendo un inicio más natural de la trayectoria.

10. A continuación, se crea un objeto tipo *rosbag::Bag* que se utiliza para abrir el archivo *rosbag* deseado. Mediante la biblioteca *rosbag::View* se determinan el *topic* sobre cuyos mensajes se quiere iterar.

11. Para cada iteración se comprueba si el mensaje es del tipo deseado (*xpp_msgs::RobotStateCartesian*). En la primera iteración del bucle, se realiza una segunda interpolación entre la posición levantada anterior y la primera posición de la trayectoria planificada. Una vez que el robot ha alcanzado la posición inicial, la trayectoria planificada se sigue normalmente en las iteraciones sucesivas del bucle de la siguiente forma:

12. Como el *rosbag* guarda los valores de posición de la base y de las patas referenciados al mundo en cartesiano es necesario resta las coordenadas de la base a las de la pata para obtener las posiciones de los efectores finales en función de la base.

    Código 5.12: Obtención de las posiciones de la pata izquierda en función de la base

```
q_FL_des[0] = force->ee_motion[0].pos.x - force->base.pose.position.x;
q_FL_des[1] = force->ee_motion[0].pos.y - force->base.pose.position.y;
q_FL_des[2] = force->ee_motion[0].pos.z - force->base.pose.position.z;
```

13. Para poder realizar la cinemática inversa utilizando las funciones auxiliares, es necesario obtener las posiciones en función de la cadera. Estas posiciones se obtienen restando las distancias que hay de la base a cada cadera a las posiciones de cada efector final.

14. Una vez obtenidas las posiciones articulares para cada pata, se calculan las jacobianas en función de cada modelo reducido, los datos y las posiciones articulares de la iteración actual utilizando *Pinocchio*.

15. La multiplicación de las jacobianas traspuestas y las fuerzas cartesianas dan lugar a los pares articulares.

16. Después de realizar el cálculo de las posiciones y pares articulares, estos valores se asignan al mensaje correspondiente, se indican las ganancias proporcional y derivativa y se publican. Al publicar el mensaje, se logra el movimiento de las articulaciones de las patas permitiendo seguir las trayectorias.

17. En la última iteración del bucle, se realiza una última interpolación entre la última posición de la trayectoria y la posición tumbada inicial.

18. En cada iteración se pausa el programa durante un tiempo determinado utilizando *rate.sleep()* para controlar la frecuencia de publicación a 500Hz. Finalmente, cuando acaban los mensajes del *rosbag*, se cierra y se termina de ejecutar el nodo.

Para poder ejecutar el nodo, se envíen los mensajes y se siga la trayectoria en necesario incluir el archivo como un ejecutable en el *CMakeList.txt* del paquete **unitree_controller**. Además es necesario incluir las bibliotecas de *rosbag* y *Pinocchio*.

Código 5.13: Cambios en el CMakeList.txt del paquete unitree_controller

```
find_package(catkin REQUIRED COMPONENTS
    ...
    rosbag
)
find_package(pinocchio REQUIRED)
include_directories(
    include
    ...
    ${Boost_INCLUDE_DIR}
    ${Pinocchio_INCLUDE_DIRS}
)
add_executable(bag_reader src/bag_reader.cpp)
target_link_libraries(bag_reader ${catkin_LIBRARIES} pinocchio::pinocchio)
```

5.2.2 Robot real

Una vez que se han generado y optimizado las trayectorias en la simulación, el siguiente paso es probarlas en el robot real. Para comenzar, es necesario establecer una conexión con el robot para poder enviar los comandos de control. Esta conexión se realiza a través de *Ethernet* mediante el protocolo UDP. Para conectarse al robot se debe ejecutar el siguiente archivo *.sh* proporcionado por *Unitree Robotics*:

Código 5.14: Archivo .sh para la conexión con el robot

```
sudo ifconfig lo multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev lo
sudo ifconfig enx000ec6612921 down
sudo ifconfig enx000ec6612921 up 192.168.123.161 netmask 255.255.255.0
```

1. En primer lugar se habilita el soporte para multidifusión en la interfaz de bucle local. La interfaz de bucle local es una interfaz de red virtual que permite la comunicación dentro del equipo local.

2. A continuación se agrega una ruta para el rango de direcciones de multidifusión a la interfaz de bucle local. Esto especifica que cualquier tráfico destinado a este rango de direcciones debe enviarse a través de la interfaz de bucle local.

3. Tras ello se desactiva la interfaz de red con el nombre especificado para la dirección actual.

4. Finalmente, se activa la interfaz de red asignando la dirección Internet Protocol (IP) 192.168.123.161 con una máscara de red de 255.255.255.0 lo que habilita la interfaz y permite la comunicación de red a través de ella.

Una vez establecida la conexión con el robot, el procedimiento para enviar los datos es similar al utilizado en la simulación, con la diferencia de que en lugar de enviar los valores de posición y pares a las articulaciones correspondientes directamente, se envían a

un *topic* específico llamado *low_cmd* mediante una variable que publica mensajes de tipo *unitree_legged_msgs::LowCmd*. Este tipo de mensaje contiene a los mensajes tipo *unitree_legged_msgs::MotorCmd* utilizado para enviar los comandos a las articulaciones. Para esto se crea un nuevo archivo llamado **go1_position.cpp** ubicado en **unitree_ros_to_real/unitree_legged_real/src/exe**.

El nodo **ros_udp.cpp** ubicado en **unitree_ros_to_real/unitree_legged_real/src/exe** es responsable de recoger los mensajes publicados en el tópico *low_cmd* y transformarlos para su posterior envío a través de UDP. En este proyecto, el enfoque es el control a bajo nivel del robot, por lo que se detalla el proceso específico en ese contexto.

1. Lo primero es importar las bibliotecas y mensajes necesarios.

2. A continuación crea una clase denominada como *Custom*. En ella se crean instancias de varias clases y se definen variables relacionadas con la seguridad y la comunicación UDP en el contexto del control a bajo nivel del robot.

   - **safe**: instancia de la clase *Safety*. Esta clase proporciona funcionalidades relacionadas con la seguridad del robot, como límites de posición y protección de energía. Es utilizada para garantizar que los comandos enviados al robot cumplan con ciertas restricciones de seguridad.
   - **low_udp**: instancia de la clase UDP para la comunicación a bajo nivel. Esta clase encapsula la lógica de envío y recepción de datos a través de dicho protocolo.
   - **low_cmd** y **low_state**: son variables de tipo *LowCmd* y *LowState*. Estas variables representan los comandos y estados de bajo nivel del robot. Los comandos de bajo nivel son utilizados para controlar directamente las articulaciones y actuadores del robot, mientras que los estados de bajo nivel contienen información detallada sobre las articulaciones y otros componentes del robot.

3. Además, la clase *Custom* encapsula la lógica relacionada con la comunicación y el envío y recepción de comandos y estados del robot. Para ello define ciertos métodos.

   - **Custom()**: Es el constructor de la clase. Aquí se inicializan las instancias de las clases *Safety*, *UDP* y se configuran los parámetros de comunicación para el robot específico (en este caso, *LeggedType::Go1*), las direcciones IP y puertos correspondientes.
   - **lowUdpSend()**: Este método se encarga de enviar los comandos de bajo nivel a través de la comunicación UDP. Primero, se utiliza el método *SetSend()* de la instancia *low_udp* para establecer los datos del comando *low_cmd*. Luego, se llama al método *Send()* para enviar el comando a través del protocolo.
   - **lowUdpRecv()**: Este otro método es el encargado de recibir los estados de bajo nivel a través de la comunicación. Primero, se llama al método *Recv()* de la instancia *low_udp* para recibir los datos. Luego, se utiliza el método *GetRecv()* para obtener los datos recibidos y almacenarlos en la variable *low_state*.

4. Tras definir la clase *Custom* se declaran varias variables y objetos de manera global que se utilizan en el nodo ROS para la comunicación con el robot.

- **Custom custom** : Se crea un objeto de la clase anterior que se utiliza para gestionar la comunicación y el envío/recepción de comandos y estados del robot.
- **ros::Subscriber sub_low** : Se declara un objeto *Subscriber* de ROS utilizada para suscribirse al *topic* de comandos de bajo nivel *low_cmd* y recibir los mensajes procedentes del nodo **go1_position.cpp**.
- **ros::Publisher pub_low** : objeto tipo *Publisher* que se utiliza para publicar los estados de bajo nivel del robot en el *topic low_state*.
- **long low_count** : variable para llevar un conteo de los mensajes de bajo nivel recibidos.

5. A continuación define la función *lowCmdCallback* que es el *callback* que se ejecuta cada vez que se recibe un mensaje en el *topic* de comandos de bajo nivel *low_cmd*.

   En primer lugar imprime el número de mensajes recibidos. Tras ello convierte el mensaje ROS recibido en un comando de bajo nivel *custom.low_cmd* utilizando la función *rosMsg2Cmd* para poder enviarlo al robot.

   Aplica límites de posición y protecciones de energía utilizando métodos del objeto *custom.safe*. FInalmente, convierte el estado de bajo nivel *custom.low_state* en un mensaje ROS utilizando la función *state2rosMsg* y se publica.

6. Como último paso se define la función *main* donde se configura la función principal del nodo. Se inicializa el nodo y se crea un objeto *ros::NodeHandle*. También se definen el *subscriber* y el *publisher* de los *topics* mencionados anteriormente.

   Finalmente, se crean dos objetos *LoopFunc* llamados *loop_udpSend* y *loop_udpRecv*. Estos objetos se encargan de ejecutar periódicamente las funciones *lowUdpSend* y *lowUdpRecv* respectivamente, lo que permite enviar y recibir los datos.

6 Resultados

Una vez finalizado el desarrollo práctico del proyecto y explicado en detalle todo el proceso de implementación, es el momento de llevar a cabo las pruebas y evaluar los resultados obtenidos. Las pruebas son una parte fundamental para verificar el funcionamiento y la eficacia del proceso realizado. Además, permiten confirmar que se cumplen con los objetivos establecidos y brindan información valiosa para futuras mejoras y optimizaciones.

En el proyecto actual, se lleva a cabo un análisis comparativo de la planificación de trayectorias utilizando *towr* en tres diferentes condiciones de gravedad: la gravedad de la Tierra, la gravedad de Marte y una gravedad intermedia. El objetivo de este estudio es observar cómo varía el movimiento del cuerpo del robot, así como las fuerzas cartesianas y posiciones articulares de las patas en cada una de estas condiciones gravitatorias.

Una vez obtenidas las trayectorias planificadas para cada condición de gravedad, se realiza la simulación de cada una de ellas en un entorno virtual. Como se ha comentado en apartados anteriores, se utiliza *Gazebo* para emular de manera realista las condiciones gravitatorias y las interacciones entre el robot y el entorno. Durante estas simulaciones, se registran y analizan datos relacionados con las aceleraciones del cuerpo del robot y las conversiones de las fuerzas y posiciones cartesianas a pares y posiciones articulares en todo momento.

El análisis comparativo de las trayectorias planificadas y simuladas en las diferentes condiciones de gravedad permiten evaluar cómo afecta la gravedad a la locomoción del robot, al comportamiento del cuerpo del robot y buscar posibles diferencias en la estabilidad, la postura y el equilibrio.

Para realizar dicha comparación se genera una trayectoria de cuatro pasos en 3.2 segundos hasta la posición (1.7, 0) para cada gravedad.

6.1 Resultados de la planificación

6.1.1 Gravedad Terrestre

En primer lugar se realizan las pruebas utilizando la gravedad terrestre. Dicha gravedad se elige como punto de referencia debido a su familiaridad, ya que es la gravedad que se experimenta en el entorno diario.

De esta forma, se establece una base sólida para comparar y evaluar el comportamiento del robot en las condiciones gravitatorias alternativas. Así, es posible comprender mejor las diferencias y similitudes en el desempeño del robot en los entornos gravitatorios distintos.

En primer lugar se muestra el movimiento de la base del robot a lo largo de la trayectoria planificada. Esto permite evaluar la precisión y suavidad del movimiento de la base, así como su capacidad para seguir la trayectoria deseada.

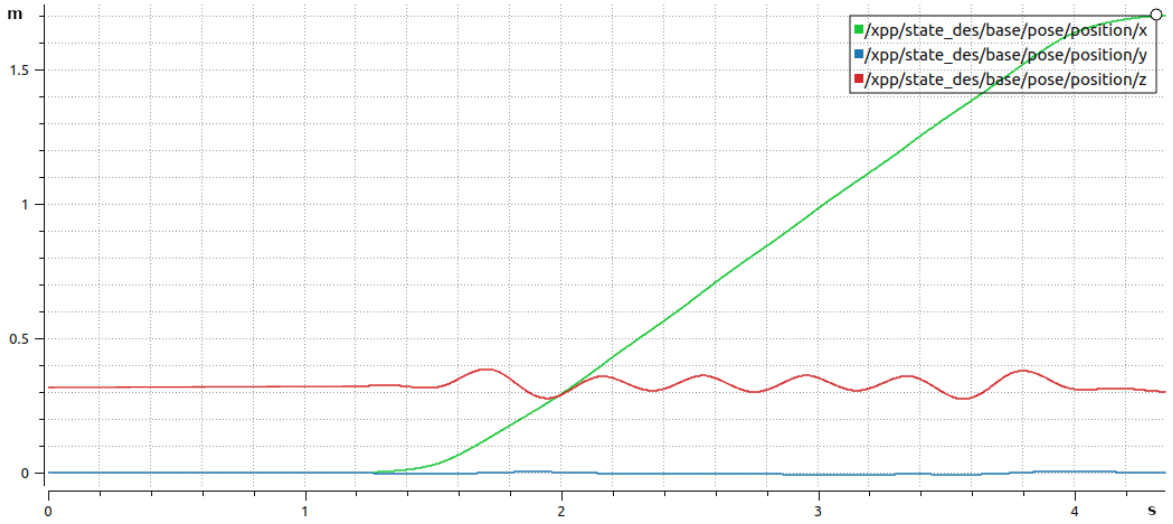


Figura 6.1: Visualización de la posiciones cartesianas de la base del robot respecto al mundo con gravedad terrestre en la planificación

En la gráfica, se puede observar claramente el comportamiento del desplazamiento. Dado que la trayectoria es una línea recta en el plano XY, la posición en el eje X del robot aumenta de manera continua a medida que avanza en el tiempo. Además, gracias al uso de *splines* por parte de *towr*, se puede observar una curva suave y continua del movimiento planificado.

En cuanto a la coordenada Y, se mantiene constante en cero, ya que no hay desplazamiento lateral. No obstante, es en el eje Z donde se observa una cierta oscilación. Esta oscilación en la posición Z se debe al movimiento del cuerpo del robot a lo largo de la trayectoria, ya que las patas, al realizar su movimiento ocasionan un movimiento vertical en el robot.

A continuación, se muestran las aceleraciones lineales y angulares que el robot experimenta. Estos datos proporcionan información crucial sobre la dinámica del movimiento del robot.

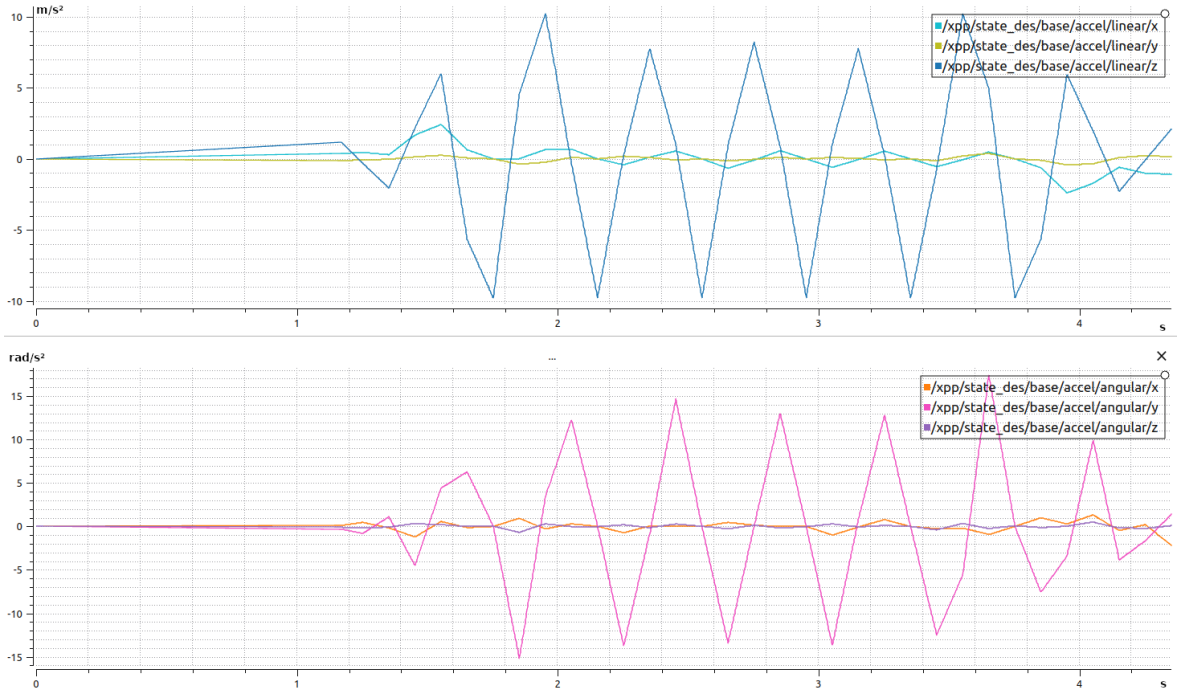


Figura 6.2: Visualización de las aceleraciones lineales y angulares tridimensionales del robot con gravedad terrestre en la planificación. En la parte superior se observa la aceleración lineal y en la inferior la angular.

Debido a las oscilaciones en la posición Z mencionadas anteriormente, podemos observar que el robot experimenta aceleraciones lineales en el mismo sentido. A medida que el cuerpo se mueve hacia arriba o hacia abajo en la trayectoria, se producen cambios en la aceleración lineal en el eje Z.

En relación al eje X, es posible observar tanto aceleraciones como deceleraciones lineales a lo largo de la trayectoria. Estas variaciones están directamente relacionadas con el avance del robot. Cuando el robot da un paso hacia adelante, experimenta una aceleración positiva para aumentar su velocidad y avanzar. Sin embargo, cuando el robot se apoya en una pata para dar el siguiente paso, disminuye su velocidad y, por lo tanto, experimenta una deceleración. Además, en el inicio y el final de la trayectoria se puede observar que dichas aceleraciones y deceleraciones son mayores, ya que al inicio, el robot pasa de estar quieto a en estar en movimiento y en el final ocurre lo contrario.

También se puede observar una interesante aceleración angular en Y. Esta aceleración es producto del balanceo natural que ocurre cuando el robot se desplaza hacia adelante. Cuando el robot da un paso, se genera una fuerza de impulso en la dirección opuesta al movimiento para propulsarlo y avanzar. Como resultado, el robot experimenta un momento angular en el eje Y, lo que provoca un balanceo alrededor de ese eje.

Este balanceo es una respuesta natural del movimiento del robot durante el desplazamiento. A medida que el robot avanza, se produce un patrón cíclico de balanceo, donde se inclina hacia adelante y luego se endereza al apoyar la siguiente pata para dar el siguiente paso.

A continuación, se presentan las fuerzas tridimensionales que actúan en cada una de las patas del robot a lo largo de la trayectoria. Esto proporciona información sobre la distribución de la carga entre las patas y además, se pueden identificar patrones y tendencias en las fuerzas que indican la eficiencia y el rendimiento de la caminata.

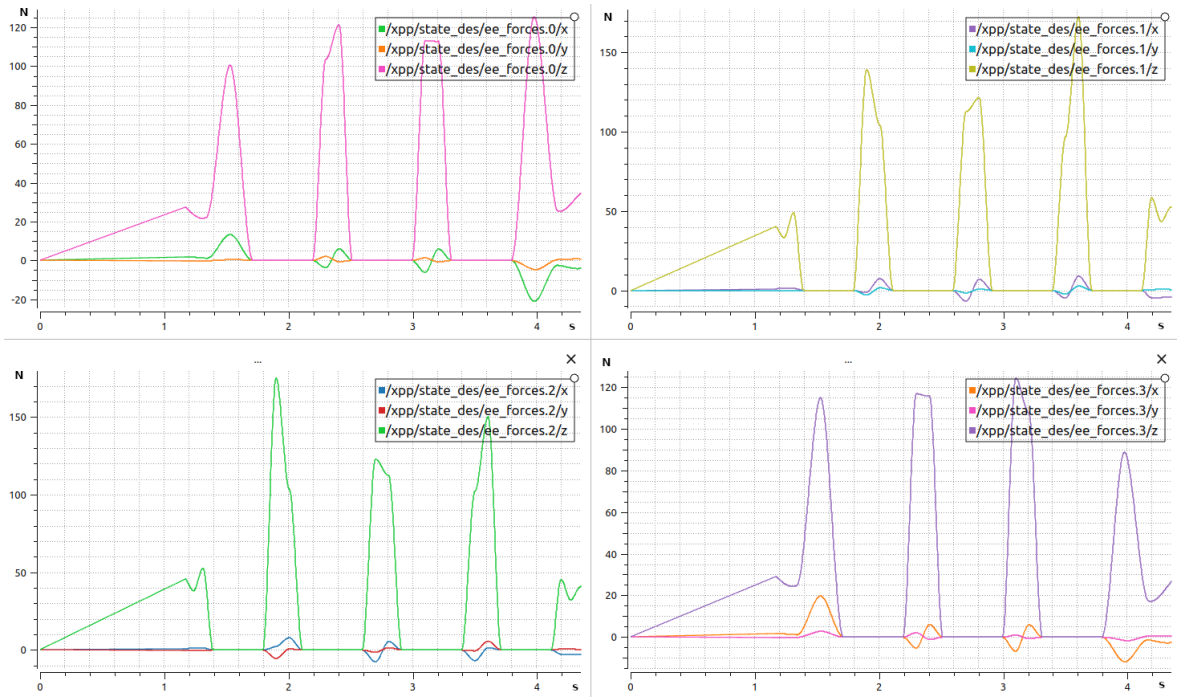


Figura 6.3: Visualización de las fuerzas tridimensionales de cada pata del robot con gravedad terrestre en la planificación. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha

Al analizar las fuerzas tridimensionales en las patas, se puede observar que cumplen con las condiciones establecidas por *towr*, explicadas en 4.2.1. Durante la fase de vuelo, cuando la pata no está en contacto con el suelo, se observa que la fuerza generada es nula. Por otro lado, durante la fase de apoyo, cuando la pata está en contacto con el suelo, se producen fuerzas; especialmente en el eje Z, que es la dirección perpendicular al punto de apoyo.

Es interesante destacar que durante la caminata, las fuerzas en las patas diagonalmente opuestas, se generan simultáneamente. Esto se debe a que las patas diagonales realizan el movimiento de avance o retroceso de manera conjunta, dejando siempre dos puntos de apoyo en el suelo. Este patrón de movimiento sincronizado es crucial para mantener el equilibrio y la estabilidad del robot durante la caminata. Al tener dos puntos de apoyo en todo momento, se logra una distribución de las fuerzas y evita desequilibrios que podrían resultar en caídas o movimientos inestables.

Finalmente, en el caso de la gravedad terrestre, se procede a mostrar las posiciones articulares del robot. Estas posiciones articulares representan la configuración de las articulaciones de cada pata del robot en cada instante de tiempo de la trayectoria planificada.

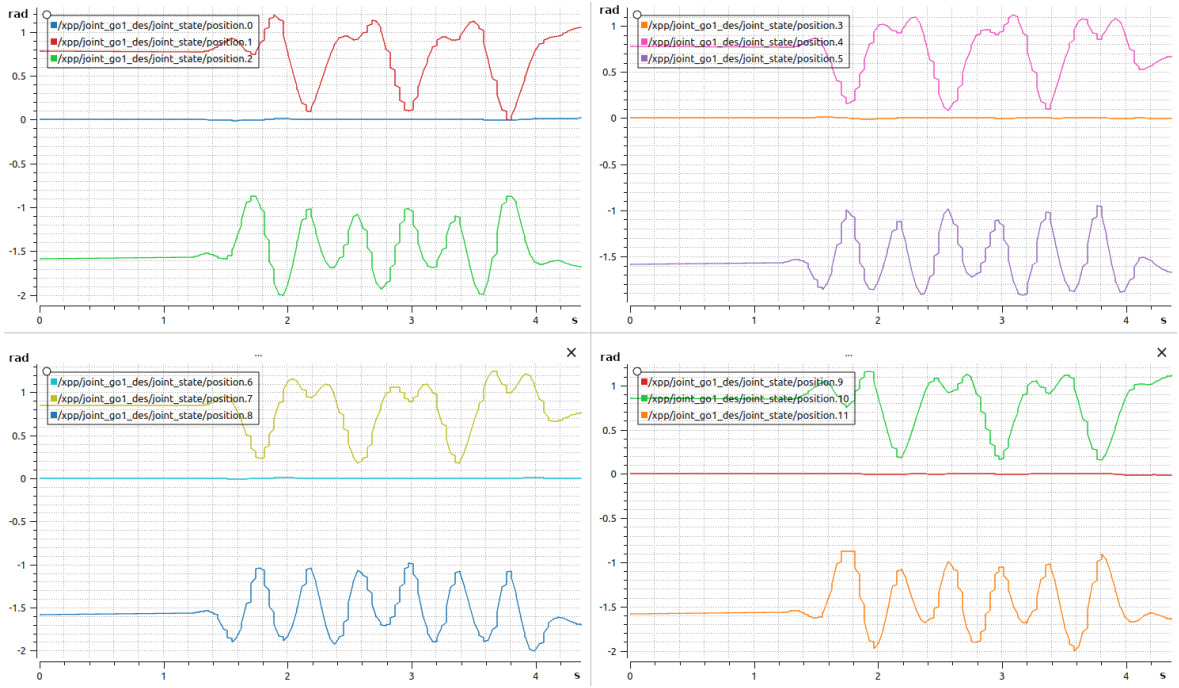


Figura 6.4: Visualización de la posiciones articulares de las articulaciones del robot con gravedad terrestre en la planificación. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

Al analizar las posiciones articulares, se puede observar cómo el robot se mueve mediante los movimientos de cada articulación. Además, se pueden apreciar los cambios en las posiciones articulares que corresponden a las transiciones entre las fases de apoyo y vuelo de las patas.

Como se puede apreciar, las articulaciones las caderas se mantienen en una posición constante. Esto se debe a que no hay necesidad de girar o cambiar la orientación, ya que el desplazamiento es lineal.

Por otro lado, en las articulaciones de los muslos y tobillos se producen cambios en sus ángulos a medida que el robot avanza. Estos cambios se producen debido a que las articulaciones se flexionan o extienden en función de la fase.

Al igual que en las gráficas de fuerzas cartesianas, se ve un comportamiento sincronizado de las patas que se encuentran en diagonal por parte de las articulaciones de los muslos y tobillos.

Por ejemplo, si observamos las articulaciones de los muslos, cuando una pata se eleva, la articulación correspondiente de la pata en posición diagonal se eleva prácticamente en ese mismo momento. Y lo mismo ocurre con las articulaciones de los tobillos. Esto se debe a la naturaleza del movimiento coordinado que se realiza para mantener el equilibrio, teniendo siempre dos puntos de contacto como mínimo.

6.1.2 Gravedad intermedia

A continuación, se genera la misma trayectoria, pero para una gravedad intermedia que se encuentra entre los valores de la gravedad terrestre y la gravedad marciana, concretamente 6.76388 m/s^2. El objetivo es observar cómo afecta la gravedad al movimiento del robot a medida que se disminuye gradualmente.

En primer lugar, al igual que en el apartado anterior, se muestra el movimiento de la base del robot a lo largo de la trayectoria planificada.

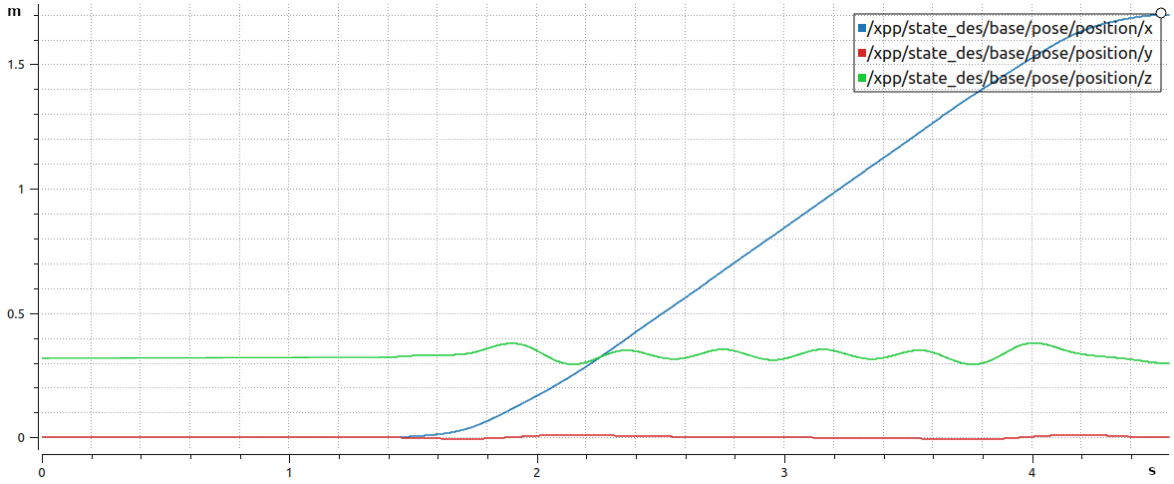


Figura 6.5: Visualización de la posiciones cartesianas de la base del robot respecto al mundo con gravedad intermedia en la planificación

Al generar la misma trayectoria pero con una gravedad diferente, se puede observar que el movimiento del robot en los ejes X e Y es prácticamente idéntico al observado con gravedad terrestre. Esto se debe a que, aunque exista una gravedad distinta, el robot debe realizar la misma trayectoria de igual forma, independientemente de esta.

Sin embargo, se puede notar una ligera disminución en la oscilación en el eje Z posiblemente causado por la reducción de las aceleraciones y las fuerzas generadas que se explican a continuación.

El siguiente paso consiste en mostrar las aceleraciones lineales y angulares del robot en el caso de la gravedad intermedia, lo que permite analizar las variaciones de las aceleraciones en comparación con las obtenidas con la gravedad terrestre.

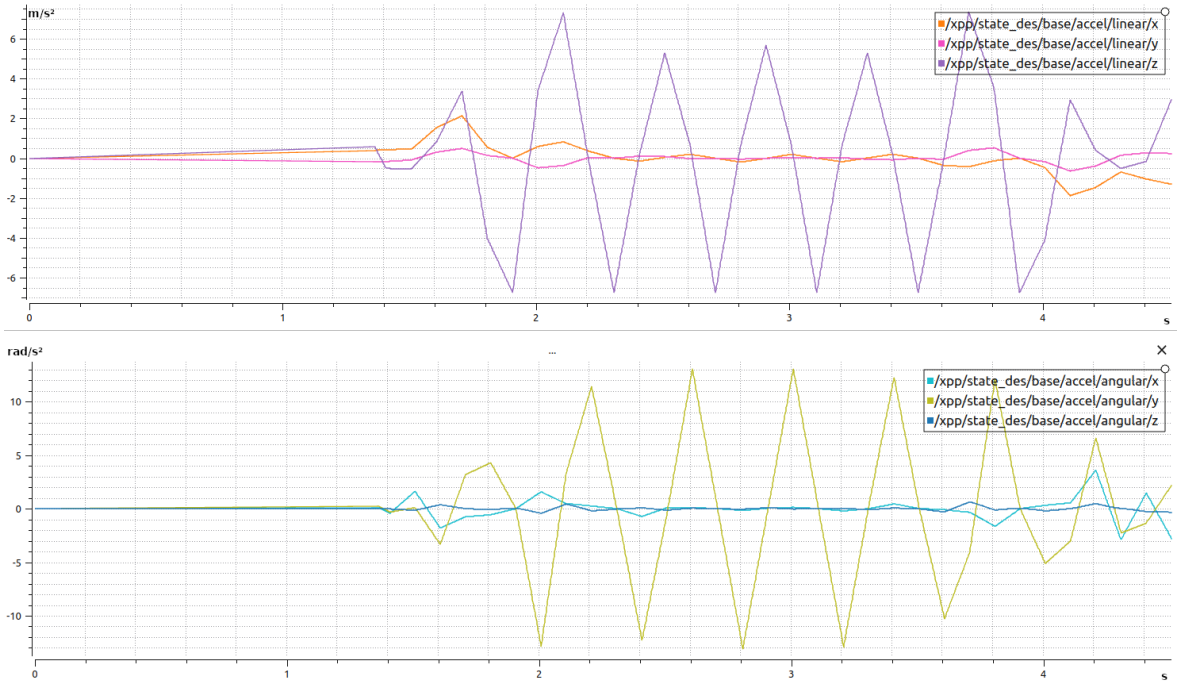


Figura 6.6: Visualización de las aceleraciones lineales y angulares tridimensionales del robot con gravedad intermedia en la planificación. En la parte superior se observa la aceleración lineal y en la inferior la angular.

Tal y como se esperaba, al disminuir la gravedad, se observa una reducción en las aceleraciones experimentadas por el mismo. Esto se debe a que la gravedad es una fuerza que actúa sobre el robot, generando una fuerza hacia abajo que afecta a la respuesta dinámica del robot. Al disminuir la gravedad, la intensidad de esta fuerza también disminuye, lo que se traduce en una disminución en las aceleraciones lineales y angulares del robot, ya que el robot requiere una fuerza menor para moverse en contra de la gravedad.

Al igual que en el caso de la gravedad terrestre, a continuación se muestran las fuerzas cartesianas que actúan sobre el robot a lo largo de la trayectoria planificada pero en gravedad intermedia. Al modificar la gravedad, se esperan cambios en las fuerzas generadas en los momentos de apoyo de las patas.

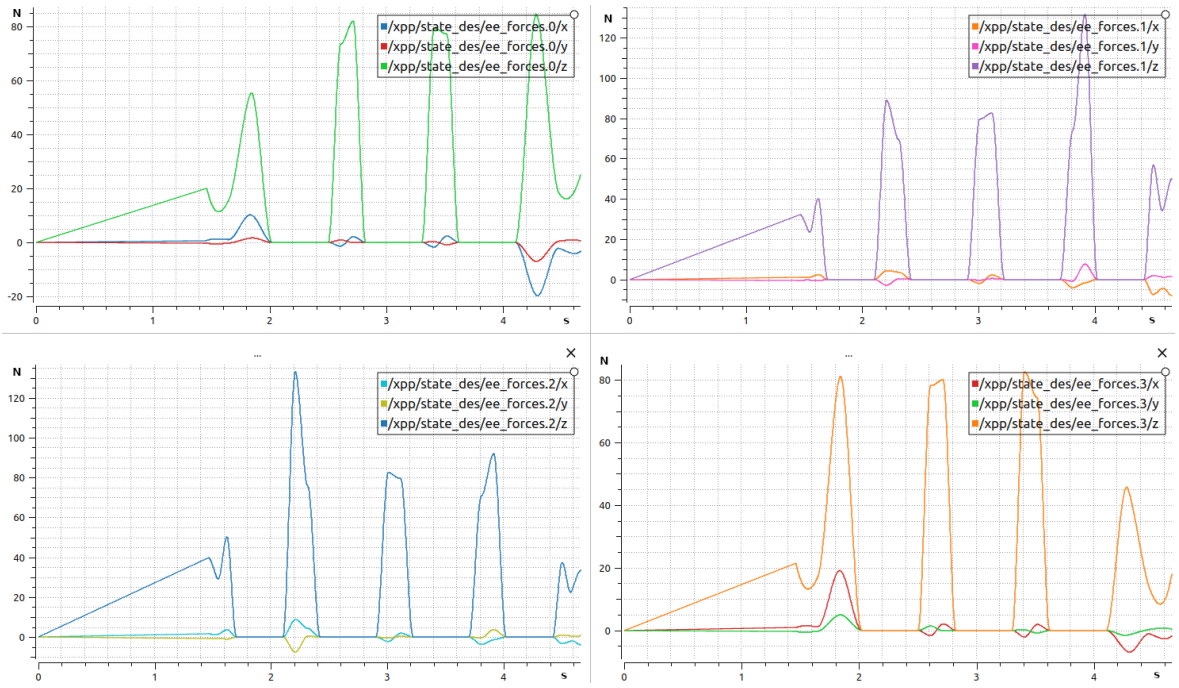


Figura 6.7: Visualización de las fuerzas tridimensionales de cada pata del robot con gravedad intermedia en la planificación. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha

Y como se puede observar, en comparación con la gravedad terrestre donde las fuerzas en la dirección Z eran más pronunciadas debido a la resistencia gravitatoria, en el caso de la gravedad intermedia, se observa una disminución en la intensidad de estas fuerzas. Esto se debe a que la gravedad intermedia es menor que la terrestre, lo que resulta en una menor carga gravitatoria que actúa sobre el robot.

Es importante destacar que, a pesar de la disminución en las fuerzas en la dirección Z, las restricciones impuestas por el modelo de planificación se mantienen independientemente de la gravedad establecida. Durante la fase de vuelo de las patas, las fuerzas generadas son nulas, mientras que durante la fase de apoyo, se producen fuerzas para mantener el equilibrio y soportar la carga del robot.

Al igual que en el caso de gravedad terrestre, en el escenario de gravedad intermedia también se muestran las posiciones articulares de las articulaciones del robot.

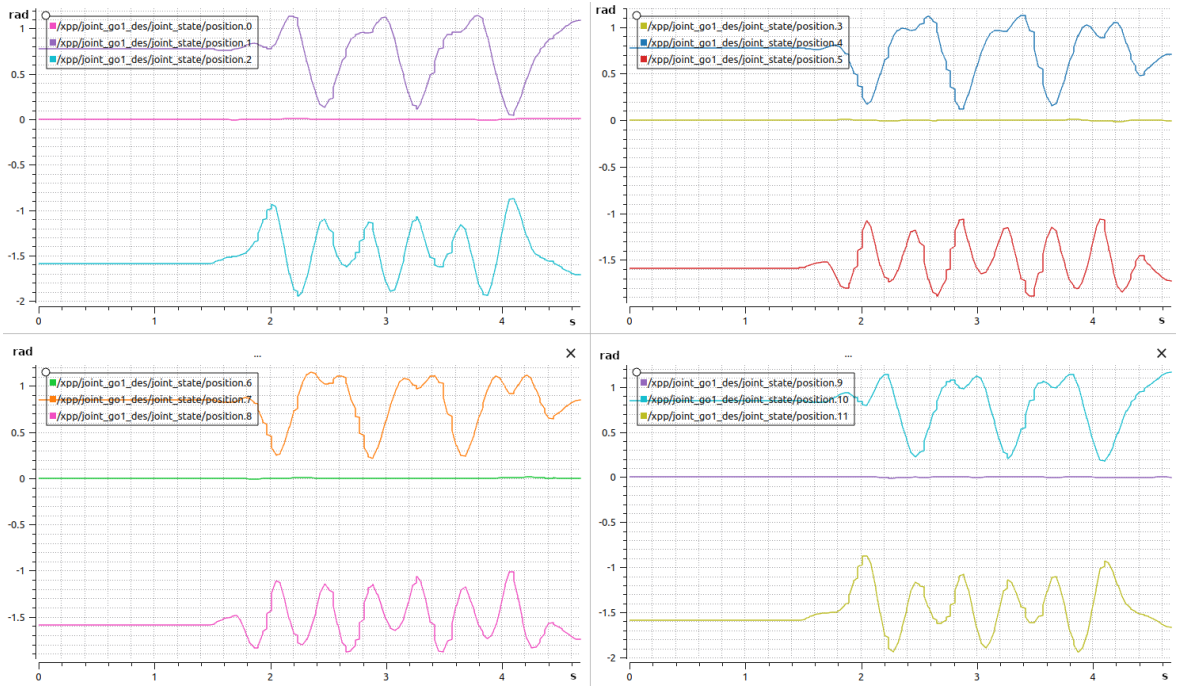


Figura 6.8: Visualización de la posiciones articulares de las articulaciones del robot con gravedad intermedia en la planificación. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

Tal y como ocurre con la posiciones cartesianas de la base, como la trayectoria que se planifica es la misma, el movimiento de lar articulaciones tiene que ser prácticamente idéntico al caso de la gravedad terrestre para poder realizar la trayectoria correctamente independientemente del cambio en la gravedad.

Pese a ello, se puede observar en las articulaciones de muslos y tobillos una amplitud ligeramente menor, lo que puede ser la causa de la pequeña disminución de la oscilación en Z del robot.

6.1.3 Gravedad marciana

En el siguiente caso de prueba, se considera la gravedad de Marte como la última variante gravitatoria a explorar. La gravedad marciana es significativamente menor que la terrestre, lo que implica que las fuerzas y las aceleraciones experimentadas por el robot se esperan aún más reducidas en comparación con los casos anteriores afectando a la respuesta física y dinámica del robot durante su movimiento.

A continuación, se presenta el movimiento de la base del robot a lo largo de la trayectoria planificada.

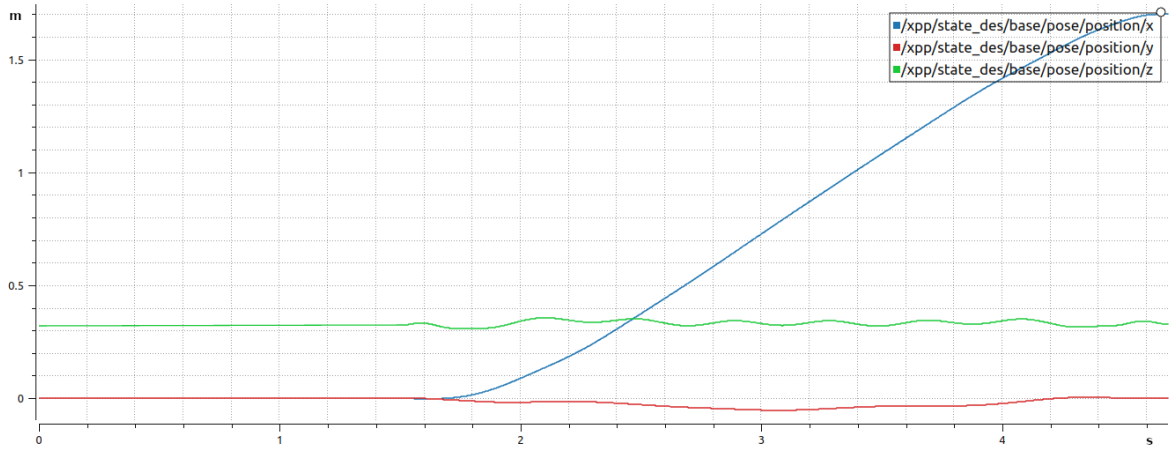


Figura 6.9: Visualización de la posiciones cartesianas de la base del robot respecto al mundo con gravedad marciana en la planificación

Dado que la trayectoria es la misma en todos los casos de gravedad, como se puede observar, el movimiento en las coordenadas X e Y es prácticamente idéntico al observado en las gravedades terrestre e intermedia.

Sin embargo, debido a la menor gravedad en Marte, se observa una mayor disminución de la oscilación del movimiento en Z puede deberse a que la fuerza gravitatoria que actúa sobre el robot es la más baja probada, lo que resulta en la menor amplitud en el movimiento vertical de la base.

El siguiente paso, al igual que en el resto de casos, se muestran las aceleraciones lineales y angulares del robot para la gravedad marciana.

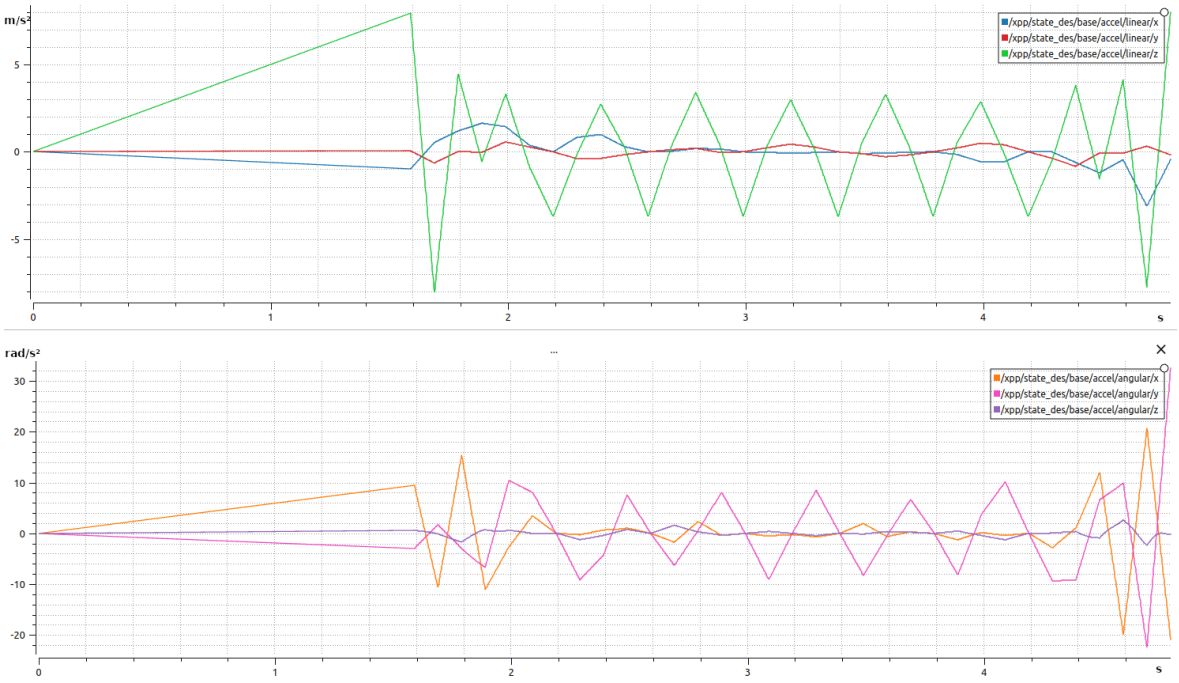


Figura 6.10: Visualización de las aceleraciones lineales y angulares tridimensionales del robot con gravedad marciana en la planificación. En la parte superior se observa la aceleración lineal y en la inferior la angular.

A medida que la gravedad disminuye, tanto en el caso de la gravedad intermedia como en la gravedad marciana, se observa una reducción en las aceleraciones lineales y angulares experimentadas por el robot. Sin embargo, debido a que la gravedad marciana es aún menor que la gravedad intermedia, las aceleraciones generadas en este entorno son aún más pequeñas.

Como se ha explicado anteriormente, esta disminución en las aceleraciones se debe a que la gravedad ejerce una fuerza menor sobre el robot en entornos con una gravedad más baja. Como resultado, el robot requiere menos aceleración para lograr el mismo nivel de movimiento y control.

A continuación, se presentan las fuerzas cartesianas experimentadas por el robot en el entorno de gravedad marciana.

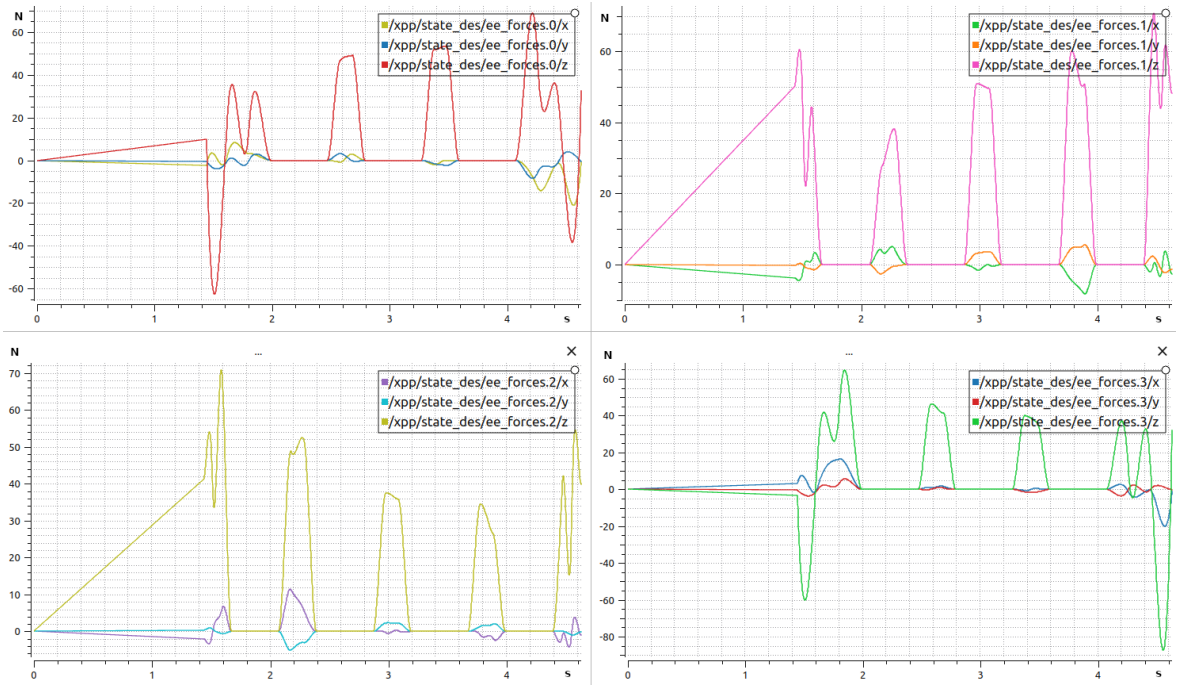


Figura 6.11: Visualización de las fuerzas tridimensionales de cada pata del robot con gravedad marciana en la planificación. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha

Al igual que en los casos anteriores y al igual que ocurre con las aceleraciones, se puede observar que a medida que la gravedad disminuye, las fuerzas generadas también se reducen. En el caso de la gravedad marciana, al ser aún menor que la gravedad terrestre e intermedia, las fuerzas son notablemente más pequeñas.

Esto se debe a que la gravedad marciana ejerce una fuerza menor sobre el robot, lo que resulta en una menor carga sobre las patas. Como resultado, las fuerzas de reacción en las articulaciones y las fuerzas totales en el sistema son más reducidas en comparación con el resto de gravedades.

Finalmente, al igual, que en los dos casos anteriores, se muestran las posiciones de las articulaciones a lo largo de la trayectoria.

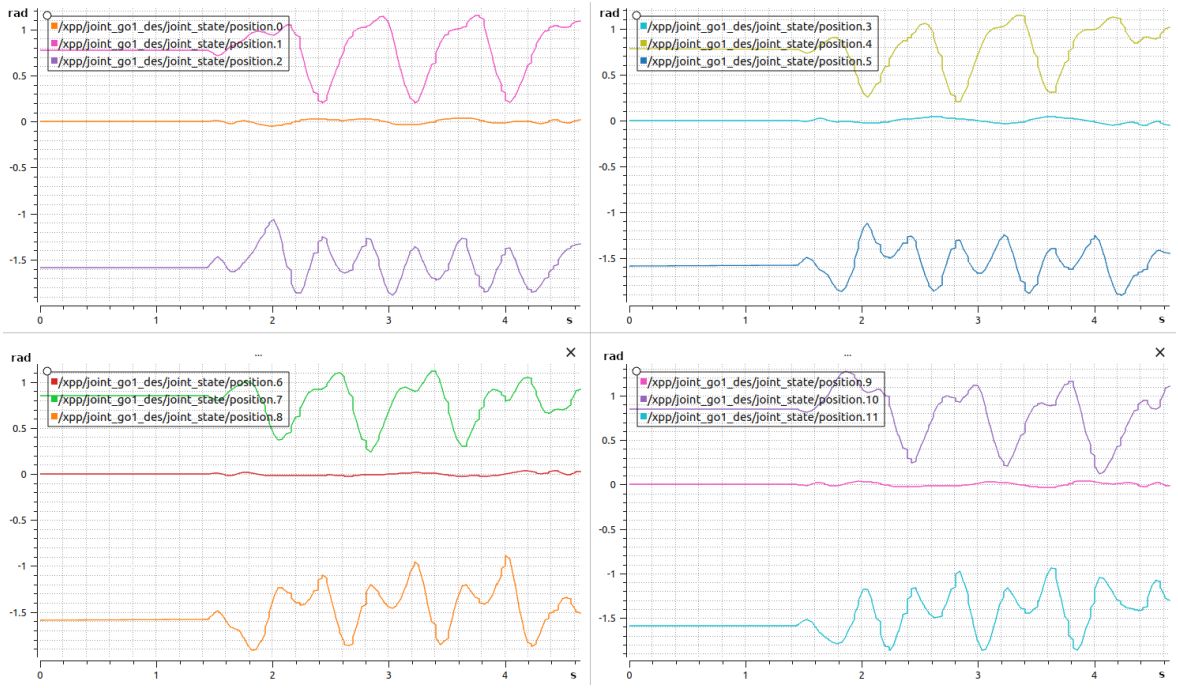


Figura 6.12: Visualización de la posiciones articulares de las articulaciones del robot con gravedad marciana en la planificación. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

A pesar de la diferencia en la gravedad, tal y como ocurre con la gravedad intermedia, se observa que las posiciones articulares se mantienen prácticamente idénticas a las obtenidas en la gravedad terrestre e intermedia, cosa que se esperaba.

6.1.4 Comparaciones

Como se ha podido observar en los diferentes casos, al realizar la misma trayectoria independientemente de la gravedad, el movimiento de las articulaciones y el movimiento del cuerpo es prácticamente idéntico. Esto se debe a que la planificación de la trayectoria se realiza teniendo en cuenta las características y limitaciones del robot, y se genera con las condiciones gravitatorias deseadas.

Aunque la gravedad puede variar entre la Tierra y otros planetas o entornos, el objetivo es mantener la consistencia en el movimiento de las articulaciones para garantizar una ejecución adecuada de la trayectoria deseada. Esto implica que las articulaciones deben seguir un patrón similar de flexión y extensión en los tres caso para realizar la misma trayectoria.

Sin embargo, se ha podido observar que al disminuir la gravedad, la oscilación en el eje Z del robot disminuye a consecuencia de la reducción de las aceleraciones y fuerzas. Este resultado tiene implicaciones importantes para la planificación y control de robots en entornos de baja gravedad, ya que permite comprender cómo la gravedad afecta al movimiento del robot en el eje vertical.

Finalmente, importante destacar que, si bien el movimiento de las articulaciones puede ser similar, las fuerzas y las cargas ejercidas sobre las articulaciones pueden variar según la gravedad. Esto se debe a que en entornos con menor gravedad, el robot necesita un menor esfuerzo para moverse, ya que la fuerza gravitatoria ejercida sobre él es menor. Por lo tanto, las fuerzas y aceleraciones requeridas para mantener la trayectoria son menores en comparación con la gravedad terrestre. Todo esto implica que el control y la planificación deben adaptarse para garantizar una ejecución segura y eficiente de la trayectoria en diferentes entornos gravitatorios.

6.2 Resultados del seguimiento

Tras realizar las pruebas enfocadas a la planificación de trayectorias con las tres gravedades diferentes, es momento de analizar y comparar los resultados obtenidos para el seguimiento en simulación de dichas trayectorias. El objetivo es evaluar si el robot es capaz de mantener la trayectoria deseada a lo largo del tiempo y en las diferentes condiciones gravitatorias.

Para llevar a cabo esta evaluación se examinan las aceleraciones del robot, así como las posiciones y pares articulares de las patas. Se espera que, a pesar de las diferencias en la gravedad, el robot sea capaz de mantener un seguimiento adecuado de las trayectorias en todos los casos. Sin embargo, es posible que se observen variaciones en los valores medidos, especialmente en los pares y las aceleraciones, debido a las diferencias en la influencia gravitatoria.

6.2.1 Gravedad terrestre

En primer lugar se realiza un análisis de las aceleraciones que experimenta el robot durante el movimiento. Se comparan las aceleraciones obtenidas durante el seguimiento con las aceleraciones generadas en la planificación.

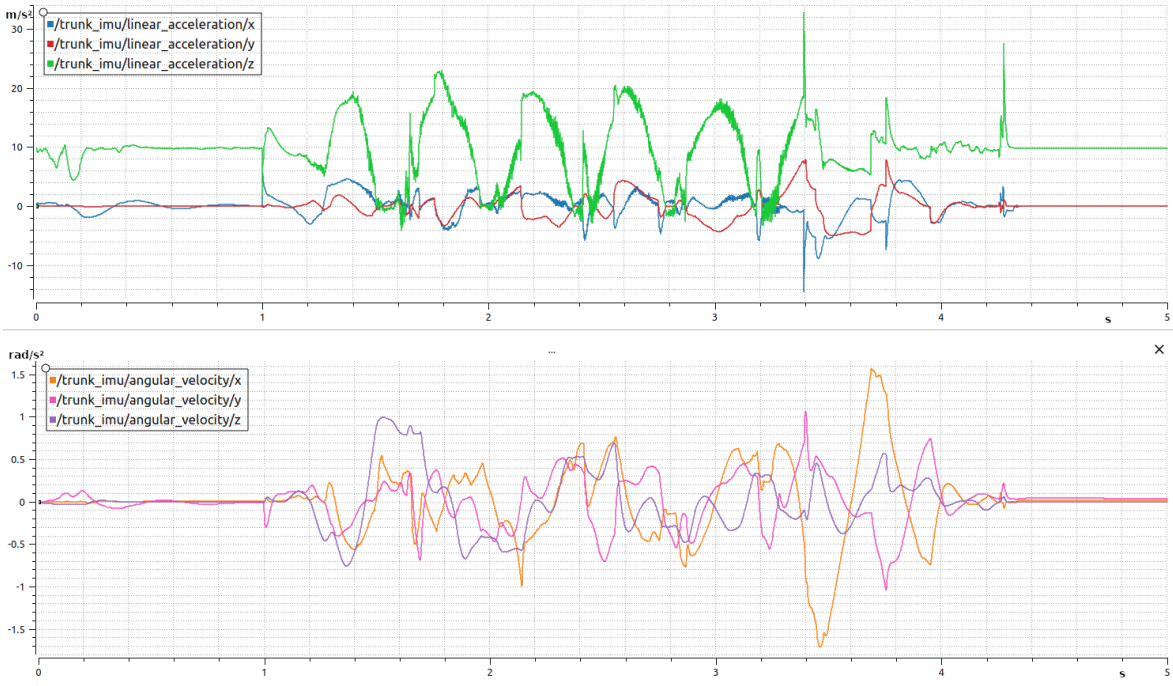


Figura 6.13: Visualización de las aceleraciones lineales y angulares tridimensionales del robot con gravedad terrestre en el seguimiento. En la parte superior se observa la aceleración lineal y en la inferior la angular.

En cuanto a las aceleraciones lineales, se observa que son bastantes más altas que las previstas en la planificación. Esto puede deberse a factores externos del entorno que *towr* no tiene en cuenta y pueden afectar el movimiento del robot.

Por otro lado, las aceleraciones angulares durante el seguimiento son mucho más bajas que las previstas en la planificación y presentan valores similares entre ellas. Esto puede ser atribuido a los esfuerzos del controlador del Go1 para mantener la estabilidad del robot y reducir cualquier oscilación no deseada.

A pesar de las diferencias en las aceleraciones lineales y angulares entre la planificación y el seguimiento de la trayectoria en gravedad terrestre, aún se pueden observar los efectos de la aceleración lineal en Z debido al movimiento del cuerpo del robot en vertical. Esta oscilación produce cambios en la aceleración a lo largo de la trayectoria, lo cual es un fenómeno esperado.

El siguiente paso es analizar los pares articulares obtenidos a partir de las fuerzas cartesianas de la planificación y que se han aplicado en el seguimiento de la trayectoria.

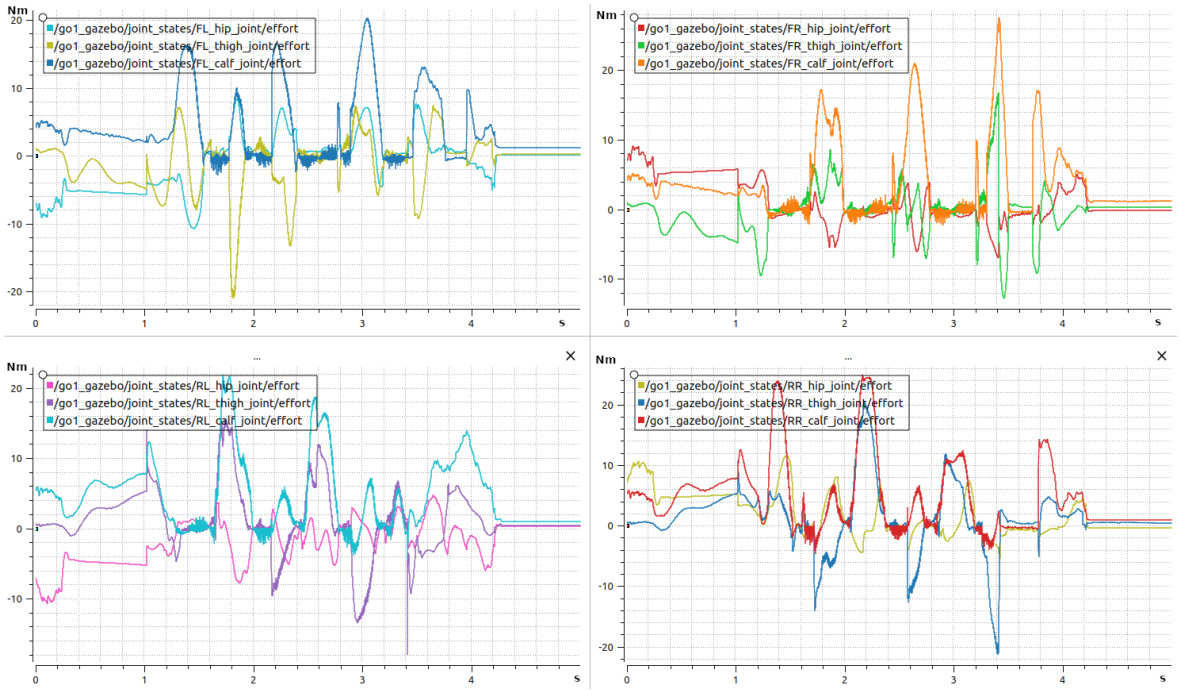


Figura 6.14: Visualización de los pares articulares de cada articulación del robot con gravedad terrestre en el seguimiento. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

En las gráficas se puede observar la distribución de los pares articulares en cada una de las articulaciones de cadera, muslo y tobillo. Es interesante observar que, al igual que ocurre en las fuerzas cartesianas obtenidas en la planificación, en los pares articulares se puede ver el mismo patrón simultáneo entre las patas diagonalmente opuestas.

Como se ha comentado anteriormente, cuando una pata se eleva para avanzar, la pata diagonalmente opuesta también experimenta un cambio en sus ángulos articulares y por lo tanto en pares, lo que garantiza una distribución equilibrada del peso y una estabilidad adecuada durante el movimiento.

Finalmente, con respecto a la gravedad terrestre, se realiza una comparativa de las posiciones articulares obtenidas durante la ejecución del movimiento frente a las obtenidas en la planificación.

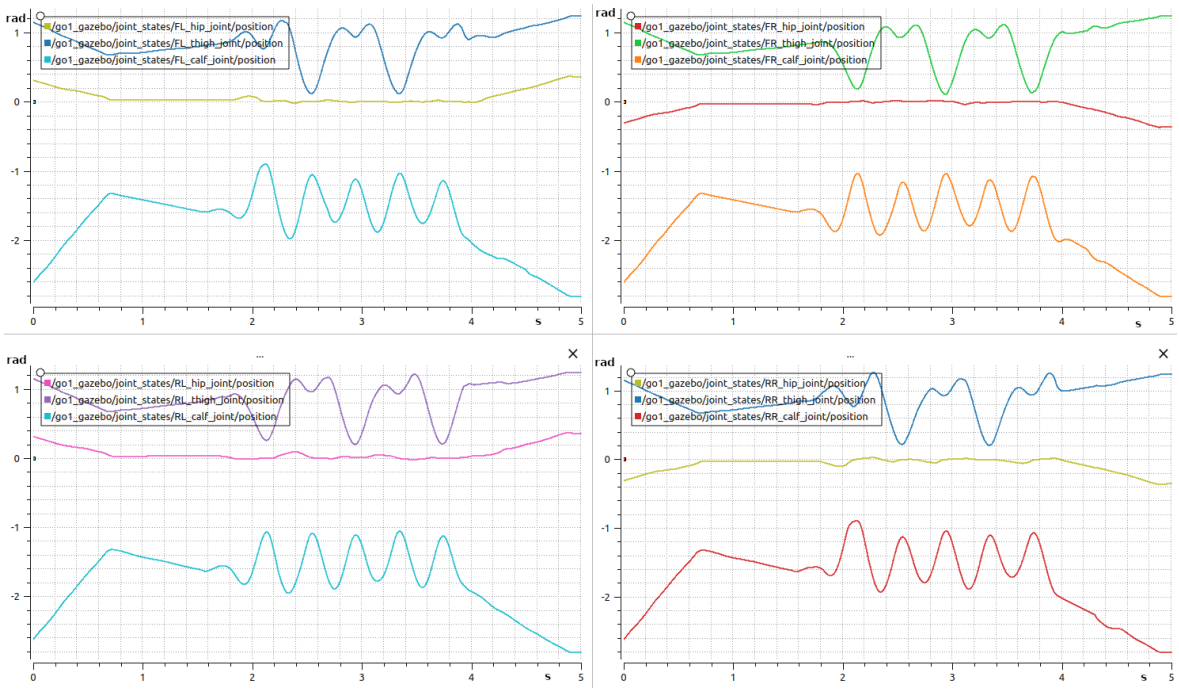


Figura 6.15: Visualización de la posiciones articulares de las articulaciones del robot con gravedad terrestre en el seguimiento. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

Al inicio de cada gráfica se observa que las articulaciones se mueven hasta una determinada posición. Este movimiento inicial corresponde a la interpolación realizada para llevar al robot desde su posición de reposo en el suelo hasta la posición inicial de la trayectoria planificada.

Una vez que el robot ha completado la trayectoria, en la parte final de la gráfica, se observa cómo las articulaciones se mueven nuevamente, esta vez regresando a la posición de reposo en el suelo, su posición original.

Estos movimientos de transición al inicio y al final de la trayectoria son parte del proceso de preparación y finalización del movimiento planificado. La interpolación inicial garantiza que el robot se encuentre en la posición adecuada para comenzar la trayectoria, mientras que el movimiento final asegura que el robot regrese a su posición de reposo después de completar la tarea.

Con respecto a las posiciones articulares durante la trayectoria, se observa que se mantienen prácticamente idénticas a las planificadas. Sin embargo, se puede ver una trayectoria de movimiento más suave en las articulaciones debido a que la frecuencia de publicación es mayor para ejecutarla en el robot.

Por lo tanto, esto demuestra la eficacia del controlador del Go1, la capacidad del robot

para seguir la trayectoria planificada con un buen grado de precisión y que se han elegido correctamente los valores de las ganancias.

6.2.2 Gravedad intermedia

Una vez finalizado el análisis de la trayectoria en condiciones de gravedad terrestre, se comprueba si el robot es capaz de seguir la misma trayectoria pero generada para ser realizada en gravedad intermedia. Esto permite observar cómo afecta la variación de la gravedad en el movimiento del robot y en las diferentes variables analizadas. Al igual que en la condición anterior, se evalúan las aceleraciones lineales y angulares, los pares y las posiciones articulares.

En primer lugar, se muestran las aceleraciones experimentadas por el robot. Se busca comparar las aceleraciones lineales y angulares registradas en este seguimiento con las correspondientes y obtenidas con gravedad terrestres.

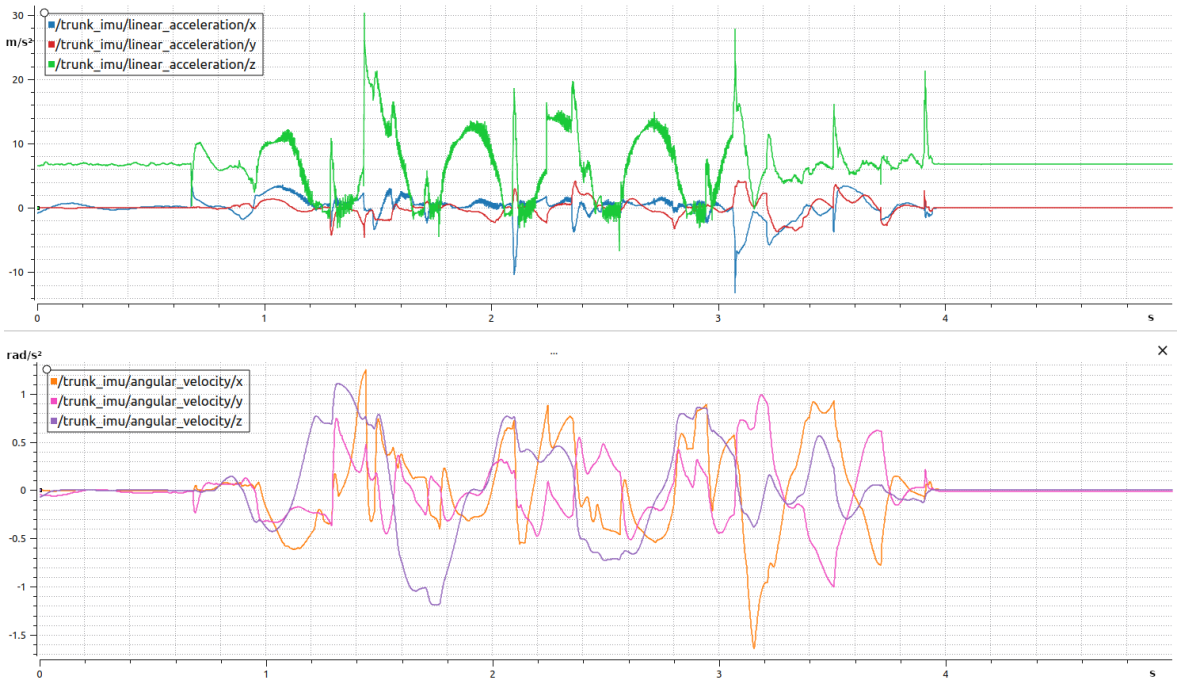


Figura 6.16: Visualización de las aceleraciones lineales y angulares tridimensionales del robot con gravedad intermedia en el seguimiento. En la parte superior se observa la aceleración lineal y en la inferior la angular.

Al igual que en el caso anterior, las aceleraciones lineales registradas durante este seguimiento son más altas en comparación con las obtenidas en la planificación de la trayectoria. Mientras que, por otro lado, las aceleraciones angulares en esta gravedad también difieren, ya que son más bajas.

Además, al comparar los valores obtenidos durante el seguimiento en gravedad intermedia con los del seguimiento en gravedad terrestre, se observa que existen diferencias. Se ha no-

tado que, en general, los valores de las aceleraciones lineales y angulares son más bajos en comparación con el seguimiento en gravedad terrestre, lo que implica que el robot requiere un menor esfuerzo y energía para mantener la trayectoria deseada en una gravedad menor. Esta disminución en las aceleraciones puede atribuirse a la menor fuerza gravitacional que actúa sobre el robot en este escenario.

A continuación, se presentan los pares articulares obtenidos durante el seguimiento de la trayectoria en gravedad intermedia. Como ya se ha comentado, los pares articulares representan las fuerzas ejercidas en las articulaciones del robot para mantener y seguir la trayectoria planificada.

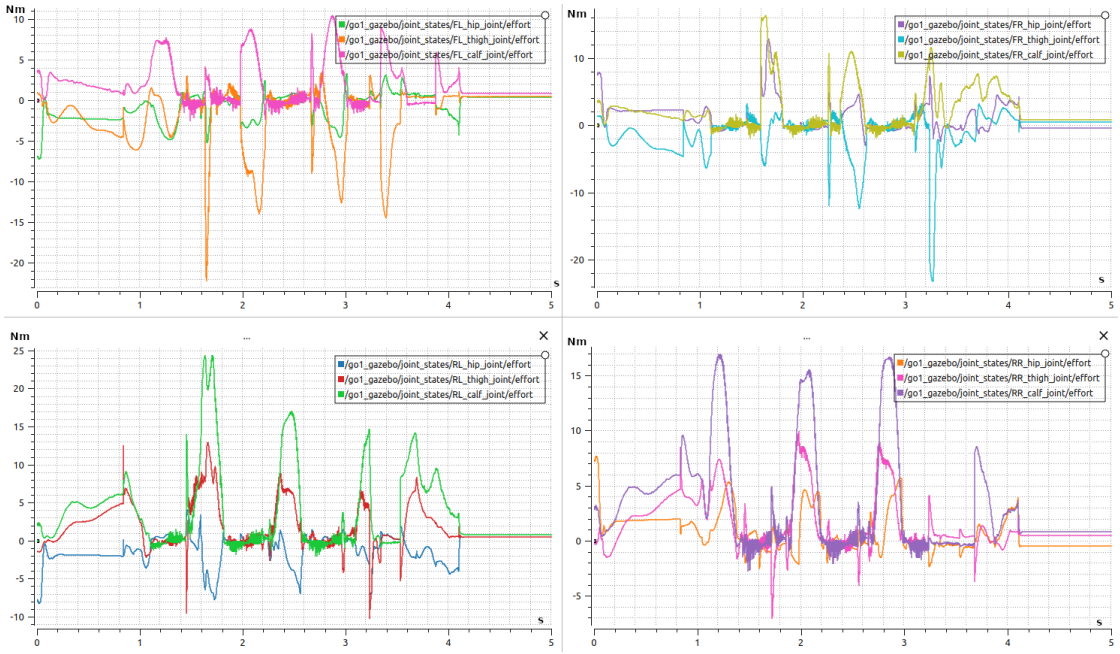


Figura 6.17: Visualización de los pares articulares de cada articulación del robot con gravedad intermedia en el seguimiento. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

En general, los valores de los pares articulares son menores en este caso. Esto indica que, debido a la menor fuerza gravitacional, el robot requiere un esfuerzo y una fuerza reducidos para mantener y seguir la trayectoria. Las articulaciones experimentan una carga menor y se ejercen fuerzas más bajas en comparación con el caso de la gravedad terrestre.

Un aspecto importante a destacar al analizar los pares articulares, es que nuevamente se observa el movimiento simultáneo de las patas diagonales del robot. Al igual que en gravedad terrestre, en la intermedia, se encuentran similitudes en los patrones de las fuerzas cartesianas de la planificación y los pares articulares del seguimiento. Esto sugiere que, aunque las magnitudes de los pares sean diferentes debido a la variación en la gravedad, el comportamiento y la coordinación de las articulaciones se mantienen consistentes.

Por último, se presentan las posiciones articulares obtenidas durante el seguimiento en gravedad intermedia.

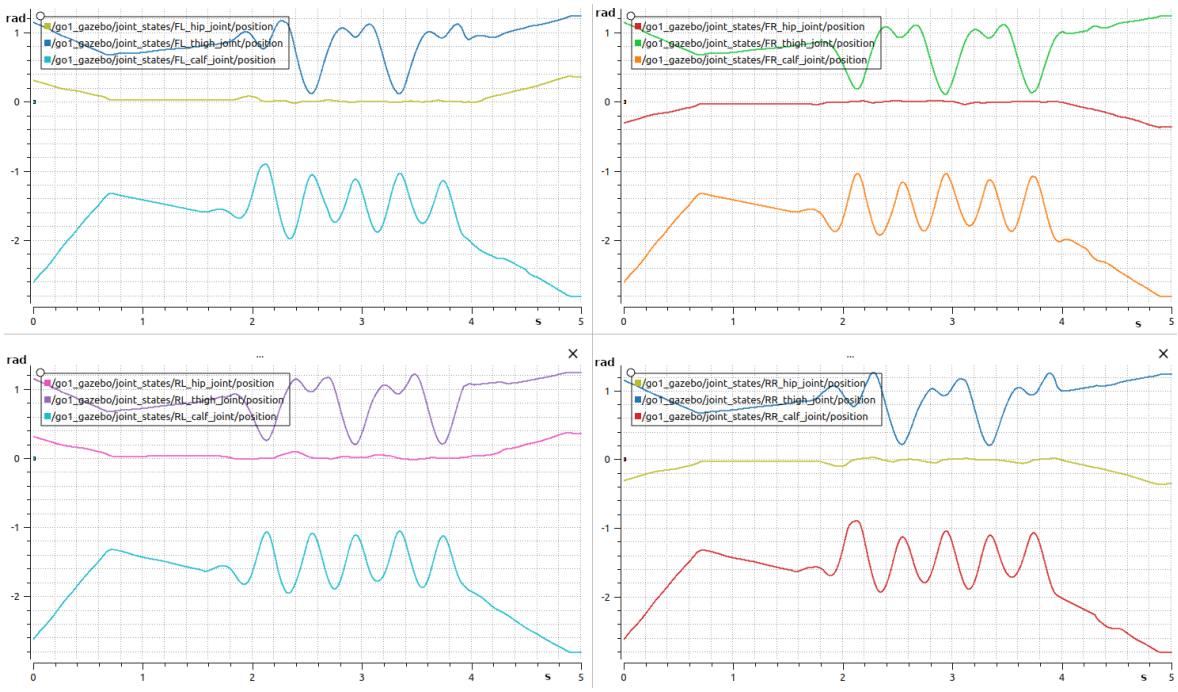


Figura 6.18: Visualización de la posiciones articulares de las articulaciones del robot con gravedad intermedia en el seguimiento. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

Al analizar las posiciones articulares, se observa que se mantienen bastante similares a las obtenidas en gravedad terrestre. A pesar de las diferencias en las fuerzas y aceleraciones, el robot mantiene una alineación y coordinación adecuadas en sus articulaciones para realizar la trayectoria de manera precisa.

Al igual que se ha comentado en el apartado de la planificación, al generar la misma trayectoria pero para cada gravedad específica, el robot debe seguir esa misma ruta independientemente de la gravedad presente. Es por ello que las posiciones de las articulaciones prácticamente coinciden con las de la gravedad terrestre.

6.2.3 Gravedad marciana

Finalmente, se comparan los resultados obtenidos en el seguimiento de la trayectoria generada para la gravedad marciana. Como se ha estado mencionando, la gravedad marciana es considerablemente menor que la gravedad terrestre e intermedia, lo que implica que se esperan diferencias en los pares y aceleraciones manteniendo las posiciones articulares del robot.

Al igual que en todos los casos se comienza analizando las aceleraciones a las que se ha sometido el robot.

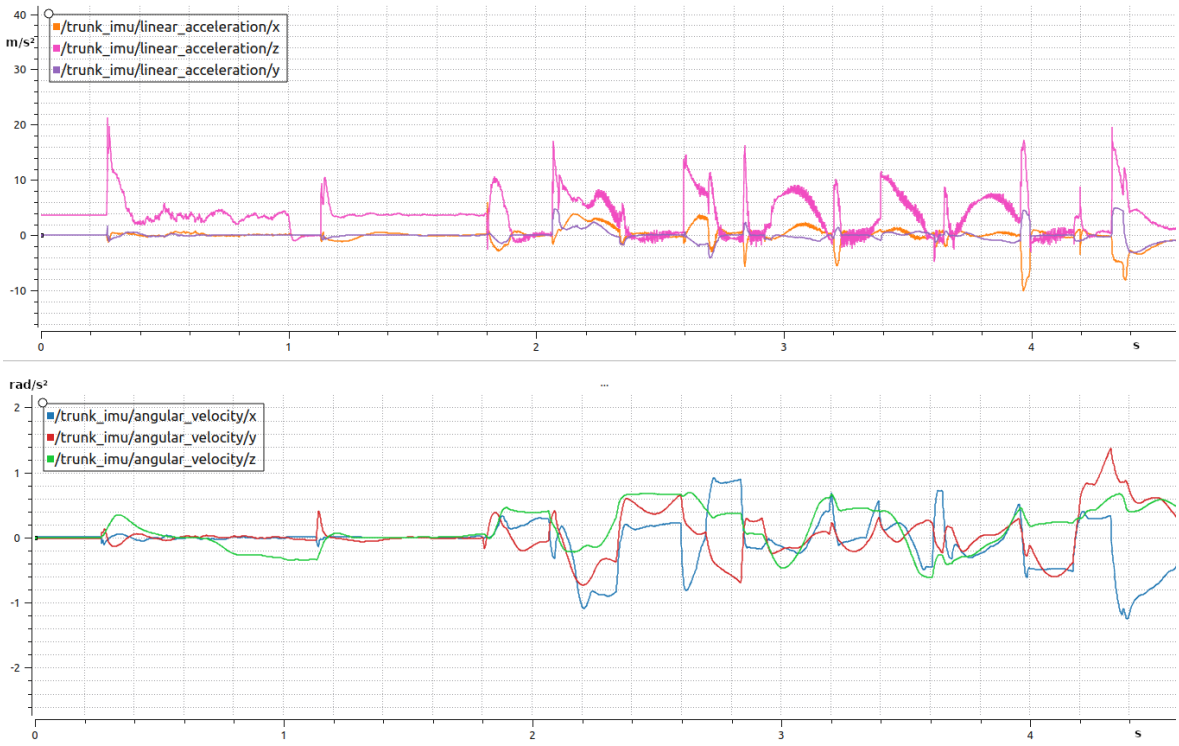


Figura 6.19: Visualización de las aceleraciones lineales y angulares tridimensionales del robot con gravedad marciana en el seguimiento. En la parte superior se observa la aceleración lineal y en la inferior la angular.

Tal y como ocurre con los dos casos anteriores las aceleraciones son diferentes de las obtenidas en la planificación. Sin embargo, como se espera, estas aceleraciones son menores que en la gravedad terrestre e intermedia debido a la reducción de la gravedad.

Como se ha comentado, además de analizar las aceleraciones, también se evalúan los pares generados en cada articulación durante el seguimiento de la trayectoria en gravedad marciana.

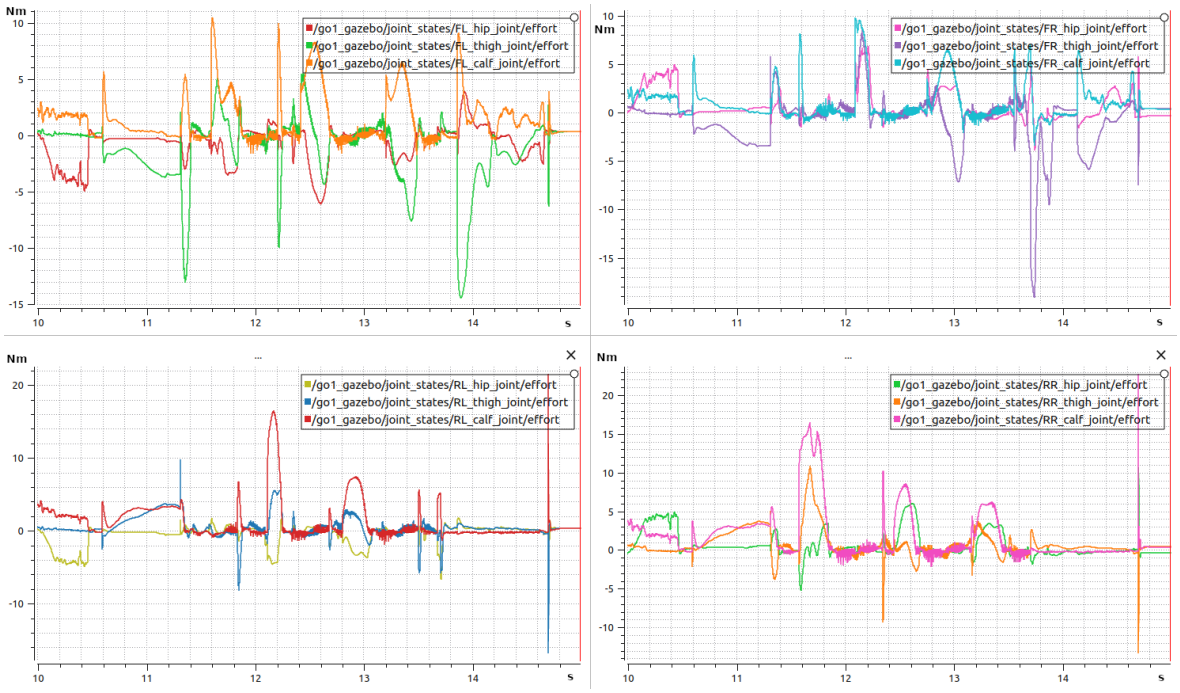


Figura 6.20: Visualización de los pares articulares de cada articulación del robot con gravedad marciana en el seguimiento. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

Tal y como se esperaba y al igual que en los casos anteriores, se observa que los pares articulares generados son aún más bajos en comparación con las condiciones de gravedad terrestre e intermedia.

La reducción de los pares articulares en gravedad marciana se debe nuevamente a la menor fuerza gravitacional a la que está expuesto el robot. Con una gravedad más débil, las articulaciones experimentan una carga reducida, lo que se traduce en pares más bajos durante el movimiento.

En último lugar, también se han evaluado las posiciones articulares de cada articulación.

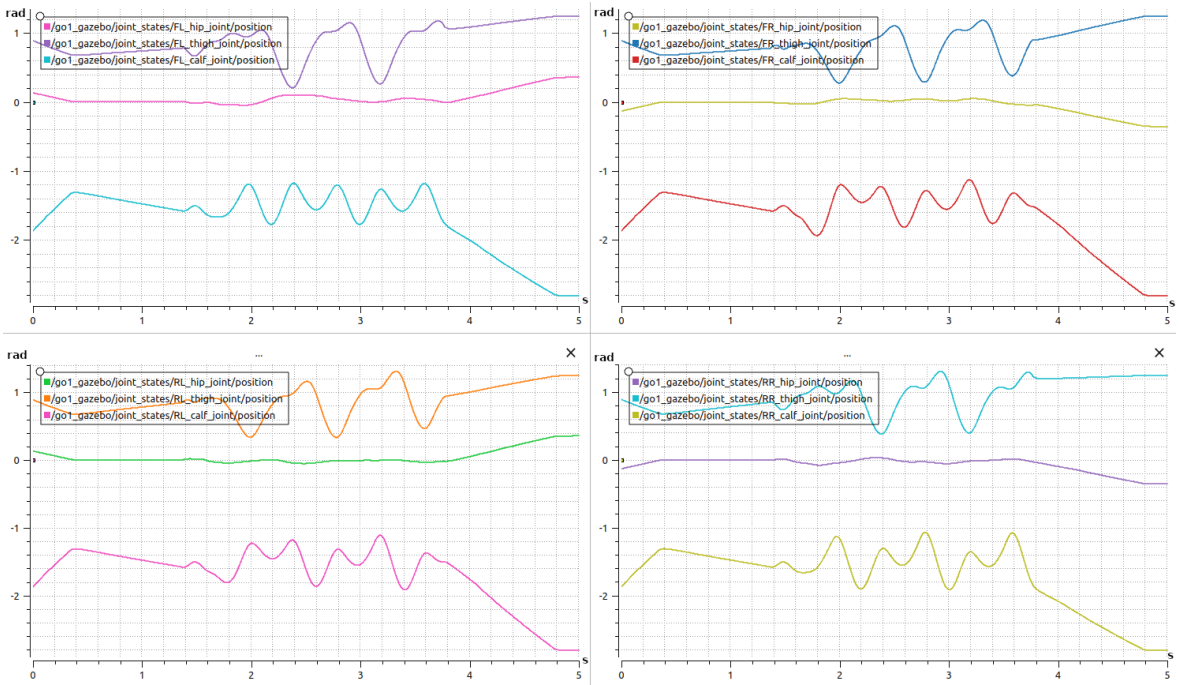


Figura 6.21: Visualización de la posiciones articulares de las articulaciones del robot con gravedad marciana en el seguimiento. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

Finalmente, es importante destacar que, al igual que en los casos anteriores, el seguimiento de la trayectoria generada para gravedad marciana es un éxito. El robot ha sido capaz de seguir correctamente la trayectoria planificada, cumpliendo con los movimientos y las posiciones deseadas y pese a los cambios surgidos en las aceleraciones.

6.2.4 Aclaraciones

Como se ha podido observar, se ha llevado a cabo el seguimiento de una misma trayectoria generada teniendo en cuenta tres condiciones de gravedad diferentes: terrestre, intermedia y marciana. Durante el seguimiento, se han analizado diversas variables, como aceleraciones y pares y posiciones articulares, con el objetivo de comprender el impacto de la gravedad en el movimiento del robot.

En el caso de la gravedad terrestre, se observa que las aceleraciones lineales son más altas durante el seguimiento en comparación con la planificación, mientras que las aceleraciones angulares son más bajas.

En la gravedad intermedia, se obtienen resultados similares, con aceleraciones lineales más altas que en la planificación y aceleraciones angulares más bajas. Asimismo, se observa una disminución general en las magnitudes de las aceleraciones y pares en comparación con la gravedad terrestre.

Por último, en la gravedad marciana, se constata nuevamente una reducción en las aceleraciones, tanto lineales como angulares, en comparación con los casos anteriores. Los valores de los pares articulares también son menores, lo que indica una menor resistencia al movimiento debido a la menor gravedad.

Finalmente, en términos de posiciones articulares, se observa que el seguimiento de la trayectoria se mantiene fiel a la planificación en los tres casos de gravedad, lo que indica la capacidad de seguir la trayectoria deseada.

A continuación se incluye un enlace a un vídeo en el que se puede ver la simulación de las trayectorias generadas y de su seguimiento para cada gravedad:

`https://youtu.be/bviYbNigV2s`

6.3 Pruebas en el robot real

Finalizadas las pruebas en simulación, se procede a realizar las pruebas en el robot real para comparar los datos obtenidos en las simulaciones con los resultados reales. El objetivo es observar posibles discrepancias y obtener conclusiones precisas.

Una vez conectado al robot real, se ejecuta la trayectoria estudiada en los pasos anteriores. Sin embargo, en el primer intento de ejecución, la velocidad de publicación es demasiado alta, lo que casi provoca la caída del robot. Esta discrepancia es la primera que se observa. Para solucionarlo, se reduce la frecuencia de publicación a la mitad, es decir, a 250 Hz. Con esta reducción, el robot es capaz de moverse sin voltearse, pero se identifica una segunda discrepancia.

A diferencia de lo que ocurre en la planificación y simulación, en la realidad el robot no levanta las patas, sino que las arrastra. Cuando las patas diagonalmente opuestas se levantan y dan un paso, el robot se mueve hacia adelante. Por otro lado, al arrastrar las patas durante el siguiente paso, el robot se mantiene en su lugar.

También se observa una última discrepancia sobre el número de puntos del *rosbag*, siendo insuficientes para generar una trayectoria suave y fluida en el robot real.

Teniendo en mente estas tres discrepancias, se generan y simulan un conjunto extenso de diferentes trayectorias con el objetivo de abordar estas limitaciones. Durante este proceso, se implementan acciones para intentar solucionar los tres problemas identificados.

En primer lugar, se realiza una modificación en la variable *visualization_dt_*, como se explica en el apartado 5.1.4, estableciendo su valor en 0.001 segundos. Esto resulta en el doble de puntos generados, resolviendo así el tercer problema relacionado con la insuficiencia de valores. Con esta modificación, la frecuencia de publicación se incrementa de 500 a 1000Hz. No obstante, debido al primer problema mencionado, se mantiene a 500Hz.

Finalmente, para evitar que el robot arrastre los pies se experimenta cambiando el tamaño del prisma descrito en el apartado 4.2.2.1. Específicamente, se acorta el prisma en el eje X y se alarga en el eje Z obligando al planificador a elevar más las patas del robot durante la ejecución de las trayectorias.

Tras generar un gran conjunto de trayectorias con diferentes tamaños del prisma, diferentes tiempos y diferentes distancias se obtiene la que se considera la mejor trayectoria con los siguientes valores:

- **Tamaño del prisma**: 0.09 m en X, 0.09 m en Y y 0.08 m en Z.
- **Tiempo**: 4 segundos.
- **Distancia**: 0.8 metros.

Con base a estos ajustes, se genera la trayectoria y se realiza el seguimiento en simulación, comparando los resultados con el seguimiento de la trayectoria en el robot real.

6.3.1 Planificación

Al igual que en el apartado de planificación de las tres gravedades, en primer lugar se muestra el movimiento de la base del robot.

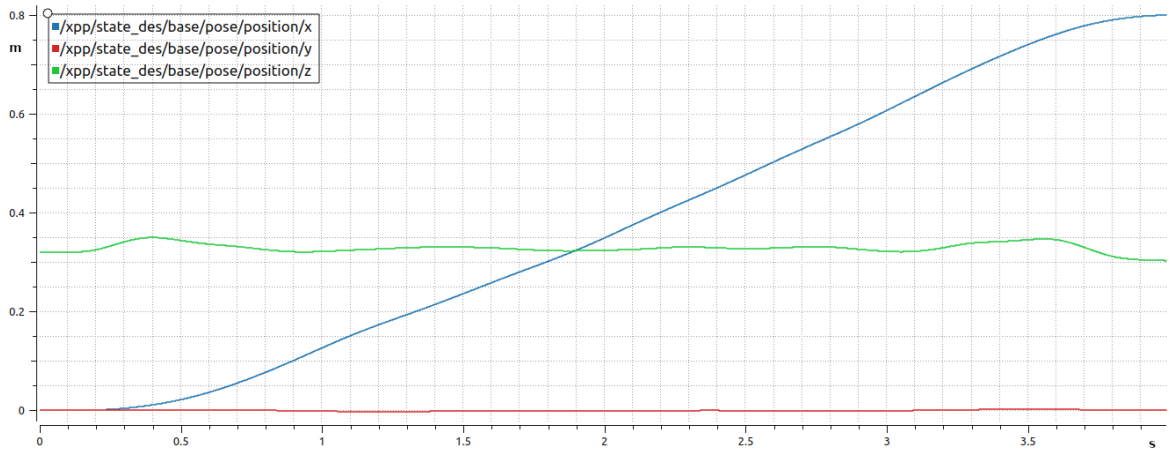


Figura 6.22: Visualización de la posiciones cartesianas de la base del robot respecto al mundo en la planificación de la nueva trayectoria

Con respecto a la nueva trayectoria, se puede observar claramente el comportamiento del desplazamiento en la gráfica. Al igual que en la anterior, la nueva trayectoria sigue siendo una línea recta en el plano XY, lo que implica que la posición en el eje X del robot aumenta de manera continua a medida que avanza en el tiempo.

El movimiento en Y, también se mantiene constante en cero, ya que no hay desplazamiento lateral en la trayectoria.

Sin embargo, en comparación con la trayectoria anterior, la oscilación en el eje Z es menor en la nueva. Esto se debe a que la trayectoria es más corta y el robot tarda más tiempo en recorrerla, lo que resulta en una reducción de la oscilación vertical.

El siguiente paso es obtener las aceleraciones lineales y angulares.

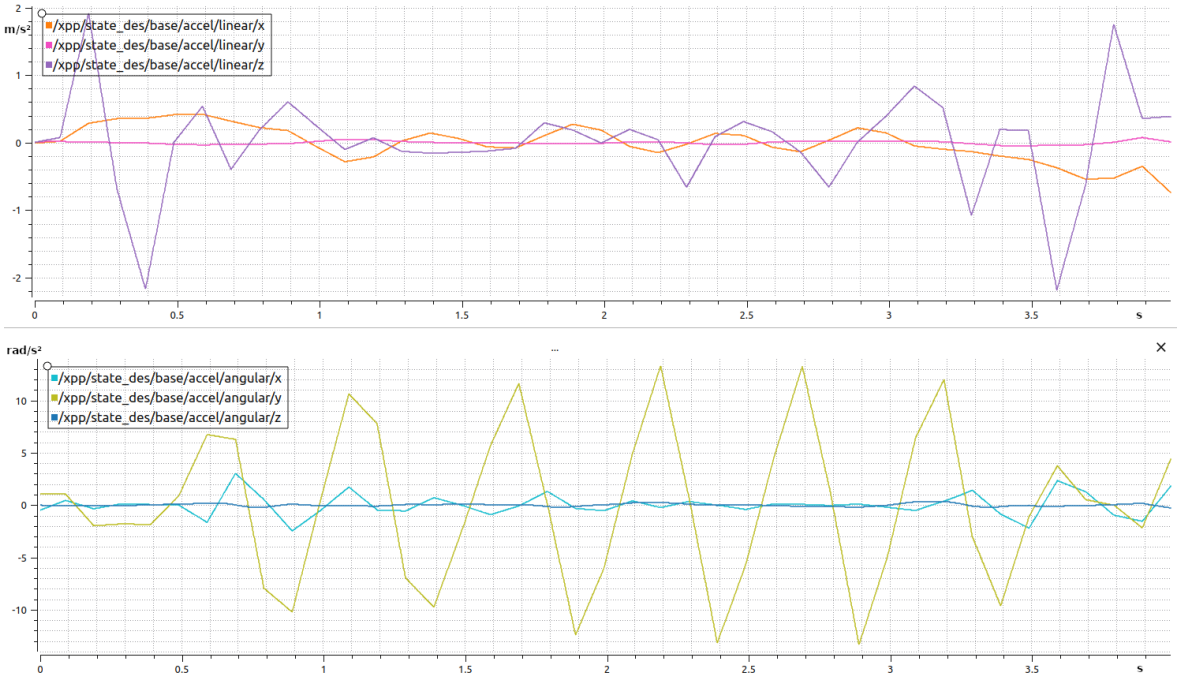


Figura 6.23: Visualización de las aceleraciones lineales y angulares tridimensionales del robot de la nueva trayectoria. En la parte superior se observa la aceleración lineal y en la inferior la angular.

Es importante destacar que, en la trayectoria nueva, las aceleraciones lineales son considerablemente menores en comparación con la anterior. Esto se debe a dos factores principales: el aumento del tiempo necesario para completar la trayectoria y la menor distancia que debe recorrer el robot.

En la nueva trayectoria, al aumentar el tiempo requerido para realizar el recorrido, el robot tiene más tiempo para acelerar y desacelerar de manera más suave y gradual lo que contribuye a reducir las aceleraciones lineales y mejorar la estabilidad del robot durante el movimiento. Además, al tener que cubrir una distancia más corta, el robot puede adaptar su velocidad a una menor siendo más eficiente y evitando aceleraciones grandes.

En cuanto a las aceleraciones angulares, en comparación con la trayectoria anterior, se observa una ligera reducción en Y, sin embargo en general, se mantienen constantes.

A continuación, se presentan las fuerzas tridimensionales que actúan en cada una de las patas del robot. De esta manera, se obtiene información sobre la distribución de la carga entre las patas en la nueva trayectoria.

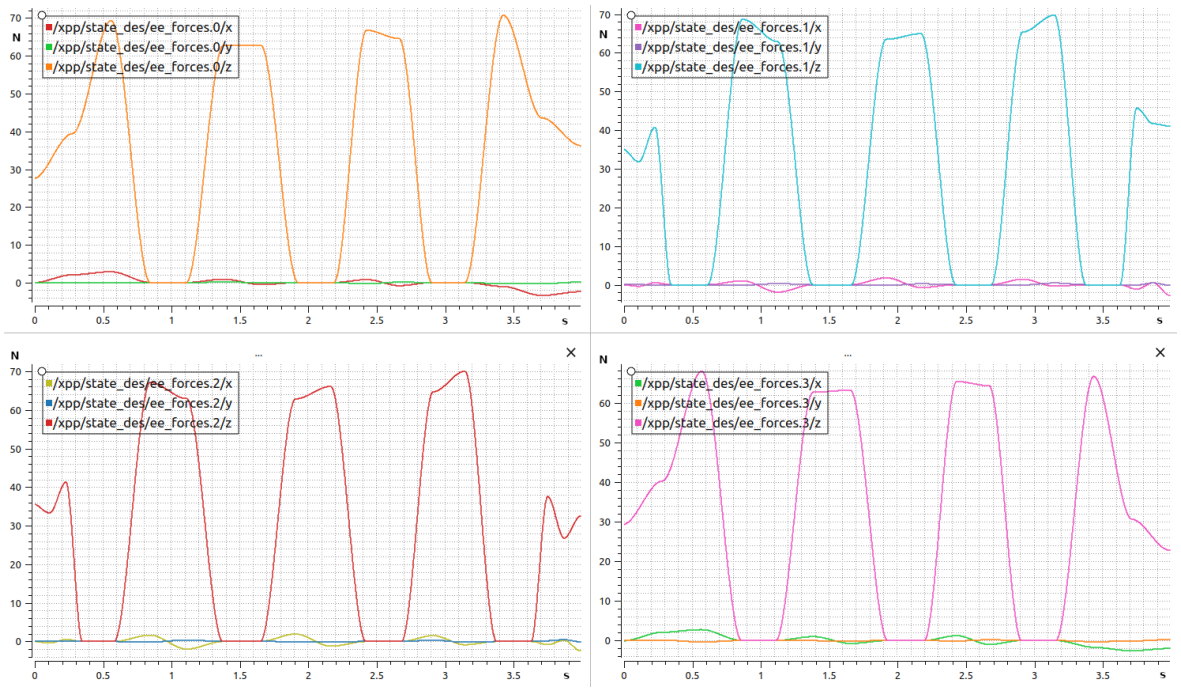


Figura 6.24: Visualización de las fuerzas tridimensionales de cada pata del robot en la nueva trayectoria. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha

En un primer vistazo se puede observar que las fuerzas cartesianas de cada pata son menores. Esto es debido a la disminución de las aceleraciones lineales en la nueva trayectoria. Al reducir las aceleraciones, se reduce la magnitud de las fuerzas necesarias para impulsar y desacelerar el cuerpo del robot durante el movimiento.

En la trayectoria anterior, donde las aceleraciones lineales son más altas, las fuerzas cartesianas requeridas para generar esas aceleraciones son mayores. Esto implica un mayor esfuerzo y potencia por parte de las articulaciones y actuadores de las patas del robot.

Sin embargo, en la nueva trayectoria con aceleraciones lineales menores, las fuerzas requeridas para lograr el movimiento son también menores reduciendo la carga en las articulaciones y actuadores de las patas y resultando en una operación más eficiente y con una menor tensión mecánica en el sistema.

Finalmente, se procede a mostrar las posiciones articulares del robot en cada instante de tiempo de la trayectoria nueva.

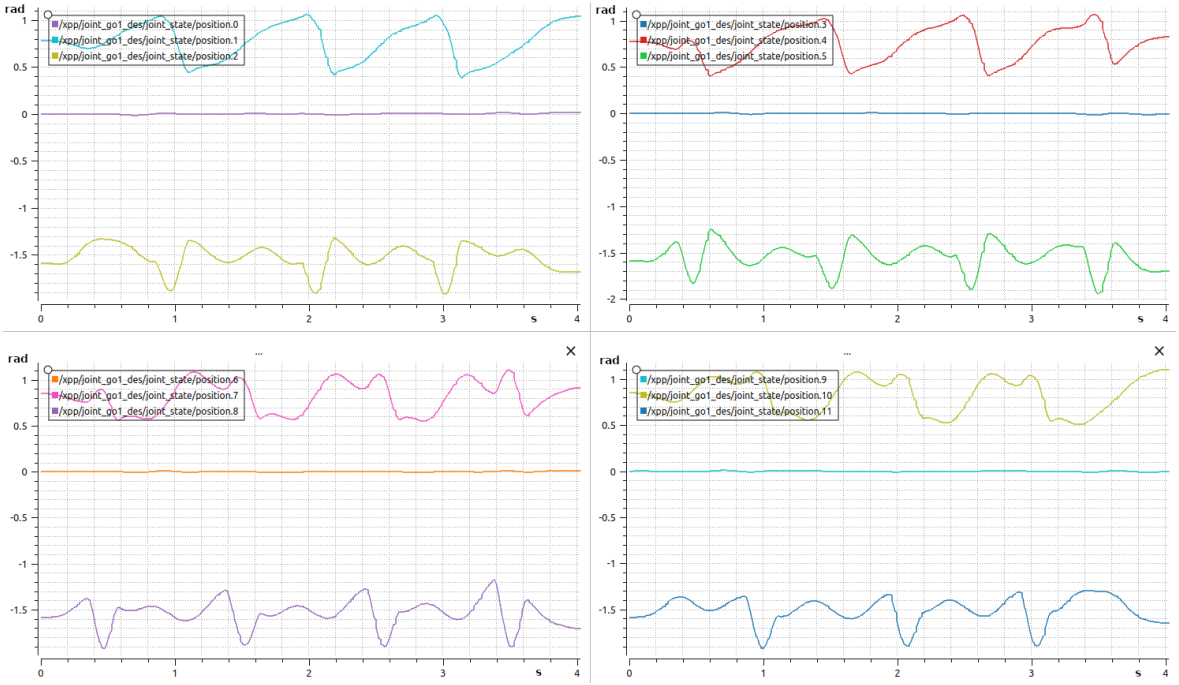


Figura 6.25: Visualización de la posiciones articulares de las articulaciones del robot en la nueva trayectoria. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

Al igual que en el resto de los datos, es evidente que existen diferencias en las posiciones articulares entre la nueva trayectoria y la anterior. Estas diferencias pueden atribuirse a dos causas principales: la generación de una trayectoria diferente y la limitación en las posiciones debido a la reducción del prisma.

En primer lugar, al tratarse de una trayectoria diferente, se esperan variaciones en las posiciones de las articulaciones del robot. Cada trayectoria tiene características únicas que determinan cómo se deben mover las articulaciones para realizar el movimiento deseado. Por lo tanto, es natural que las posiciones de las articulaciones difieran entre las dos trayectorias.

En segundo lugar, la reducción del prisma mencionada previamente, limita las posiciones alcanzables por las articulaciones del robot. Al acortar el prisma en el eje X y alargarlo en el eje Z, se restringe el rango de movimiento de las articulaciones afectando a las posiciones que pueden alcanzar las articulaciones y contribuyendo también a las diferencias observadas en las posiciones entre ambas trayectorias.

6.3.2 Seguimiento en simulación

Una vez obtenidos los datos de la planificación de la trayectoria, es importante verificar si el robot es capaz de seguir dicha trayectoria en simulación. Además, a medida que el robot se mueve, se registran los datos, como las posiciones de las articulaciones, los pares ejercidos y las aceleraciones realizadas.

En primer lugar, como en los apartados de gravedad, se muestran las aceleraciones lineales y angulares.

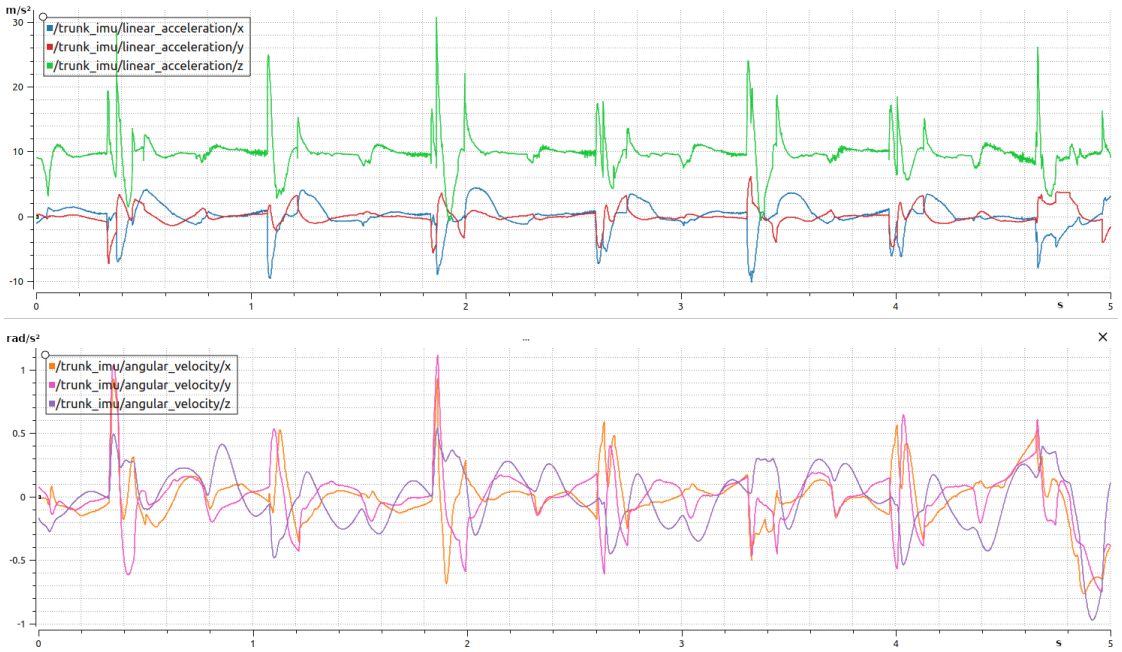


Figura 6.26: Visualización de las aceleraciones lineales y angulares tridimensionales del seguimiento en la nueva trayectoria. En la parte superior se observa la aceleración lineal y en la inferior la angular.

Tal y como ocurre en los casos de gravedad con las trayectorias anteriores, en las aceleraciones lineales se observan valores considerablemente superiores en comparación con los obtenidos durante la fase de planificación.

Además, los picos observables en las aceleraciones lineales durante el seguimiento coinciden con el momento en el que el robot apoya las patas al dar un paso. Este fenómeno se debe a la interacción entre las patas y el suelo, que puede generar vibraciones y agitaciones en el robot. Dichas agitaciones pueden ser causadas por varios factores, como la naturaleza del terreno, la rigidez de las articulaciones y los mecanismos de amortiguación del robot.

En contraste con las aceleraciones lineales, las angulares son considerablemente menores en magnitud. De hecho, los valores son aproximadamente diez veces menores en comparación con las aceleraciones angulares en la planificación, lo que implica una mayor capacidad para mantener el equilibrio y evitar oscilaciones excesivas durante la marcha.

El siguiente paso es mostrar los pares articulares de las articulaciones de cada una de las patas.

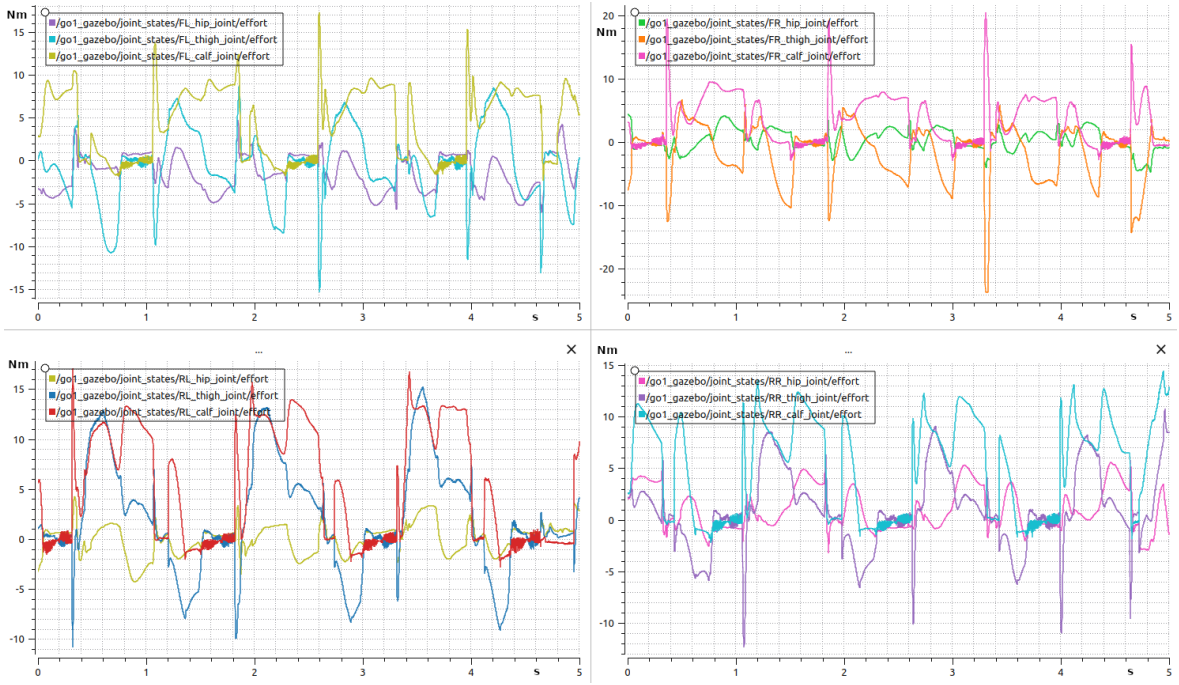


Figura 6.27: Visualización de los pares articulares de cada articulación en la nueva trayectoria. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

Al disminuir las fuerzas cartesianas en la planificación de la trayectoria nueva, es de esperar que los pares articulares también sean menores en comparación con la anterior. Al reducir estas fuerzas, se reduce la cantidad de esfuerzo requerido en las articulaciones para seguir la trayectoria deseada. Esto implica una disminución en los pares articulares, lo que a su vez como ya se ha comentado, puede contribuir a una mayor eficiencia y durabilidad de las articulaciones del robot.

Esto puede ser beneficioso en términos de reducir el desgaste y la fatiga en las articulaciones, lo que puede tener un impacto positivo en la vida útil y el rendimiento del robot a largo plazo.

No obstante, en la gráfica, se pueden apreciar claramente los momentos en los que las patas del robot entran en contacto con el suelo, lo cual se traduce en picos de pares. Estos picos representan las fuerzas y momentos generados en las articulaciones cuando el robot apoya sus patas.

Finalmente, se muestran las posiciones articulares de todas las articulaciones.

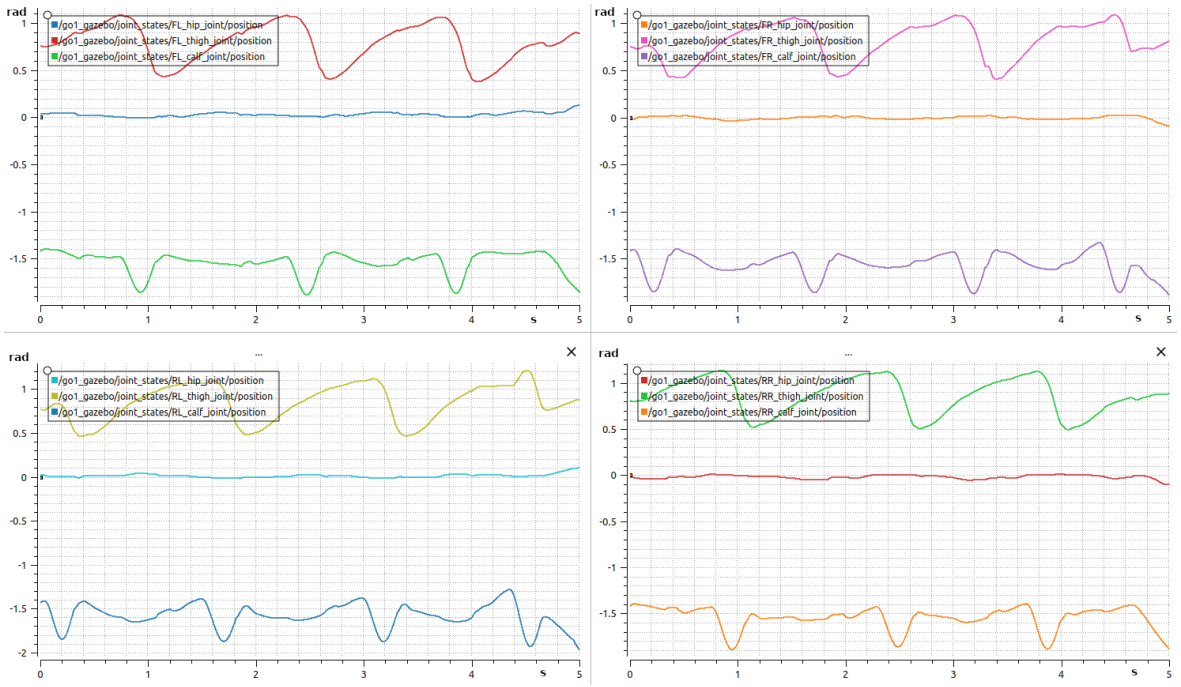


Figura 6.28: Visualización de la posiciones articulares en la nueva trayectoria. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

En comparación con las posiciones articulares obtenidas en la planificación, se observa una diferencia en las articulaciones de los muslos de las patas traseras durante el seguimiento. Es posible que esta discrepancia se deba al contacto con el suelo, el cual puede limitar el rango de movimiento y no permitir que las articulaciones alcancen la posición planificada.

Sin embargo, a pesar de las diferencias observadas, se logra un seguimiento exitoso de la trayectoria planificada, siendo el robot capaz de seguir la trayectoria en la simulación de manera precisa y consistente.

6.3.3 Seguimiento en el robot real

Una vez simulada la nueva trayectoria, se procede a verificar si el robot es capaz de realizarla y avanzar, a diferencia de la anterior. En el caso anterior, como se ha comentado, se han observado dificultades en el avance del robot, como problemas de estabilidad y riesgo de volteo debido a la alta velocidad de publicación y arrastre de las patas. Sin embargo, con la nueva trayectoria, se espera superar estas limitaciones y que el robot pueda moverse sin inconvenientes.

Al igual que en la simulación, durante la ejecución de la nueva trayectoria en el robot real, se monitorizan diferentes aspectos, como las aceleraciones, los pares de las patas y el movimiento de las articulaciones.

En primer lugar se tienen las aceleraciones.

Figura 6.29: Visualización de las aceleraciones lineales y angulares tridimensionales del robot real. En la parte superior se observa la aceleración lineal y en la inferior la angular.

Al observar la gráfica, se puede notar que las magnitudes de las aceleraciones lineales y angulares son similares a las obtenidas en la simulación. Esto indica que el robot está reproduciendo de manera precisa los movimientos planificados y está experimentando las mismas aceleraciones en el entorno real, sin embargo, los picos de aceleración son bastante menores.

Esto puede ocurrir por reducir la frecuencia de publicación para solucionar el primer problema, lo que limita la capacidad del robot para responder rápidamente a cambios en la trayectoria y se puede producir una suavización de los picos de aceleración. Además al reducir la frecuencia a la mitad, el tiempo que tarda el robot en realizar la trayectoria aumenta

a el doble (la trayectoria pasa de durar poco más de cuatro segundos a ocho).

A continuación, como en todos los casos, se muestran los pares articulares de cada pata.

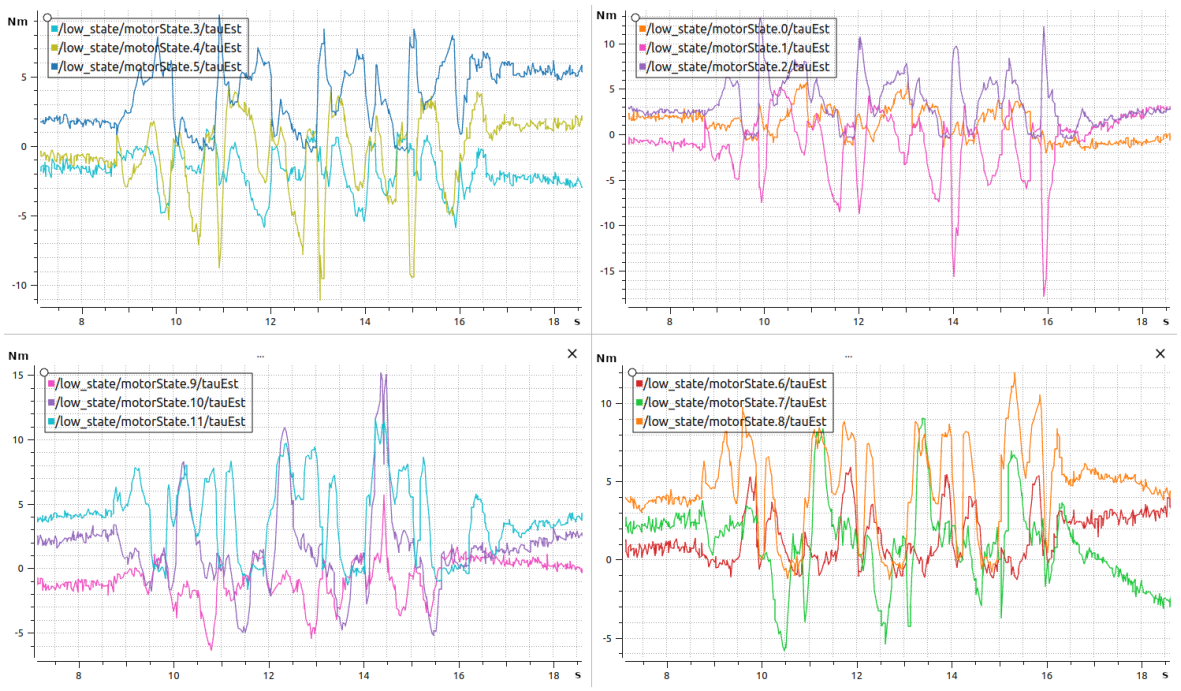


Figura 6.30: Visualización de los pares articulares de cada articulación en el robot real. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

Es interesante observar que, a pesar de la aparente diferencia inicial en las gráficas de pares articulares entre el robot real y la simulación debido al aumento del tiempo de la trayectoria, al analizar en mayor detalle ambas gráficas, se puede notar una similitud significativa en la forma de las curvas y en los valores de los pares articulares.

Sin embargo, esto no quiere decir que no existan diferencias, sugiere que, a pesar de la variación en el tiempo, el comportamiento general de las articulaciones se mantiene consistente en ambos casos. La forma similar de las curvas indica que el robot está siguiendo la misma secuencia de movimientos planificada en la simulación, aunque a una escala temporal diferente.

Además, la similitud en los valores de los pares articulares entre la simulación y el robot real indica que el esfuerzo requerido por las articulaciones para seguir la trayectoria es comparable en ambos casos, lo que resulta en una correcta elección de la frecuencia de publicación.

Finalmente se muestran las posiciones articulares de las articulaciones.

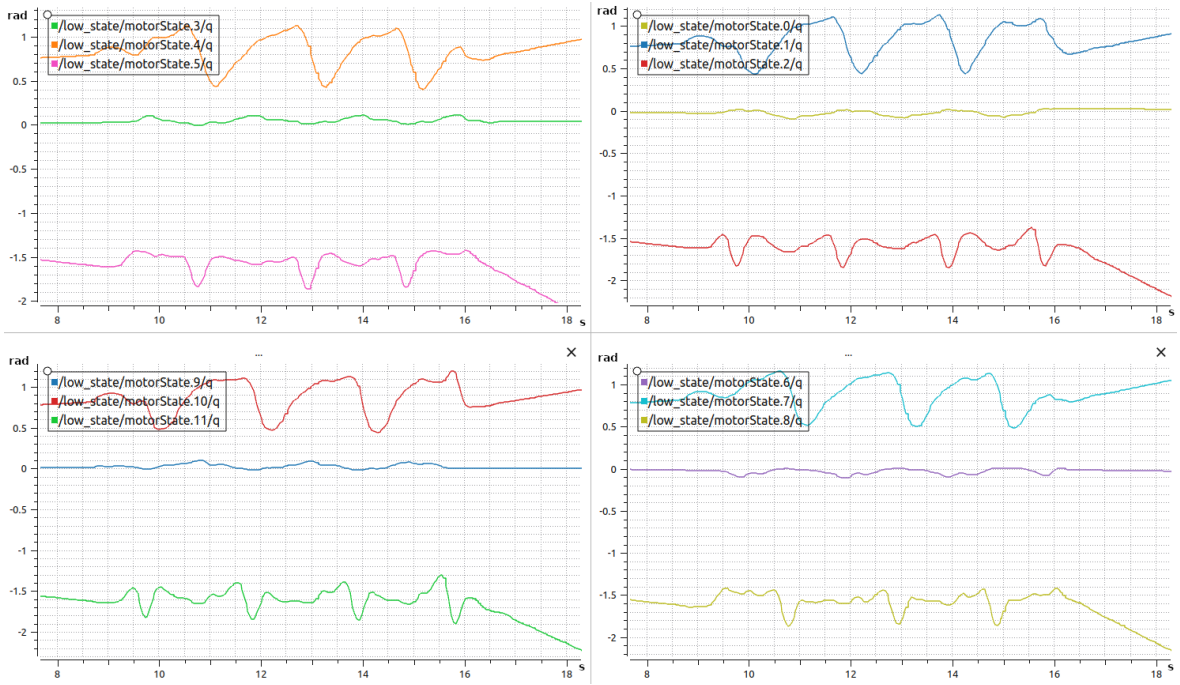


Figura 6.31: Visualización de la posiciones articulares en el robot real. De izquierda a derecha y de arriba a abajo: Pata delantera izquierda, pata delantera derecha, pata trasera izquierda y pata trasera derecha. En cada gráfica de arriba a abajo en la leyenda: posiciones de la cadera, muslo y tobillo

Al igual que ocurre en los pares articulares, es notable y positivo observar que las posiciones articulares obtenidas en el robot real coinciden con las posiciones obtenidas en la simulación. Esto indica que el robot es capaz de reproducir fielmente los movimientos planificados y seguir la trayectoria deseada.

Además, es interesante destacar que, en contraste con las trayectoria anterior, en esta el robot es capaz de moverse hacia adelante, lo cual es un avance positivo. Sin embargo, es importante señalar que los movimientos del robot se perciben como ligeramente torpes y que arrastra las patas más de lo que debería en comparación a la simulación.

A continuación se puede ver un vídeo con la planificación, simulación y ejecución en el robot real de la trayectoria:

`https://youtu.be/xwKWYA1Jfb8`

7 Conclusiones

A lo largo del presente trabajo se ha explorado el funcionamiento del *framework* de optimización de trayectorias conocido como *towr* y se han explicado las diferentes restricciones que establece para generar trayectorias lo más correctas y realistas posibles.

Además, se ha realizado un estudio de la cinemática del robot, tanto directa como inversa, lo que ha permitido comprender cómo se relacionan las posiciones y ángulos de las articulaciones con la posición y orientación del robot en el espacio. También, se ha analizado en detalle el modelo dinámico utilizado, entendiendo cómo se ven afectadas las fuerzas y aceleraciones en cada articulación a medida que el robot se mueve a lo largo de la trayectoria.

Tras ello, se ha explicado paso a paso como incluir el robot Go1 en *towr*. Una vez incluido, se ha utilizado para generar una misma trayectoria para tres condiciones de gravedad distintas: terrestre, marciana y una intermedia. A continuación, se ha explicado el proceso de como realizar el seguimiento de estas trayectorias mediante los paquetes de *Unitree Robotics*.

Una vez generadas y realizados los seguimientos de dichas trayectorias en *Gazebo* se han analizado los resultados llegando a las siguientes conclusiones:

- A medida que la gravedad disminuye, las aceleraciones, fuerzas cartesianas y pares articulares lo hacen con ella.
- Las aceleraciones obtenidas en la generación de trayectorias difieren de las obtenidas en el seguimiento, debido posiblemente a factores que *towr* no tiene en cuenta.
- Las posiciones cartesianas y articulares se mantienen independientemente de la gravedad.
- Se consigue un seguimiento consistente de la trayectoria en los tres escenarios.

Tras comprobar el impacto gravitatorio y las consecuencias que sufre el robot en cada caso, se ha llevado a cabo el seguimiento de la trayectoria en gravedad terrestre pero en el robot real. Sin embargo, al contrario de lo que ocurre en la simulación, el robot es incapaz de seguir la trayectoria revelando la existencia de tres problemas.

- La frecuencia de publicación era demasiado alta causando casi la caída del robot.
- El Go1 arrastraba los pies durante la ejecución.
- El número de valores que contenía el rosbag era insuficiente para tener una trayectoria suave y fluida.

Tras tomar las medidas necesarias para solucionar estos problemas, se han planificado muchas trayectorias hasta encontrar una aparentemente prometedora. Tras realizar su ejecución en el robot real se ha conseguido que el robot avance, sin embargo, a diferencia de lo que ocurre en la simulación su movimiento es ligeramente torpe.

Para finalizar, como trabajo futuro se plantea la realización de un controlador visual propio y control a alto nivel, así como la realización de diferentes pruebas en el robot real simulando diferentes gravedades en la realidad.

Bibliografía

Bartsch, S. (2014, 06). Development, control, and empirical evaluation of the six-legged robot spaceclimber designed for extraterrestrial crater exploration. *KI - Kunstliche Intelligenz*, *28*, 127-131. doi: 10.1007/s13218-014-0299-y

Boyd, S., y Vandenberghe, L. (2004). *Convex optimization.* Cambridge university press.

Byeonggi, Y., Junyoung, L., Sang hyun, P., y Maolin, J. (2023). A universal method for constructing dh parameters from unified robot description format. *The Journal of Korea Robotics Society*, *18*. Descargado de `https://jkros.org/_common/do.php?a=current&b=15&bidx=3204&aidx=35746` doi: 10.7746/jkros.2023.18.1.037

Carpentier, J., Saurel, G., Buondonno, G., Mirabel, J., Lamiraux, F., Stasse, O., y Mansard, N. (2019, enero). The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. En *SII 2019 - International Symposium on System Integrations.* Paris, France. Descargado de `https://hal.laas.fr/hal-01866228`

Coumans, E., y Bai, Y. (2016–2021). *Pybullet, a python module for physics simulation for games, robotics and machine learning.* `http://pybullet.org`.

Dirk, S., y Kirchner, F. (2007, 10). The bio-inspired scorpion robot: Design,control lessons learned.. doi: 10.5772/5081

Featherstone, R. (2007). *Rigid body dynamics algorithms.* Berlin, Heidelberg: Springer-Verlag.

Gill, P., Murray, W., y Saunders, M. (2002, 04). Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, *12*, 979-1006. doi: 10.2307/20453604

Guennebaud, G., Jacob, B., y cols. (2010). *Eigen v3.* http://eigen.tuxfamily.org.

Hutter, M., Gehring, C., Jud, D., Lauber, A., Bellicoso, C. D., Tsounis, V., … Hoepflinger, M. (2016). Anymal - a highly mobile and dynamic quadrupedal robot. En *2016 ieee/rsj international conference on intelligent robots and systems (iros)* (p. 38-44). doi: 10.1109/IROS.2016.7758092

Inoue, K., y Kaminogo, M. (2015, 05). Steep slope climbing using feet or shins for six-legged robots. En (p. 1-6). doi: 10.1109/ASCC.2015.7244563

Ivaldi, S., Padois, V., y Nori, F. (2014). *Tools for dynamics simulation of robots: a survey based on user feedback.*

Jiang, Z., Liu, J., Liu, Y., y Li, H. (2022). Advances in space robots. *Space: Science & Technology*, *2022*. Descargado de https://spj.science.org/doi/abs/10.34133/2022/9764036 doi: 10.34133/2022/9764036

Kam, H., Lee, S.-H., Park, T., y Kim, C.-H. (2015, 10). Rviz: a toolkit for real domain data visualization. *Telecommunication Systems*, *60*, 1-9. doi: 10.1007/s11235-015-0034-5

Kelly, M. (2017). An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, *59*(4), 849-904. Descargado de https://doi.org/10.1137/16M1062569 doi: 10.1137/16M1062569

Koenig, N., y Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. En *2004 ieee/rsj international conference on intelligent robots and systems (iros) (ieee cat. no.04ch37566)* (Vol. 3, p. 2149-2154 vol.3). doi: 10.1109/IROS.2004.1389727

Lee, J., Grey, M. X., Ha, S., Kunz, T., Jain, S., Ye, Y., … Liu, C. K. (2018, Feb). DART: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, *3*(22), 500. Descargado de https://doi.org/10.21105/joss.00500 doi: 10.21105/joss.00500

Ode. (s.f.). Descargado de https://www.ode.org/

Plotjuggler. (s.f.). Descargado de https://plotjuggler.io

Popi. (2021). Descargado de https://github.com/popi-mkx3/popi_project

Quadratic programming. (2006). En *Numerical optimization* (pp. 448–492). New York, NY: Springer New York. Descargado de https://doi.org/10.1007/978-0-387-40065-5_16 doi: 10.1007/978-0-387-40065-5_16

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., … Ng, A. (2009, 01). Ros: an open-source robot operating system. En (Vol. 3).

Remy, C., Baur, O., Latta, M., Lauber, A., Hutter, M., Hoepflinger, M., … Siegwart, R. (2011, 05). Walking and crawling with alof—a robot for autonomous locomotion on four legs. *Industrial Robot: An International Journal*, *38*, 264-268. doi: 10.1108/01439911111122761

Seeni, A., Schafer, B., Rebele, B., y Tolyarenko, N. (2008, 04). Robot mobility concepts for extraterrestrial surface exploration. En (p. 1 - 14). doi: 10.1109/AERO.2008.4526237

Semini, C., Tsagarakis, N. G., Guglielmino, E., Focchi, M., Cannella, F., y Caldwell, D. G. (2011). Design of hyq – a hydraulically and electrically actuated quadruped robot. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, *225*(6), 831-849. Descargado de https://doi.org/10.1177/0959651811402275 doi: 10.1177/0959651811402275

Simbody. (s.f.). Descargado de https://simtk.org/projects/simbody/

Unitree robotics github. (s.f.). Descargado de https://github.com/unitreerobotics

Winkler, A. W. (2017). *Xpp - A collection of ROS packages for the visualization of legged robots.* Descargado de `https://doi.org/10.5281/zenodo.1037901` doi: 10.5281/zenodo.1037901

Winkler, A. W. (2018). *Ifopt - A modern, light-weight, Eigen-based C++ interface to Nonlinear Programming solvers Ipopt and Snopt.* Descargado de `https://doi.org/10.5281/zenodo.1135046` doi: 10.5281/zenodo.1135046

Winkler, A. W., Bellicoso, D. C., Hutter, M., y Buchli, J. (2018, July). Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters (RA-L)*, *3*, 1560-1567. doi: 10.1109/LRA.2018.2798285

Wächter, A., y Biegler, L. (2006, 03). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, *106*, 25-57. doi: 10.1007/s10107-004-0559-y

Yoshida, K., Nenchev, D., Ishigami, G., y Tsumaki, Y. (2014). Space robotics. En M. Macdonald y V. Badescu (Eds.), *The international handbook of space technology* (pp. 541–573). Berlin, Heidelberg: Springer Berlin Heidelberg. Descargado de `https://doi.org/10.1007/978-3-642-41101-4_19` doi: 10.1007/978-3-642-41101-4_19

Lista de Acrónimos y Abreviaturas

CCRBA	Centroidal Composite Rigid Body Algorithm.
DART	Dynamic Animation and Robotics Toolkit.
DH	Denavit–Hartenberg.
IP	Internet Protocol.
NLP	Nonlinear Programming.
ODE	Open Dynamics Engine.
ROS	Robot Operating System.
TFG	Trabajo Final de Grado.
UDP	User Datagram Protocol.