UNIVERSITAT POLITÈCNICA DE VALÈNCIA

2021

Desarrollo de controladores modulares de posición/fuerza basados en ROS2 para robots paralelos de rehabilitación de miembro inferior

TRABAJO DE FIN DE MÁSTER EN AUTOMÁTICA E INFORMÁTICA INDUSTRIAL

AUTOR: JESÚS FERRÁNDIZ ALARCÓN

TUTORA: MARINA VALLÉS MIQUEL

RESUMEN

El objetivo de este trabajo es el desarrollo de controladores de posición y de fuerza para robots paralelos de rehabilitación de 3 y de 4 grados de libertad. En primer lugar, se ha seleccionado el software ROS2 debido a la programación modularidad que ofrece y a que ha sido diseñado para cumplir con los requerimientos de tiempo real. La modularidad ha permitido que una vez validados los controladores de posición, se hayan podido añadir de una manera sencilla más capas de control como un controlador para escapar de singularidades por medio de una realimentación de OptiTrack y controladores de fuerza. Posteriormente, se ha desarrollado una interfaz gráfica basada en una aplicación web y, finalmente, se han implementado controladores de posición basados en algoritmos de aprendizaje por refuerzo profundo.

RESUM

L'objectiu d'aquest treball és el desenvolupament de controladors de posició y de força per a robots paral·lels de rehabilitació de 3 y de 4 graus de llibertat. En primer lloc, s'ha seleccionat el software ROS2 per la programació modular que ofereix i perquè ha sigut dissenyat per tal de complir en els requeriments de temps real. La modularitat ha permès que una vegada validats els controladors de posició, s'hagin pogut afegir d'una manera senzilla més capes de control com un controlador per escapar d'una singularitat per mitjà d'una realimentació d'Optitrack y controladors de força. Posteriorment, s'ha desenvolupat una interfície gràfica basada en una aplicación web y, finalment, s'han implementat controladors de posició basats en algoritmes d'aprenentatge per reforç profund.

ABSTRACT

The objective of this project is the development of position and force controllers for rehabilitation parallel robots, which have 3 and 4 degrees of freedom. Firstly, the software ROS2 has been selected due to the modular programming it provides and because it has been designed to meet real time requirements. The modularity has made that once the position controllers have been validated, external control loops have been easily incorporated. These external control loops include a controller to escape from singularities thanks to feedback from OptiTrack and force controllers. After that, a web based graphical interface has been developed and finally, position controllers based on deep reinforcement learning algorithms have been implemented.

DOCUMENTO 1: MEMORIA

DOCUMENTO 2: PLIEGO DE CONDICIONES

DOCUMENTO 3: PRESUPUESTO

2021

Desarrollo de controladores modulares de posición/fuerza basados en ROS2 para robots paralelos de rehabilitación de miembro inferior

DOCUMENTO 1: MEMORIA

AUTOR: JESÚS FERRÁNDIZ ALARCÓN

TUTORA: MARINA VALLÉS MIQUEL

TABLA DE CONTENIDO

1. OBJETO

El presente trabajo de fin de máster tiene como objeto desarrollar una arquitectura de control para un robot paralelo de cuatro grados de libertad y otro de tres grados diseñados por el Instituto de Automática e Informática Industrial de la Universitat Politècnica de València. El desarrollo de la arquitectura incluye la selección del software, la implementación de los controladores, de una interfaz de usuario y de llevar a cabo experimentos con los robots.

1.1. INTRODUCCIÓN

Durante más de 50 años se han estado utilizando los robots industriales, generalmente enfocados a tareas simples repetitivas y con poca interacción con los seres humanos. Conforme la tecnología avanza, también lo hace la robótica de servicios, una robótica diferente a la industrial en el sentido de que se opera en un entorno menos estructurado, las tareas son más complicadas y sí que hay una interacción entre el robot y los seres humanos [1]. Esta interacción con los seres humanos trae consigo nuevos problemas que resolver en el sentido de seguridad, precisión y fuerza de los movimientos.

La robótica de servicios engloba todas aquellas aplicaciones de robótica que no tienen una finalidad industrial y que realizan un servicio a los seres humanos. Entre estas aplicaciones se encuentra la rehabilitación, la cual es el propósito de este proyecto. Una rutina de rehabilitación convencional entre un paciente y un médico generalmente conlleva realizar ejercicios repetitivos durante horas, de esta manera, se justifica el desarrollo de un robot de rehabilitación para miembro inferior. Este ayuda al doctor a realizar las tareas repetitivas que le conllevan tanto tiempo.

Esto supone resolver una serie de problemas relacionados con no solamente la interacción mecánica entre el paciente y el robot, si no también entre el médico y la máquina. La primera interacción supone primero resolver el problema del control de posición del robot de una manera muy precisa y sin vibraciones. Esta precisión justifica también el uso de un robot paralelo, su estructura mecánica hace que se minimicen los errores en las articulaciones y tienen una relación fuerza-peso más alta. Una vez resuelto el problema de posición, la colaboración estrecha con el paciente exige también una realimentación de la fuerza que el robot ejerce sobre este. El control también de la fuerza hace que la interacción sea más segura y, además, permite implementar ejercicios de rehabilitación resistivos.

Otro problema por resolver es la cantidad de parámetros inciertos que conlleva la interacción entre un robot y un ser humano, es en este punto donde se pueden aplicar técnicas de control adaptativo o las nuevas técnicas de inteligencia artificial. Por otro lado, se ha mencionado también una relación entre el médico y el robot. Esto exige la implementación de una interfaz entre ambos, diferente a la de los robots industriales

debido a que estas están más pensadas para trabajadores con una carrera en el ámbito de la ingeniería.

En este proyecto se ha trabajado con hasta tres robots paralelos diferentes. El hecho de desarrollar software para tres robots diferentes también implica el desarrollar una arquitectura de control modular, para que la integración del software y la interacción con los diferentes robots sea lo más rápida y sencilla posible. En la siguiente imagen se pueden observar los robots con los que se ha trabajado:

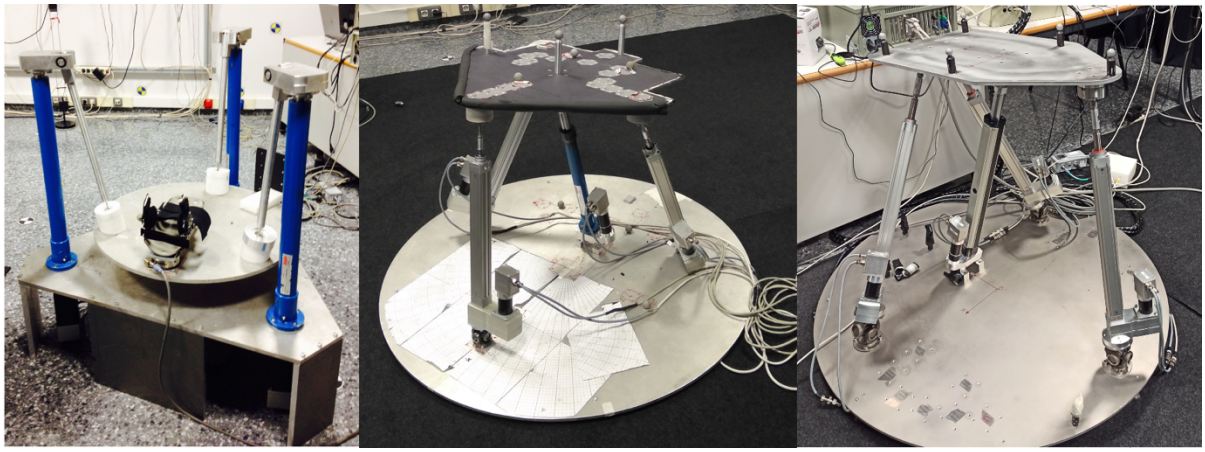


Figura 1: Robots paralelos desarrollados en el laboratorio de robótica de la Universitat Politècnica de València.

En la figura, los robots están ordenados en orden cronológico de su desarrollo. El primero tiene 3 grados de libertad, entonces está ideado para la rehabilitación de tobillo. Los otros dos tienen 4 grados de libertad y además de rehabilitar tobillo, también pueden ejercer trayectorias de rehabilitación para rodilla. El último robot es una versión nueva desarrollada en el año 2021 que mejora al robot anterior.

1.2. FINALIDAD Y OBJETIVOS DEL PROYECTO

La finalidad de este proyecto de fin de máster consiste en la implementación de una arquitectura de control abierta para robots de rehabilitación de miembros inferiores. Para ello, se establecen una serie de objetivos:

- Seleccionar un software que cumpla con los requerimientos de control.
- Diseñar la arquitectura de control de manera modular.
- Implementar controladores de posición.
- Una vez validados los controladores de posición, desarrollar controladores de fuerza.
- Desarrollar una interfaz de usuario.
- Llevar a cabo experimentos con los robots.
- Explorar controladores alternativos al control clásico, como las técnicas de inteligencia artificial.

2. ANTECEDENTES Y JUSTIFICACIÓN

2.1. JUSTIFICACIÓN DEL PROYECTO

Los robots paralelos desarrollados en el laboratorio de robótica de la Universitat Politècnica de Valencia ya tienen una arquitectura de control implementada. Sin embargo, esta arquitectura está basada en un software que ya ha dejado de tener soporte y es difícil de integrar en nuevas versiones de librerías, sistemas operativos, etc.

Por este motivo, es necesario desarrollar una nueva arquitectura de control basada en un software más reciente, que cumpla con los requisitos técnicos establecidos y que tenga soporte. A partir de esta arquitectura se deben poder construir aplicaciones como una interfaz gráfica de usuario, entonces tiene que tener una gran modularidad, que además permita trabajar con los distintos robots paralelos de una manera sencilla.

2.2. MARCO TEÓRICO

2.2.1. INTRODUCCIÓN A LA ROBÓTICA

Un robot puede ser definido como un mecanismo programable a lo largo de dos o más ejes, con el fin de desplazarse en el entorno para ejecutar diferentes tareas, como pueden ser la manipulación de objetos, navegar, o interactuar con las personas. Independientemente del objetivo, todos constan de mecanismos con cuerpos rígidos conectados, de sensores, actuadores, procesadores e intercambio de información.

Atendiendo los mecanismos que forman los robots, incluyendo los cuerpos rígidos y la manera en que estos están unidos por las articulaciones, se pueden distinguir dos tipos de cadenas cinemáticas. En primer lugar, se encuentran las cadenas cinemáticas simples o abiertas. Se pueden definir como aquellas en las que cada cuerpo tiene un grado de conexión menor o igual a 2, excepto para el efector final y la base que tienen un único grado. Estas cadenas son la base de los brazos robóticos, mecanismos que están sujetos a una base y que suelen incluir una herramienta en el efector final.

Por otro lado, se encuentran las cadenas cinemáticas cerradas. Estas se dan cuando uno de los cuerpos rígidos (distinto de la base) tiene un grado de conexión mayor o igual a 3 [2]. Estas cadenas son la base de los robots paralelos, que están formados por una plataforma móvil sujeta a una plataforma rígida a través de cuerpos rígidos y articulaciones activas y/o pasivas.

2.2.2. ROBOTS SERIE

Como se ha mencionado antes, los robots serie están basados en cadenas cinemáticas abiertas. Se tratan de mecanismos que están sujetos a una base fija y que en su efector suelen incorporar una herramienta para realizar una tarea de manipulación. Estos robots son los que más están presentes en la industria. Los cuerpos rígidos de estos robots están conectados en serie a través de articulaciones activas, esto es, que se puede actuar sobre ellas por medio de motores. Estas articulaciones pueden ser de revolución o prismáticas. Los robots serie imitan de cierta forma el brazo humano alcanzando una gran destreza gracias también a las últimas articulaciones que forman una muñeca. También pueden abarcar un gran espacio de trabajo.

Sin embargo, debido a las conexiones en serie de los mecanismos, también se transmiten con más facilidad los errores de una articulación a otra amplificándose. También tienen una baja rigidez y bajas características dinámicas en el sentido de velocidades y aceleraciones.

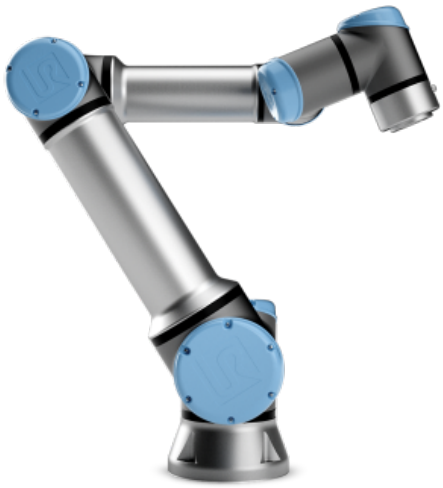


Figura 2: Robot serie. Fuente: https://www.universal-robots.com/fr/produits/

2.2.3. ROBOTS PARALELOS

Los robots paralelos, en cambio, están formados por cadenas cinemáticas cerradas. Uno de los cuerpos que lo forman es fijo y se denomina la base. En la base se conectan articulaciones y cuerpos con el fin de poder desplazar el efector final que en este caso se refiere como la plataforma móvil. Diversas patas y articulaciones de diferente naturaleza como esféricas, universales, cilíndricas, de revolución o prismáticas conectan la base con la plataforma móvil y formando cadenas cinemáticas cerradas.

Las articulaciones que los forman no tienen por qué estar actuadas, entonces se puede diferenciar entre articulaciones acticas y articulaciones pasivas. La ventaja de esta configuración es que los robots paralelos son mucho más precisos que los robots serie, los errores no se amplifican tanto debido a que no hay tantas articulaciones en serie. Y

un error de una articulación, puede compensarse con otra debido a la naturaleza paralela. Además, tienen una gran rigidez y una relación fuerza-peso muy alto. Entonces se suelen utilizar en aplicaciones en las que se requiere una gran precisión y en las que se maneja una carga pesada.

Como desventaja, los robots paralelos tienen un espacio de trabajo pequeño y más restringido que los robots serie. Los mecanismos que lo forman también son más complicados de estudiar matemáticamente y presentan más singularidades que los robots serie[3].

Figura 3: Robot paralelo. Fuente: https://pentarobotics.com/products/

2.2.4. PROGRAMACIÓN MODULAR EN ROBOTS

Los robots están formados por múltiples articulaciones activas, cada una con un actuador y un sensor. Entonces, en términos de software, es razonable realizar un enfoque modular de la programación. Las ventajas que conlleva son la posibilidad de reutilizar el código, por ejemplo, se puede implementar el código para acceder a una salida analógica para controlar un actuador. Este mismo código se puede utilizar entonces para un segundo actuador. La modularidad también permite, cambiar con facilidad a otro robot distinto pero de la misma naturaleza. Permite también encapsular operaciones, evitando errores, y la posibilidad de construir aplicaciones complejas de alto nivel a partir de controladores modulares de bajo nivel.

Para alcanzar esta modularidad, es necesario el uso de un framework que haga posible el intercambio de información de una forma segura entre los componentes que forman el software del robot. Actualmente, alguno de estos frameworks son OROCOS, ROS y ROS2.

2.2.5. CONTROL POR COMPUTADOR Y SISTEMAS DE TIEMPO REAL

La programación

Para alcanzar movimientos fiables y precisos en robótica, es necesario desarrollar un sistema de control basado en computador. Este sistema es realimentado por variables significativas a partir de sensores, entonces, tanto para actuar como para medir la respuesta, es necesario acceder al mundo físico, de naturaleza continua. Sin embargo, el control establecido por el computador es de naturaleza discreta, debido a que las operaciones que realizan son discretas y a la capacidad de cómputo, un tiempo de muestreo infinitamente pequeño no es posible.

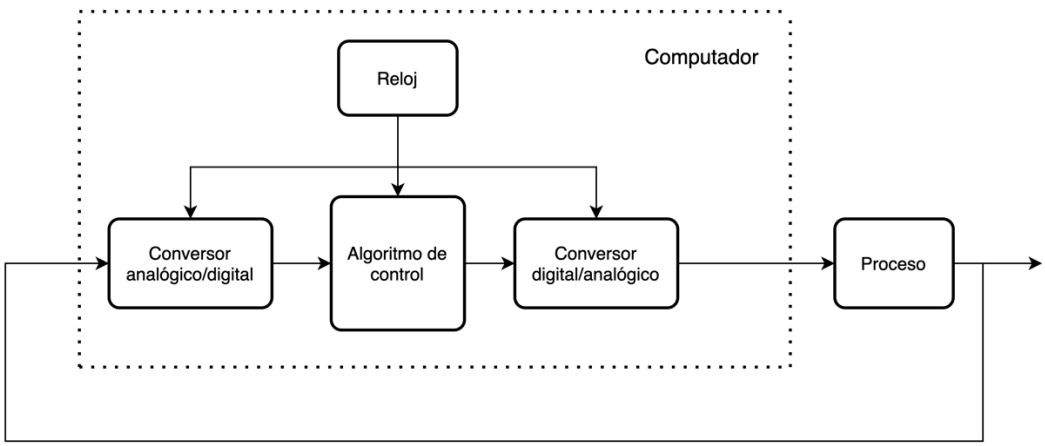


Figura 4: Sistema de control por computador

Estos sistemas de control por computador están íntimamente relacionados con los sistemas de tiempo real. Los sistemas de tiempo real tienen muchas definiciones, se pueden entender como cualquier sistema en el que el tiempo en el que se produce la respuesta es significativo, normalmente debido a que estos sistemas están relacionados con el movimiento de algún cuerpo físico. También se puede añadir que el tiempo de la respuesta sea finito y especificado. Este tiempo en un bucle de control en lazo cerrado se denomina tiempo de muestreo. Tiene que ser lo suficientemente pequeño como para que en el muestreo no se pierda información dinámica importante.

Otra cuestión que destaca en estos sistemas es que un fallo del cumplimiento del tiempo de respuesta establecido puede ser considerado y dar lugar a una respuesta incorrecta. Entonces, no solamente es crucial el cómputo correcto de la información, si no también que la respuesta se obtenga en el momento indicado. Se pueden distinguir entre los sistemas de tiempo real estrictos y los no estrictos. En los primeros, es absolutamente necesario que la salida se dé en el momento indicado y en los segundos esta condición es más relajada. En función del proceso físico que se esté controlando, se deberá aplicar un sistema estricto o no. No es lo mismo el control de un avión que un videojuego, en ambos se necesita una respuesta en un tiempo determinado, sin embargo, un fallo en el caso del avión puede ser catastrófico.

Los sistemas de tiempo real se caracterizan por ser grandes y complejos, pero se suelen dividir en componentes modulares más pequeños gracias a frameworks de programación. Estos sistemas deben ser capaces también de manipular números reales, los algoritmos de control que se implementan en el computador son generalmente matemáticamente complejos. Ya se ha mencionado que un fallo en estos puede traer graves consecuencias, entonces tienen que ser también fiables y seguros. El sistema de control, al estar formado por múltiples componentes, debe tener un computador y un software que garantice operaciones concurrentes de una manera segura [4].

2.2.6. ROS, ROS2

2.2.6.1. INTRODUCCIÓN

ROS (Robot Operating System) es un middleware de código abierto para desarrollar software para robots. Este middleware está formado por una serie de librerías y de paquetes que parten de una base de programación concurrente hasta llegar a herramientas de alto nivel como planificadores de trayectorias, visualizaciones, navegación, control, etc.

Uno de los objetivos principales de ROS es poder realizar una programación modular para robots, es decir, que sea sencillo reutilizar código y poder utilizar diversos lenguajes de programación a la vez, como por ejemplo C++ y Python. Para estos lenguajes, ROS proporciona las librerías cliente roscpp y rospy que funcionan como una interfaz de programación para abstraer la capa de transmisión de datos entre programas.

ROS ha sido ampliamente utilizado en la investigación y cada vez más en la industria. Con el fin de orientarse más a la industria y a las aplicaciones críticas, ha surgido recientemente ROS2 (en 2017 se lanzó la primera distribución). Este framework le añade y mejora a ROS los aspectos de tiempo real como la seguridad, el determinismo o el control del tiempo de latencias entre cadenas, desde que se recoge una muestra con sensores, hasta que se aplica una acción de control.

2.2.6.2. CARACTERÍSTICAS TÉCNICAS

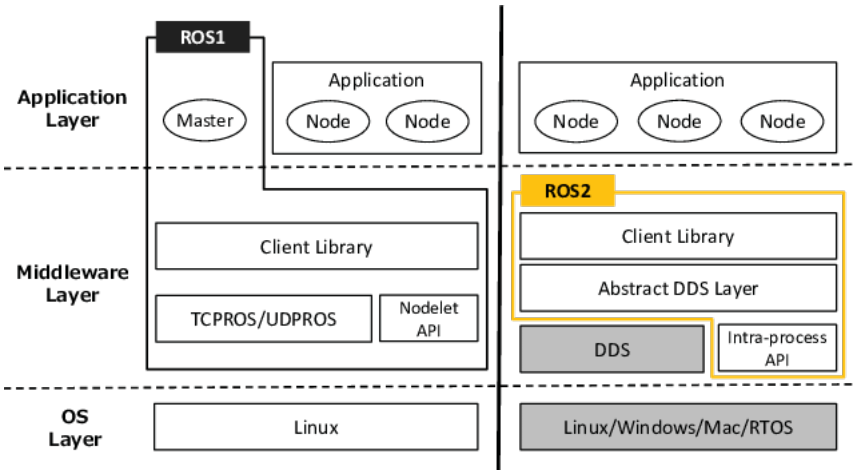


Figura 5: Comparación entre capas de ROS y de ROS2. Fuente: https://www.researchgate.net/figure/ROS1-ROS2-architecture-for-DDS-approach-to-ROS-We-clarify-the-performance-of-the-data_fig1_309128426

ROS2 puede ser ahora utilizado no solo en sistemas operativos Linux, si no también en Windows, Mac y RTOS. Sobre el sistema operativo, ROS2 incorpora una capa de distribución de mensajes basada en DDS (Data Distribution Service) que proporciona un transporte de datos similar a un publicador y a un subscriptor. Una de las ventajas que añade el uso de DDS es que permite que dos programas se comuniquen sin la necesidad de incorporar un programa maestro como en el caso de ROS en el que necesitaba un Master para funcionar. Esto se debe a que el sistema de descubrimiento proporcionado por DDS por defecto, es un sistema de descubrimiento distribuido. ROS2, por tanto, se puede utilizar de una manera sencilla y flexible en un sistema distribuido con diferentes máquinas. Además, se ha desarrollado una interfaz de programación para realizar una comunicación entre procesos más eficiente.

Posteriormente, se encuentra una librería cliente (rcl) implementada en lenguaje C a partir de la cual se han implementado otras librerías de más alto nivel y de múltiples lenguajes de programación con la que se pueden hacer aplicaciones en ROS2. Las librerías más utilizadas son rclcpp para C++ y rclpy para Python.

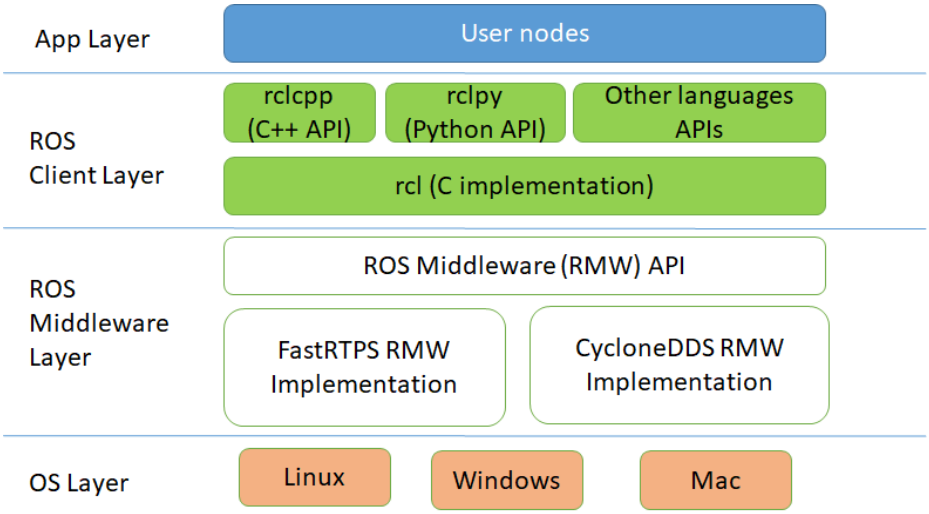


Figura 6: Capas de ROS2. Fuente: https://aws.amazon.com/es/blogs/robotics/ros2-foxy-fitzroy-robot-development/

En cuanto al nivel de aplicación de ROS2, en primer lugar, cada programa que realice una función como, por ejemplo, leer un encoder o detectar un objeto de una imagen, se denomina Nodo. Así pues, una aplicación de ROS2 se basa en nodos que se comunican entre ellos.

ROS2 proporciona tres maneras de comunicar los nodos, siendo la más importante los Topics. Un topic es un medio por el cual los nodos se pueden comunicar a través de mensajes. Un nodo puede publicar por un topic de forma que envía mensajes y también puede subscribirse a otro para recibir. Los mensajes son estructuras que almacenan datos, por ejemplo, un mensaje puede contener tres datos de tipo float que corresponden a las coordenadas x, y, z de un punto.

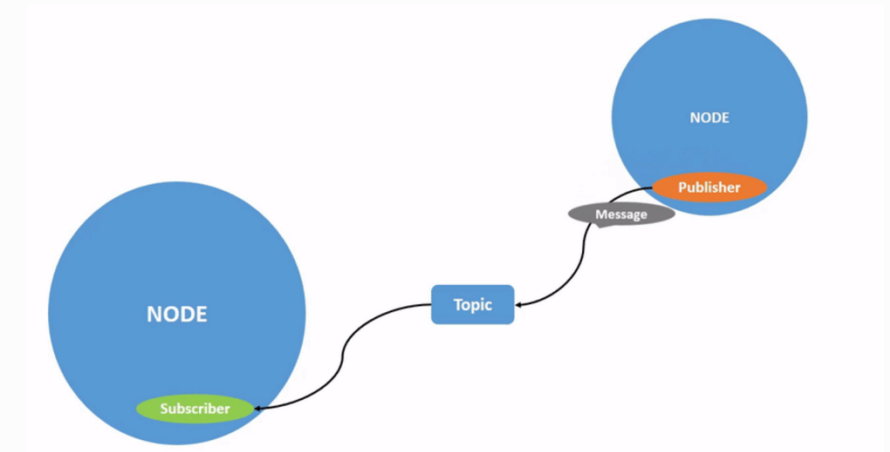


Figura 7: Comunicación publicador-subscriptor a través de un topic.
Fuente:https://docs.ros.org/en/foxy/Tutorials/Topics/Understanding-ROS2-Topics.html

Otro sistema de comunicación de ROS2 son los servicios. Estos se basan en una comunicación entre un cliente que envía una petición a un servidor y este devuelve una respuesta.

Por último, se encuentran las acciones, que están construidas a partir de los topics y de los servicios. Las acciones son similares a los servicios ya que también están basadas en un modelo cliente-servidor, pero están orientadas a tareas de gran duración y están formadas por tres partes: un objetivo, un resultado y una realimentación.

Otro aspecto importante de ROS2 son los parámetros. Estos son valores de configuración de un nodo que dotan a este de una mayor modularidad. Los parámetros, en función de como se hayan programado, pueden ser modificados a la hora de ejecutar un nodo o también en cualquier momento de la ejecución.

En cuanto a nivel de ejecución, los nodos que intercambian información instancian una callback, de manera que cuando un nodo que está suscrito a un topic recibe un mensaje nuevo, se ejecuta la callback asociado a dicha suscripción. También hay callbacks asociadas a temporizadores y a la comunicación entre un servidor y un cliente. El modelo de ejecución de ROS2 está basado en lanzar ejecutores. Los ejecutores reciben mensajes desde rcl y activan las callbacks asociadas. Hay dos tipos de ejecutores desarrollados en ROS2, uno secuencial que ejecuta un solo hilo de ejecución, y otro paralelo que maneja múltiples callbacks de manera paralelas en varios hilos.

Las callbacks de ROS2 tienen diferentes niveles de prioridad en función de su naturaleza. El ejecutor, primero comprueba si un temporizador ha expirado. Después comprueba las colas de mensajes asociadas a las callbacks de suscripciones, servicios y de clientes, en dicho orden [5].

Una nueva característica de ROS2 es la posibilidad de crear componentes. Un componente es una subclase de rclcpp::Node que se construye para dar lugar a una librería compartida [6]. La ventaja de desarrollar componentes es que estos se pueden componer en un único proceso, de manera que se puede construir una aplicación de ROS2 formada por varios componentes, y, por tanto, de varios nodos que se comunican entre sí en un único proceso. La ventaja de esto es que la comunicación entre procesos es más eficiente.

Así pues, se puede implementar un programa de C++, que instancie cada componente y un ejecutor secuencial o paralelo como los que se ha mencionado anteriormente que serán los encargados de gestionar la ejecución de las callbacks. ROS2 proporciona también un programa “component_container” que, por medio de llamadas a servicios, puede recoger en tiempo de ejecución un componente, instanciarlo y ejecutarlo.

Una vez explicadas estas características técnicas, falta resolver como ejecutar una aplicación o varias de ROS2 de una manera sencilla. Para ello, ROS2 ofrece la posibilidad de desarrollar scripts de Python que ejecutan todos los nodos o componentes necesarios para la aplicación, de forma que lo único que se necesita es llamar al script de lanzamiento.

2.2.7. TÉCNICAS DE INTELIGENCIA ARTIFICIAL EN ROBÓTICA

La inteligencia artificial suele estar asociada a un programa que aprende de manera automática. Esto da lugar a múltiples aplicaciones en robótica. Puede ser aplicado para que un programa aprenda a realizar tareas difíciles de programar por un ser humano, como puede ser resolver un cubo de Rubik con una mano robótica [7]. Esta mano tiene tantas articulaciones y los movimientos que realizamos los seres humanos son tan complicados a la vez que inconscientes, que son difícilmente programables.

El aprendizaje también puede ayudar construir de manera automática un modelo a partir de una gran cantidad de datos de entrada y que da lugar a unas salidas con una cierta precisión. En estos modelos es importante que no solamente acierte los datos con los que ha sido entrenado, si no que debe acertar datos totalmente nuevos, es decir, debe poder generalizar. Los modelos basados en aprendizaje automático se están aplicando en robótica en campos relacionados con la visión artificial para clasificar objetos de una imagen, en control para construir un modelo del proceso de manera automática más rápidamente que a partir de fundamentos teóricos y también para realizar tareas en las que es necesario adaptarse a un entorno cuyos parámetros son inciertos.

2.2.7.1. REDES NEURONALES PROFUNDAS

Muchos de los modelos de aprendizaje automático están basados en redes neuronales profundas. La unidad mínima de estas es la neurona, y se denominan redes profundas cuando se ha construido una red formada por varias capas de neuronas artificiales. La neurona artificial realiza una operación matemática sencilla, se suman las entradas multiplicadas por unos coeficientes, se añade un umbral y el resultado obtenido se introduce en una función de activación. La complejidad y también la capacidad de construir modelos no lineales se obtiene cuando se forma una capa de más de una neurona y se crea un modelo con múltiples capas.

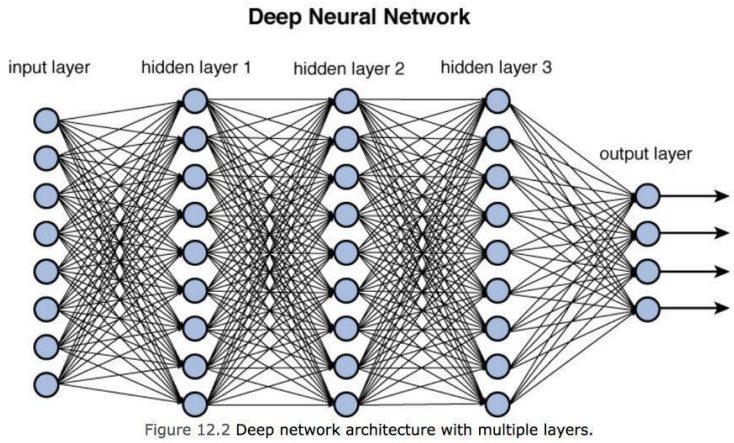


Figura 8: Esquema de una red neuronal profunda. Fuente: https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Las redes neuronales fueron propuestas teóricamente hace más de 50 años, sin embargo, entonces no se disponía de la capacidad de cómputo ni de la gran cantidad de datos que hay disponible en la actualidad gracias a Internet para que los modelos basados en redes neuronales tuvieran una aplicación práctica.

En la actualizad se han superado estos problemas, el desarrollo de las GPUs (Unidades de Procesamiento Gráfico) permite el aprendizaje de una red neuronal en un tiempo razonable, en función principalmente de la cantidad de parámetros que tenga el modelo y de la cantidad de datos de entrenamiento y validación. Además, Internet es una fuente de datos para estos modelos.

A la hora de entrenar los modelos basados en redes neuronales (o de cualquier otro modelo basado en aprendizaje automático), es necesario evitar que este se sobreajuste a los datos de entrenamiento. Es decir, que acierte una gran cantidad de datos de entrenamiento, pero que falle en las predicciones de datos nuevos, esto es, que no haya generalizado bien. Esto puede suceder cuando se han utilizado más parámetros de los necesarios para modelar un sistema.

En aplicaciones de control de procesos, un sobreajuste puede dar lugar a que, en vez de modelar el proceso, se modele el ruido de los sensores y que la salida amplifique este ruido.

2.2.7.2. APRENDIZAJE POR REFUERZO

El proceso de entrenamiento a partir de los datos puede ser distinto en función de la naturaleza del algoritmo. Hay algoritmos que se entrenan una vez se han recogido todos los datos de entrenamiento y de validación, pero esto no tiene por qué ser así. Otros algoritmos aprenden a partir de simulaciones o experimentos, es decir, de la experiencia, los datos los obtiene de cada episodio y modifica su comportamiento. Esta es la base del aprendizaje por refuerzo, una técnica de inteligencia artificial en la que un

algoritmo aprende a partir de la experiencia de múltiples episodios. El principio básico de funcionamiento se puede ver en la siguiente figura:

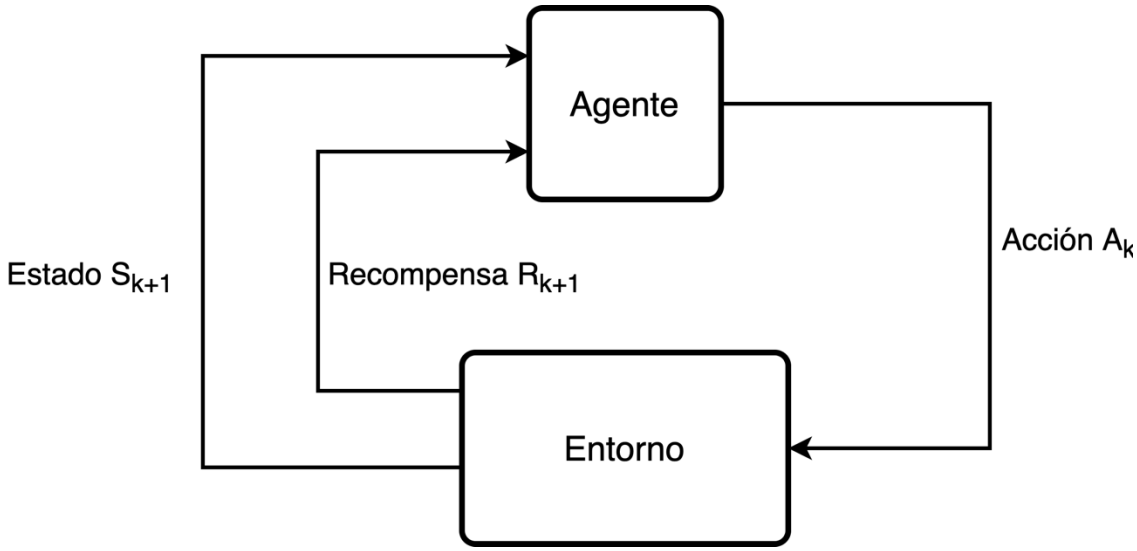


Figura 9: Diagrama de funcionamiento básico de un algoritmo de aprendizaje por refuerzo.

Un agente (el algoritmo de aprendizaje), emite una acción a un entorno (real o simulado), el entorno, entonces, devuelve un nuevo estado y una nueva recompensa. El objetivo del agente es maximizar la recompensa. A través de múltiples simulaciones, el agente modifica y actualiza su comportamiento de manera que optimiza la recompensa. El comportamiento define entonces, la relación que hay entre los estados y las acciones.

Algo importante para tener en cuenta es que una acción puede tener un retraso en una recompensa. Es decir, una acción puede dar lugar a una mala recompensa en el siguiente instante, pero a una mucho mejor recompensa en el futuro. Entonces, los algoritmos de aprendizaje por refuerzo también tienen que solucionar este problema. Otro problema es la decisión entre explotar o explorar. Explotar dará lugar a una buena recompensa, pero si un agente no descubre todos los estados de un entorno, no obtendrá la mejor recompensa. Sin embargo, el hecho de explorar dará lugar con mayor probabilidad a peores recompensas.

Como elementos o conceptos importantes del aprendizaje por refuerzo se tienen los siguientes:

- **Policy**: Se corresponde con el comportamiento del agente, concretamente con la relación (o mapeo) entre estados y la probabilidad de escoger cierta acción.
- **Señal de recompensa**: Se trata de un número escalar que se utiliza para modificar el comportamiento del agente. El objetivo del algoritmo es maximizar la cantidad de recompensa obtenida a lo largo de un episodio, no el valor del instante posterior. Para resolver este problema, existe un coeficiente (tasa de descuento) que determina el valor de las futuras

recompensas. El diseño de la función de recompensa no es sencillo, el agente puede aprender a maximizar la recompensa, pero esto no significa que haya alcanzado el objetivo real.

- **Función de valor**: Hay varios tipos de funciones de valor. Las funciones de valor del estado estiman cómo de bueno es para el agente encontrarse en un determinado estado. O también funciones de valor estado-acción, que estiman cómo de bueno es ejecutar una acción determinada en un estado determinado. Las funciones de valor, necesitan también de la policy para poder estimar los valores. Estas funciones se aprenden a base de la experiencia [8].
- **Proceso de decisión de Markov**: Se trata de una manera de modelar los entornos. Lo que se observa de ellos es el estado, para que un proceso tenga la propiedad de Markov, el siguiente estado, únicamente puede depender del estado actual, no del histórico de estados anteriores. Otro elemento son las transiciones de un estado a otro, estas dependen de la dinámica del entorno (probabilística definida por una matriz de transición) y de la acción escogida por el agente. Por último, en cada transición se añade una recompensa.

En robótica, lo más probable es encontrarse con entornos cuyos estados y acciones son continuas. Esto significa, que se pueden observar infinitos valores y elegir infinitas acciones. Para manejar este "infinito" es necesario parametrizar las funciones de valor de forma que, mediante iteraciones y aprendizaje, se consiga aproximar a las funciones óptimas de valor. Generalmente, esta parametrización se modela mediante redes neuronales. De esta manera surge también el Aprendizaje por refuerzo profundo aplicado a la robótica. Un ejemplo de arquitectura sería la siguiente:

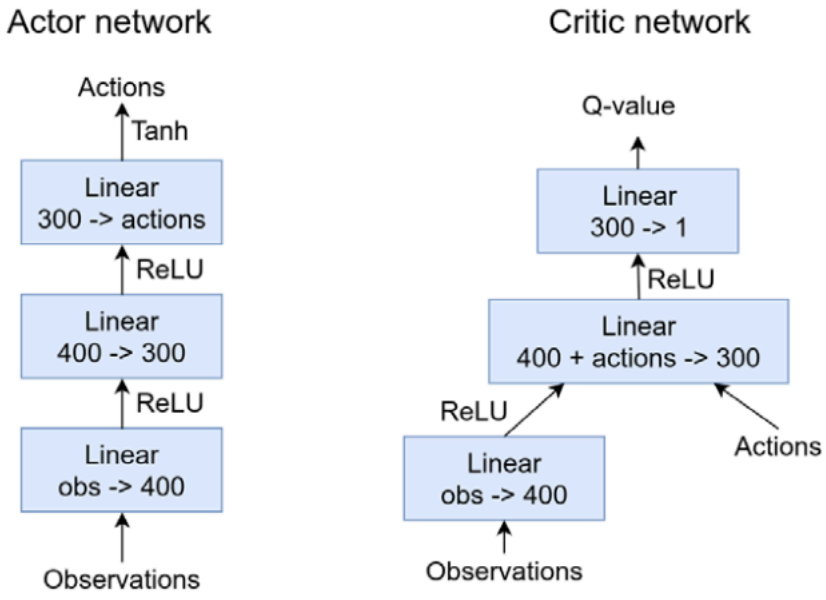


Figura 10: Arquitectura actor-crítica del algoritmo DDPG. Fuente: Deep Reinforcement Learning Hands-On

Esta arquitectura se corresponde con el algoritmo DDPG (Deep Deterministic Policy Gradient) [9][10]. Este algoritmo tiene dos redes neuronales profundas. La red actora decide qué acción escoger y la red crítica, a partir de estimar la función de valor estado-acción teniendo en cuenta el comportamiento, informa a la red actora de cómo de buena son sus acciones. Esto es la base de los algoritmos actor-críticos.

En cuanto a las redes neuronales profundas, en la imagen se puede observar que ambas funciones incorporan diversas capas de neuronas, cada recuadro azul es una capa. Así pues, también se puede construir un algoritmo de aprendizaje por refuerzo más o menos complejo en función de la cantidad de parámetros (o de neuronas) que formen el agente.

2.3. ESTUDIOS TÉCNICOS PREVIOS

Este proyecto parte de un trabajo extenso previo sobre los robots paralelos de rehabilitación llevado a cabo en la Universitat Politècnica de València por el Instituto de Automática e Informática Industrial. En lo que respecta a este trabajo, la parte de hardware que corresponde con los robots ya está solucionada. En primer lugar, las articulaciones activas del robot paralelo de tres grados de libertad y de los robots de cuatro grados, están formadas por un motor de corriente continua conectado a un tornillo sin fin y a una reductora, para convertir el giro del motor a un desplazamiento lineal. Entonces, para conocer estos desplazamientos, los motores tienen incorporados encoders en el eje.

También se ha desarrollado previamente un cuadro de control para cada uno de los robots que incluye la alimentación, un sistema de seguridad con una seta de emergencia y organiza entre el ordenador de control y las entradas y salidas del robot.

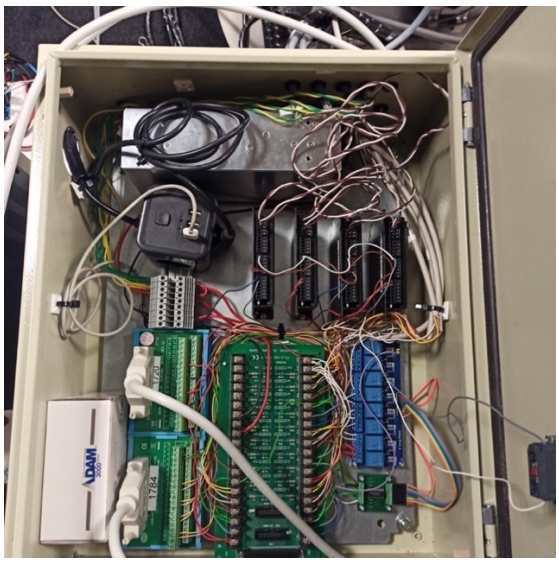


Figura 11: Cuadro de control del robot de 4DOF (versión nueva)

Figura 12: Cuadro de control del robot de 4DOF (versión nueva)

Así pues, previo a este proyecto, existía una arquitectura de control ya implementada basada en una combinación de ROS y de OROCOS. A través de este software se han implementados diversos controladores para el robot de tres grados de libertad y para el robot de cuatro grados. Se desarrolló también otra arquitectura basada sobre un sistema empotrado de tiempo real (Compact-RIO) a través de LabVIEW [11].

También está solucionado el modelo cinemático y dinámico de los robots. Se dispone también de un modelo de cada uno implementado en Simulink.

Por otra parte, también se ha desarrollado recientemente un nuevo robot paralelo de cuatro grados de libertad que mejora a su anterior versión. Este se ha diseñado con el fin de minimizar sus puntos singulares. Sus articulaciones esféricas son de más precisión y esta característica hace que se reduzcan las holguras. Por último, el recorrido de las articulaciones activas es mayor. Por lo que el espacio de trabajo es más grande también.

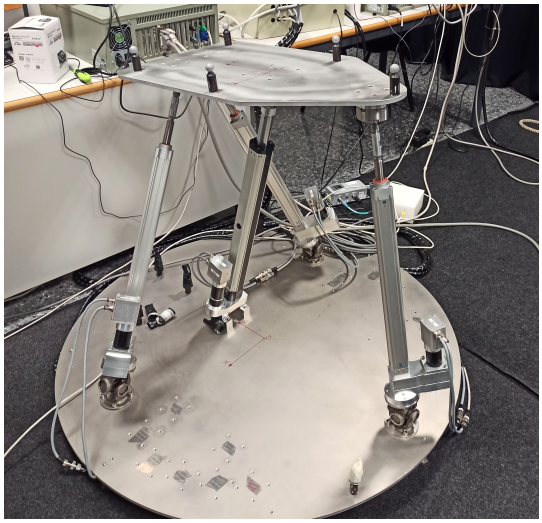


Figura 13: Nuevo robot de 4DOF

Así pues, este trabajo parte de el amplio estudio teórico y práctico mencionado anteriormente sobre los robots paralelos de rehabilitación. Por lo tanto, aspectos relacionados con el hardware de los robots el diseño teórico de controladores de posición y del modelo del robot teórico ya están solucionados.

3. FACTORES A CONSIDERAR

En los siguientes apartados se va a mostrar una lista de los factores a considerar para escoger la solución final y para el desarrollo del proyecto. Todos estos factores son de naturaleza técnica.

3.1. FACTORES TÉCNICOS

- La unidad de control (hardware) que se escoja para controlar el robot, debe ser compatible para controlar tanto el robot paralelo de 4 grados de libertad, como el robot de 3 grados. Esto incluye, tener 4 salidas analógicas y poder leer 4 encoders con dos canales digitales cada uno, en el caso de los robots de 4 grados de libertad, se requiere de una salida digital para actuar sobre los frenos.
- La unidad de control debe ser también abierta, se deben poder añadir sensores de una manera sencilla y se debe poder acceder a los parámetros de los controladores internos.
- El hardware debe tener una suficiente capacidad de cómputo como para poder realizar operaciones matemáticas complejas de una manera rápida. Esto es debido a que se quiere implementar controladores más complejos que un PID.
- El software y hardware debe tener la posibilidad de cumplir los requerimientos de tiempo real.
- El bucle de control debe tener como máximo un periodo de muestreo de 10 milisegundos.
- Se debe poder monitorizar la respuesta del controlador. Es decir, almacenar todos los datos de cada iteración del bucle de muestreo sin que esto afecte al rendimiento temporal del controlador.
- El software a seleccionar debe permitir realizar un desarrollo modular con el fin de facilitar la integración de diferentes sensores y desarrollar aplicaciones de usuario de más alto nivel.
- Se debe poder acceder desde múltiples dispositivos a la interfaz de usuario a desarrollar, no solamente desde ordenadores Linux.

4. SOLUCIONES ALTERNATIVAS

4.1. SOLUCIONES DE HARDWARE

Las soluciones de hardware propuestas se corresponden con la unidad de control del robot.

4.1.1. PLC INDUSTRIAL

Los Controladores Lógicos Programables permiten implementar controladores de una manera y sencilla para varios ejes recibiendo señales de sensores y actuando en motores. Por tanto, se podrían utilizar para controlar un robot. Además, estos son herramientas robustas y de tiempo real.

No obstante, en este proyecto se pretende implementar controladores más complejos que un PID en los que se requiere de un cálculo matricial complejo. Además, se desea implementar diversas estrategias de control de una manera modular y con varios enfoques, desde el control clásico hasta la implementación de técnicas de inteligencia artificial. Estos objetivos son difíciles de cumplir a través de las interfaces de programación de los PLC.

4.1.2. SISTEMA EMPOTRADO DE TIEMPO REAL

Un sistema empotrado de tiempo real como el controlador Compact RIO de National Instruments tiene la ventaja de que es un sistema robusto y certificado que puede adquirir señales digitales y actuar en salidas analógicas, lo necesario para los robots paralelos de rehabilitación. Los interfaces físicos están organizados en módulos que se pueden extraer e intercambiar. El lenguaje de programación con el cual se programa este controlador es LabVIEW, este lenguaje permite también la ejecución de operaciones matriciales. Entonces se podrían implementar controladores más complejos que un PID a través de este sistema empotrado.

Figura 14: Sistema CompactRIO. Fuente: https://www.ni.com/es-es/shop/compactrio.html

Como desventajas, el objetivo es implementar controladores complejos que requieren de un alto consumo de CPU, así como de memoria RAM (no excesivamente elevado, pero sí para un sistema empotrado). Sobre la implementación de controladores de posición (que sí que son posibles de implementar en un sistema CRIO)

se pretende añadir más capas de control como controladores de fuerza, escapar de singularidades y algoritmos de inteligencia artificial.

4.1.3. SISTEMA EMPOTRADO CON GPU

Otra posibilidad es el uso de un sistema empotrado con una GPU. Un ejemplo de este sistema es el dispositivo Jetson TX2 de Nvidia. Es un sistema que permite la implementación de técnicas de inteligencia artificial en un computador de muy pequeño tamaño. Están diseñados para que la predicción directa de algoritmos basados en redes neuronales sea muy rápida. Generalmente el entrenamiento se realiza en otros dispositivos con mayor potencia.

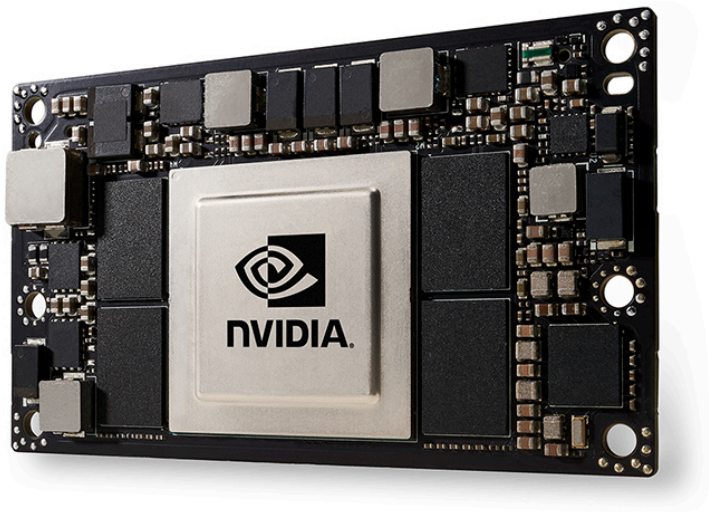


Figura 15: Envidia Jetson TX2. Fuente: https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-tx2/

En cuanto a memoria RAM, La Nvidia Jetson TX2 tiene suficiente memoria para poder desarrollar controladores complejos (8GB).

4.1.4. ORDENADOR INDUSTRIAL CON TARJETAS DE ADQUISICIÓN DE DATOS

La última solución propuesta es el uso de un ordenador industrial con tarjetas de adquisición de datos. Estas tarjetas deben tener el número necesario de entradas y salidas para poder acceder a todos los sensores y actuadores de los robots (4 salidas analógicas y 4 encoders con dos canales digitales cada uno, también hace falta otra salida digital para controlar el freno del robot de 4 grados de libertad).

En cuanto al ordenador, debe tener una CPU lo suficientemente potente como para poder implementar controladores complejos que requieren grandes operaciones numéricas en poco tiempo. Los controladores de posición quizá no consumen mucha CPU, pero sobre estos se van a construir más capas de controladores como de fuerza y escape de singularidades que sí requieren de mucho consumo de procesador.

4.2. SOLUCIONES DE SOFTWARE

4.2.1. ROS + OROCOS

Como se ha mencionado anteriormente, ya se ha realizado una implementación a través de una combinación de ROS (Robot Operating System) y de OROCOS (Open Robot Control Software Project). El software utilizado para los robots paralelos hace uson también de la OROCOS toolchain, que permite crear componentes de OROCOS al igual que se ha explicado anteriormente con ROS2. Entonces, estos softwares cumplirían con la modularidad y los requerimientos de tiempo real que se exigen en este proyecto.

Sin embargo, OROCOS está dejando de tener soporte y es difícil de integrar con nuevas versiones de librerías y de sistemas operativos.

4.2.2. ROS2

ROS2 es una versión nueva de ROS cuya implementación es totalmente nueva y distinta. Este diseño distinto de ROS puesto que la nueva versión pretende cumplir con los requerimientos de tiempo real. ROS2 permite implementar componentes que realicen funciones determinadas en un robot de manera que el desarrollo sea modular como en OROCOS. Gracias también al nuevo sistema de distribución de datos (DDS), es sencillo desarrollar un sistema distribuido en el que diversos ordenadores se comuniquen entre sí utilizando los modelos de comunicación que ROS2 ofrece. Además, se puede desarrollar la programación en diversos lenguajes de programación, como C++ y Python. También existe una API de ROS2 para MATLAB. Esta interfaz permitiría comunicar el modelo de los robots implementados en Simulink con la implementación del controlador real.

Otra ventaja es que ROS2 al ser nuevo es fácil de integrar junto con otros paquetes desarrollados por la comunidad, un ejemplo es el paquete ROS2Learn [12]. Se trata de un conjunto de herramientas para entrenar a algoritmos de aprendizaje por refuerzo en ROS2.

Como desventaja, ROS2 acaba de comenzar su desarrollo (se lanzó la primera distribución en 2017), entonces está pendiente de mejora y la versión utilizada en este proyecto no estará tan optimizada como las versiones siguientes (como por ejemplo Foxy que termina su desarrollo en 2023).

5. DESCRIPCIÓN DETALLADA DE LA SOLUCIÓN ADOPTADA

5.1. SOLUCIÓN HARDWARE

5.1.1. ORDENADOR INDUSTRIAL DE CONTROL

Se ha elegido utilizar un ordenador industrial como unidad de control junto con tarjetas de adquisición de datos. Esta es la solución que permite más flexibilidad en el sentido de seleccionar los componentes que forman el ordenador. Para una aplicación

comercial sí que sería interesante utilizar un sistema empotrado, con las restricciones que conlleva. Pero ahora mismo, utilizar una placa no aporta ventajas.

Las características más importantes del ordenador seleccionado son las siguientes:

- Memoria RAM: 16 GB
- Procesador: Intel Core i7-6700 3,40 GHz, 4 núcleos
- Disco duro: 1 TB

Se trata de un ordenador con un buen procesador debido a que los controladores a implementar van a requerir de una gran capacidad y velocidad de cómputo.

5.1.2. TARJETAS DE ADQUISICIÓN DE DATOS

Como interfaces físicos para el ordenador industrial, se han utilizado tarjetas de adquisición de datos. Para las entradas y salidas digitales se ha utilizado la tarjeta PCI-1784U de Advantech. Esta tarjeta permite leer hasta cuatro ejes con encoders con una resolución de 32 bits. Cada par de canales destinado a cada encoder se ha configurado en modo ABPhase para realizar el conteo de manera incremental empezando desde 0. La tarjeta también cuenta con 4 salidas digitales de las cuales una se va a utilizar para controlar los frenos de los robots de 4 grados de libertad.

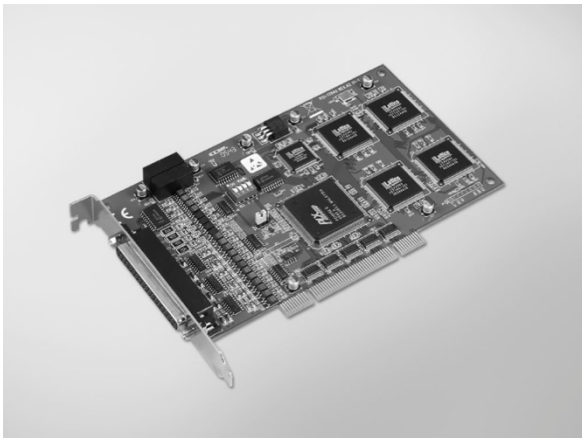


Figura 16: Tarjeta de adquisición de datos 1784U de Advantech. Fuente: Datasheet 1784U
https://www.advantech.com/products/1-2mlk9r/pci-1784u/mod_96795d5f-1e79-4f65-94e6-60dd07ec8133

Para el manejo de los motores se ha utilizado una segunda tarjeta de salidas analógicas. El modelo de la tarjeta es la PCI-1720U de Advantech también.

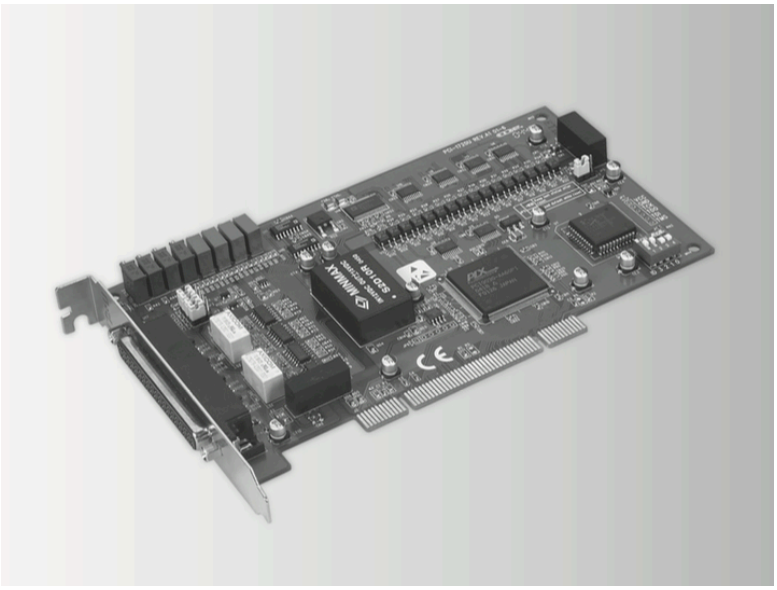


Figura 17: Tarjeta de adquisición de datos PCI-1720U de Advantech. Fuente: Datasheet 1720U
https://www.advantech.com/products/1-2mlk8x/pci-1720u/mod_e2ea2508-346e-410f-b18f-0c6aa0e1f28e

Esta tarjeta tiene cuatro canales de salida analógica de 12 bits de resolución cuyos rangos de voltaje bipolar se comprende entre ±10 V. Para el acceso a estas tarjetas desde un programa es necesario el kit de desarrollo de software proporcionado por Advantech.

5.1.3. ORDENADOR PARA EL ALMACENAMIENTO DE DATOS

Las operaciones de escritura en disco duro no son deterministas y es recomendable evitarlas dentro de un bucle de control de tiempo real. Por este motivo, aprovechando también la capacidad de ROS2 de trabajar con diferentes máquinas de una manera sencilla, se ha utilizado un segundo ordenador que accede a los topics creados en el ordenador de control y almacena los datos de cada iteración del bucle de control en ficheros. Los datos almacenados son variables significativas en el control del robot que se utilizan para estudiar y evaluar la respuesta de los controladores.

Otra alternativa podría ser guardar los datos de cada iteración en memoria RAM y después, cuando termina la trayectoria, escribir estos datos en un fichero. Sin embargo, los ejercicios de rehabilitación pueden ser muy largos dando lugar a ocupar mucha memoria RAM. Por este motivo se ha optado por utilizar un segundo ordenador para escribir los datos en disco duro sin que estas operaciones afecten al control del robot. Las características de este son las siguientes:

- Memoria RAM: 16GB

- Intel Core i3-10100 CPU 3,6 GHz, 4 núcleos
- Disco duro: 1 TB

El procesador no es tan potente como el anterior puesto las operaciones que realiza este ordenador no son críticas.

5.1.4. Sistema de captura de movimiento

Los robots paralelos utilizados, concretamente los de 4 grados de libertad, tienen el inconveniente de presentar singularidades. Estos puntos son peligrosos para el control y es necesario detectar cuando el robot se encuentra en estos puntos. En estos puntos singulares el robot generalmente gana como mínimo un grado de libertad y el punto se caracteriza por tener un determinante del Jacobiano Directo cercano a 0. Este valor cercano a 0 es peligroso para el control en el caso de que se calcule la cinemática directa. El cálculo de la cinemática directa depende de la inversa de la matriz Jacobiana Directa. Si esta matriz tiene un bajo determinante, los valores calculados pueden inestabilizar.

Entonces, cerca de una singularidad, la estimación obtenida por el modelo del robot (cinemática directa) tampoco es fiable. Es necesario un sistema de medición redundante con el que se pueda obtener la posición con fiabilidad de la plataforma móvil.

El sistema seleccionado es un sistema de captura de movimiento, Optitrack. Este sistema está disponible en el Laboratorio de Robótica. Este es un sistema formado por 10 cámaras Flex13 con las que se puede obtener la posición 3D de objetos reflectantes dado que las cámaras trabajan a partir de la emisión de infrarrojos.

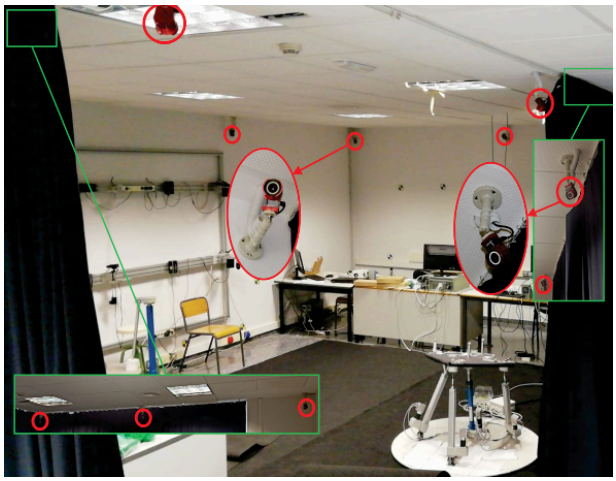


Figura 18: Sistema de cámaras de Optitrack instalado en el laboratorio. Fuente: Vision-Based Hybrid Controller to Release a 4-DOF Parallel Robot from a Type II Singularity, https://www.mdpi.com/1424-8220/21/12/4080

En cuanto a especificaciones técnicas, las cámaras tienen una resolución de 1,3 Megapíxeles y una velocidad de captura de 120 Hz (8,3 ms). Entonces, es posible utilizar la información obtenida por las cámaras para realimentar los controladores puesto que

estos funcionan a 10 ms. En el laboratorio se dispone de un conjunto de marcadores reflectantes que serán los objetos cuya posición será obtenida por las cámaras.

Para sincronizar las cámaras, se dispone de dos dispositivos OptiHub, que también funcionan para alimentar de manera más consistente las cámaras y mejorar la captura de movimiento.

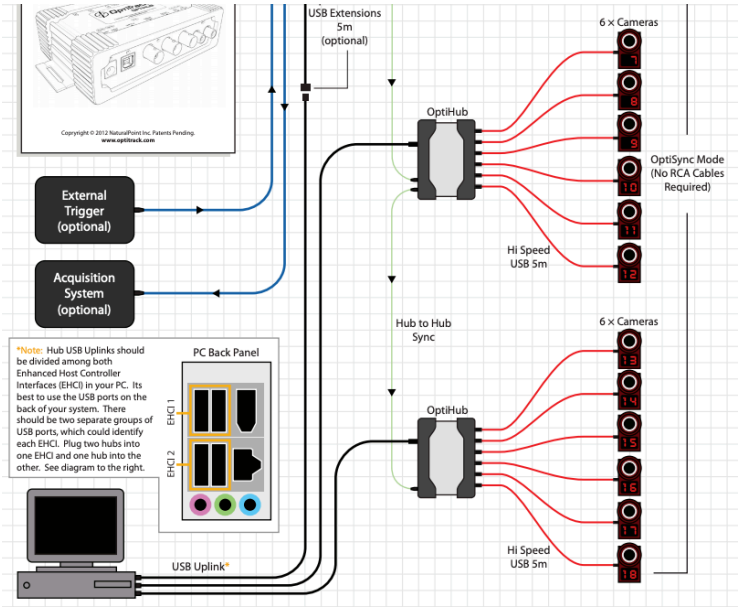


Figura 19: Diagrama de conexión con dispositivos OptiHub. Fuente: Optihub start guide, https://optitrack.com/support/hardware/optihub2.html

Estos dispositivos están conectados con el ordenador de control de las cámaras a través de una conexión USB. Este ordenador es distinto al ordenador de control de los robots. Entonces, para poder acceder a la información de la captura de movimiento y calibrar las cámaras, se ha instalado el software Motive.

5.1.5. SENSOR DE FUERZA

Se ha instalado un sensor de fuerza junto con una bota encima de la plataforma del robot para conocer la fuerza que esta ejerce sobre el paciente y poder implementar controladores de fuerza cuyo propósito es primero, de establecer una interacción segura con el ser humano y segundo de realizar ejercicios de rehabilitación resistivos.

En la siguiente imagen se muestra el montaje del sensor junto con el arnés para sujetar el pie:

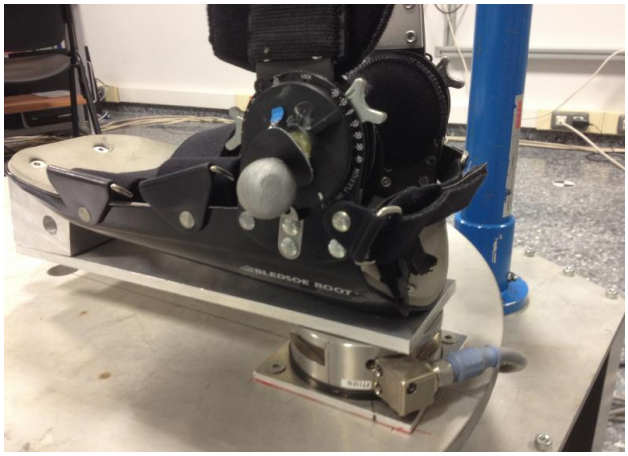


Figura 20: Sensor de fuerza con el arnés sobre la plataforma del robot de 3DOF. Fuente: https://imbio3r.ai2.upv.es/galeria_de_fotos/force-sensor

El sensor de fuerza utilizado es el correspondiente con el modelo F/T Sensor Delta. Se caracteriza por poder medir altos valores de fuerza y un alto valor del ratio señal-ruido. Estas características hacen que el sensor sea apropiado para aplicaciones de robótica de realimentación de fuerza en tiempo real.

El sensor de fuerza incluye una NetBox que permite la comunicación a través de Ethernet o un BUS CAN.

Figura 21: NetBox. Fuente: https://www.ati-ia.com/products/ft/ft_NetFT.aspx

5.2. SOLUCIÓN SOFTWARE

5.2.1. SISTEMA OPERATIVO UBUNTU

Anteriormente, se ha mencionado que ROS2 puede funcionar en sistemas operativos Windows, macOS y Linux. Para este proyecto se ha utilizado el sistema operativo Ubuntu que está basado en Linux. Se ha preferido a este con respecto a los otros dos debido a que el software es libre y por tanto gratuito. También debido a que hay mucho más desarrollo de ROS y de ROS2 sobre este software y es más sencillo de desarrollar aplicaciones basadas en lenguaje C++ y Python sobre este sistema operativo.

La versión de Ubuntu elegida ha sido la 18.04 (con escritorio). Se ha escogido esta debido a que es la última versión compatible con los drivers de las tarjetas de adquisición de datos de Advantech.

5.2.2. ROS2

Para desarrollar los controladores se ha seleccionado el software ROS2. Como se ha explicado anteriormente, ROS2 permite realizar una programación modular para los controladores de los robots. Se ha diseñado para cumplir los requerimientos de tiempo real, en el sentido de operaciones deterministas, gestión de memoria y calidad del servicio [13]. En este sentido, ROS2 es una mejor opción que ROS para el control de los robots. Se ha escogido ROS2 sobre OROCOS debido a que tiene más soporte actualmente, haciéndolo más fácil de integrar con nuevo software.

ROS2 está actualmente siendo desarrollado, pero cuenta con diversas distribuciones. Generalmente, los cambios añadidos entre distribuciones no afectan a la manera de programar, excepto en las primeras distribuciones, actualmente se han lanzado 7 distribuciones. La que se ha utilizado en este proyecto es la distribución Eloquent Elusor. Se ha elegido esta porque es la última versión que puede soportar la versión de Ubuntu 18.04. La siguiente versión, Foxy Fitzroy (que tiene más soporte), solamente es compatible con Ubuntu 20.04. En la siguiente figura se muestran las distribuciones que se han lanzado:

Distro	Release date	Logo	EOL date
Galactic Geochelone	May 23rd, 2021		November 2022
Foxy Fitzroy	June 5th, 2020		May 2023
Eloquent Elusor	November 22nd, 2019		November 2020
Dashing Diademata	May 31st, 2019		May 2021
Crystal Clemmys	December 14th, 2018		December 2019
Bouncy Bolson	July 2nd, 2018		July 2019
Ardent Apalone	December 8th, 2017		December 2018
beta3	September 13th, 2017		December 2017
beta2	July 5th, 2017		September 2017
beta1	December 19th, 2016		Jul 2017
alpha1 - alpha8	August 31th, 2015		December 2016

Figura 22: Planificación de distribuciones de ROS2. Fuente: https://docs.ros.org/en/foxy/Releases.html

Sin embargo, con el código fuente desarrollado, se puede cambiar de una manera sencilla desde Eloquent a Foxy. Por otro lado, se ha seleccionado Eloquent sobre Dashing porque en el momento de realizar la interfaz gráfica, las herramientas de ROS2 para programar aplicaciones web no eran compatibles con la versión Dashing.

Para la instalación de ROS2 Eloquent, se han instalado los paquetes Debian disponibles. ***sudo apt install ros-eloquent-desktop*** [14].

Se han instalado los paquetes Desktop, que además de instalar los componentes y librerías esenciales de ROS2, instala aplicaciones gráficas de visualización, demos y tutoriales útiles. Una aplicación gráfica útil que se ha utilizado en este trabajo es RQT, la cual permite visualizar la red de topics y nodos de ROS2 construida (rqt_graph) y también representar gráficamente la información que fluye por los topics.

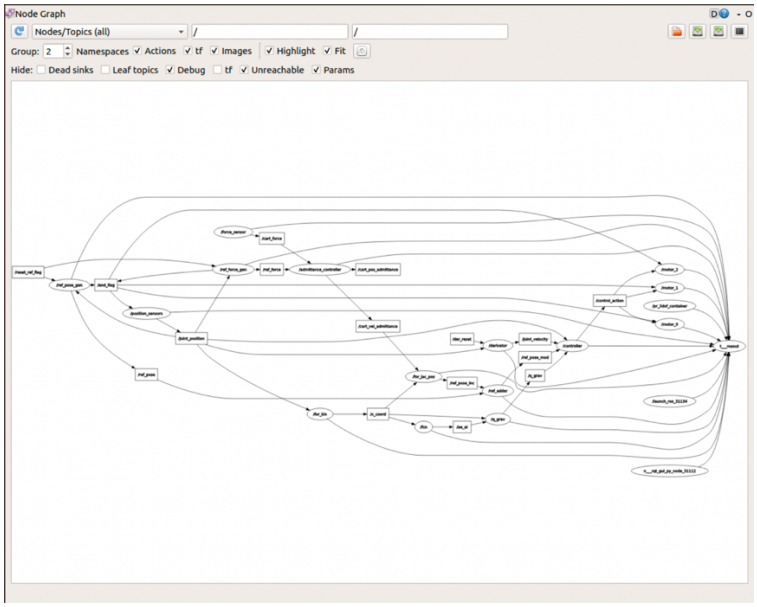


Figura 23: Aplicación rqt_graph

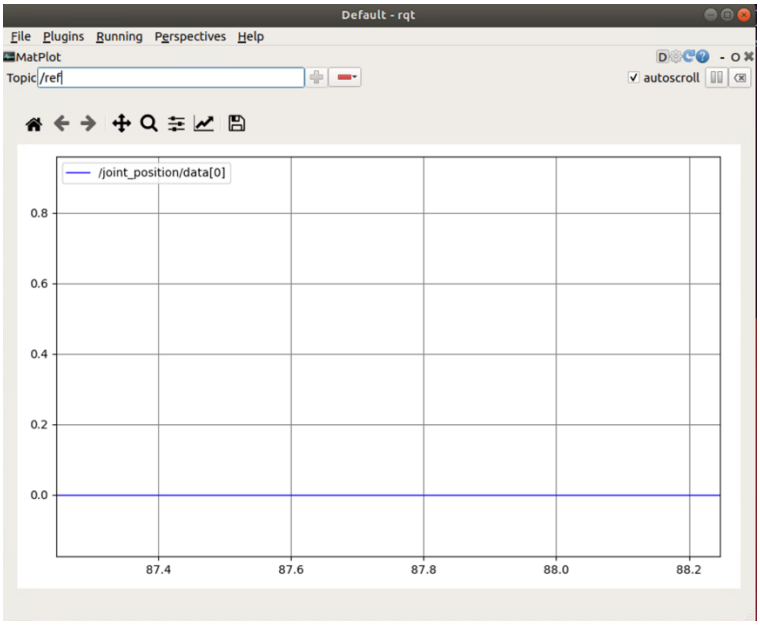


Figura 24: Aplicación rqt

Una vez instalado ROS2, es necesario instalar colcon. Este es un programa (herramientas para la línea de comandos) que se utiliza para compilar o construir los paquetes de ROS2. Se debe utilizar colcon en el directorio del espacio de trabajo. Con la siguiente instrucción se compilarían todos los paquetes: ***colcon build*** Con esta otra sentencia se puede seleccionar que solamente se compile uno o varios paquetes determinados: ***colcon build –packages-select <nombres_paquetes>***.

Así pues, ya se tiene todo preparado para poder crear un paquete de ROS2. Se pueden crear paquetes basados en C++ o basados en Python. Los basados en C++ requieren de un script cmake para construir un ejecutable o una librería compartida.

Además de las funciones propias de cmake, también se añaden otras herramientas como ament_cmake, que ayudan a compilar los paquetes de ROS2 y a crear componentes entre otras funciones.

Entonces, para trabajar con ROS2, se ha construido un directorio que es el espacio de trabajo. Dentro de este directorio, se ha creado otro directorio src en el cual se incluyen los paquetes desarrollados. A la hora de compilar el entorno de trabajo, se crean las carpetas build, install y log que incluyen los archivos construidos y scripts para que estos sean visibles y se añadan a la ruta de las variables de entorno que ROS2 utiliza.

Se ha construido un entorno de trabajo para el robot de 4 grados de libertad y otro para el de 3 grados. De manera que cada entorno contiene los paquetes necesarios para cada robot. Con el entorno del robot de 4 grados, se pueden controlar los dos robots de 4DOF disponibles en el laboratorio, ya que lo único que cambia entre estos es la configuración geométrica y de masas de los robots. Todas las funciones implementadas están parametrizadas en función de estos parámetros, es lo que se ha denominado la configuración del robot.

La programación se ha desarrollado en componentes de ROS2. Estos componentes están formados a partir de una clase de C++ con rclcpp implementada en un archivo .cpp y definida en un archivo de cabecera .hpp. Un ejemplo de clase básica para un nodo de ROS2 es el siguiente:

```
#include <memory>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"
using std::placeholders::_1;

class MinimalSubscriber : public rclcpp::Node
{
  public:
    MinimalSubscriber()
    : Node("minimal_subscriber")
    {
      subscription_ = this->create_subscription<std_msgs::msg::String>(
      "topic", 10, std::bind(&MinimalSubscriber::topic_callback, this, _1));
    }

  private:
    void topic_callback(const std_msgs::msg::String::SharedPtr msg) const
    {
      RCLCPP_INFO(this->get_logger(), "I heard: '%s'", msg->data.c_str());
    }
    rclcpp::Subscription<std_msgs::msg::String>::SharedPtr subscription_;
};
```

Esta clase implementa lo mínimo necesario para hacer un nodo con una suscripción en ROS2. La clase tiene que heredar de rclcpp::Node y en el constructor se le asigna un nombre por defecto al nodo. Después, se construye la suscripción asignando una callback que se ejecutará cuando haya un mensaje nuevo en el topic al cual está

suscrito. La ventaja de utilizar los componentes es que se pueden componer varios en un único proceso.

A continuación, se muestra un ejemplo mínimo de un publicador implementado en Python con rclpy, se han desarrollado nodos en este lenguaje únicamente para implementar el controlador basado en técnicas de inteligencia artificial.

```
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class MinimalPublisher(Node):

    def __init__(self):
        super().__init__('minimal_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', 10)
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello World: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1
```

El contenido de esta clase es similar a la anterior implementada en C++. Pero en este caso, en vez de una suscripción, se tiene un publicador con un temporizador. En el constructor se crea el publicador indicando el nombre del topic y después se crea un temporizador que está asociado a una callback. Esta función se ejecuta cada vez que vence el tiempo.

Los dos ejemplos de nodos pueden comunicarse entre sí independientemente de si uno ha sido desarrollado en C++ o de si ha sido implementado con Python. Sin embargo, la implementación de los controladores de posición y de fuerza se ha realizado a partir de componentes escritos en C++, esto es debido a que en C++ se puede gestionar más eficientemente la memoria, las operaciones son más rápidas y que se pueden componer los componentes en un único proceso. La desventaja es que se requiere más tiempo de desarrollo que Python.

Una vez se cree una aplicación de ROS2 formada por varios nodos y topics, durante el desarrollo es difícil de poder ver dónde ha podido haber un error. Es por ello que ROS2 ofrece una serie de herramientas útiles para visualizar rápidamente los nodos que se han ejecutado, los topics, los mensajes, etc. Estas herramientas se han utilizado a lo largo del desarrollo de este proyecto y son las siguientes:

Comando	Función
ros2 node list	Mostrar una lista con todos los nodos se han creado.
ros2 node info <nombre_nodo>	Muestra información importante de un nodo, sus suscripciones, en qué topics publica, servicios y acciones.
ros2 topic list	Lista todos los topics.
ros2 topic echo <nombre_topic>	Se suscribe al topic indicado y muestra por pantalla los mensajes.
ros2 topic info <nombre_topic>	Muestra por pantalla el tipo de mensaje del topic, el número de publicadores y el número de suscriptores.
ros2 interface show <nombre_interfaz>	Muestra el contenido de una interfaz de ROS2, estas son los mensajes (msg), servicios (srv) y acciones (action).
ros2 topic pub <nombre_topic> <tipo_msg> '<argumentos>'	Publica el mensaje definido en la sentencia en el topic indicado.
ros2 param list	Lista todos los parámetros que hay.
ros2 param get <nombre_nodo> <nombre_parametro>	Devuelve el valor del parámetro indicado.
ros2 param set <nombre_nodo> <nombre_parametro> <valor>	Establece el valor del parámetro al indicado en la sentencia.
ros2 run <nombre_paquete> <nombre_ejecutable>	Ejecuta un nodo implementado en un ejecutable.
ros2 launch <nombre_paquete> <nombre_ejecutable>	Ejecuta un archivo launch que generalmente se utiliza para ejecutar varios nodos.

Tabla 1: Listas de comandos útiles de ROS2

Otra cuestión importante de ROS2 es lo denominado calidad del servicio. Se trata de una serie de parámetros que se pueden establecer a la hora de construir una comunicación publicador subscriptor para hacer que esta sea tan fiable como TCP o más rápida pero menos fiable como UDP. Otros parámetros son el número de la cola, la duración del dato publicado, etc.

5.2.2.1. MENSAJES DE ROS2 CONSTRUIDOS

Con el fin de comunicar los diferentes nodos de ROS2 implementados para controlar los robots se han desarrollado mensajes personalizados. Los mensajes son estructuras de datos para comunicar los nodos a través de la comunicación publicador-subscriptor. Estos mensajes parten de datos básicos y se pueden construir estructuras más complejas.

Para construir los mensajes, se han implementado dos paquetes de ROS2, uno para el robot de 4 grados de libertad (pr_msgs) y otro para el de 3 grados

(pr_3dof_msgs). Estos mensajes se compilan a través de scripts de cmake. En la siguiente tabla se lista cada mensaje implementado, con las estructuras de datos que contiene y una explicación de su uso:

pr_msgs	Contenido	Uso
msg		
PRArray	float64[4] data	Mensaje para comunicar los 4 grados de libertad del robot (posición cartesiana/articulaciones, velocidad, etc.)
PRArrayH	std_msgs/Header header builtin_interfaces/Time current_time float64[4] data	Mensaje con la misma estructura que PRArray, pero añadiendo una cabecera y una estructura para guardar un valor de tiempo.
PRForceStateH	std_msgs/Header header builtin_interfaces/Time current_time float[3] force float[3] momentum	Se utiliza para comunicar los 3 valores de fuerza y los 3 valores de momento obtenidos por el sensor de fuerza. Incluye una cabecera y un mensaje para guardar el tiempo.
PRMat	uint32 rows uint32 cols float64[<=140]	Este mensaje se utiliza para enviar matrices (se envían en un vector unidimensional).
PRMatH	std_msgs/Header header builtin_interfaces/Time current_time uint32 rows uint32 cols float64[<=140]	Este mensaje es el mismo que el anterior, pero añadiendo la cabecera y la estructura para guardar un valor de tiempo.
PRMocapH	std_msgs/Header header builtin_interfaces/Time current_time pr_msgs/PRArray x_coord float64 error float64 latency	Con este mensaje se comunica la posición de la Plataforma móvil obtenida por las cámaras. Incluye una cabecera, un dato de tiempo, un mensaje PRArray para guardar la posición y un dato tipo float64 para comunicar el error entre el modelo y las cámaras, y otro dato para comunicar la latencia entre el Motive y el ordenador de control.
PROTS	std_msgs/Header header builtin_interfaces/Time init_time float64[6] ots_ang pr_msgs/PRMat ots	Con este mensaje se comunican los ángulos OTS y sus diferentes soluciones, esto se tratará en el apartado del controlador para escapar de una singularidad.
PRStateH	std_msgs/Header header builtin_interfaces/Time current_time pr_msgs/PRArray q	Este mensaje almacena el estado del robot (posición y velocidad de las

	pr_msgs/PRArray q_vel	articulaciones). Se utiliza para el algoritmo de inteligencia artificial.
PRFloatH	std_msgs/Header header builtin_interfaces/Time current_time float64 data	Se almacena un dato float64 junto con una cabecera y un valor de tiempo.
srv		
Trajectory	string path_trajectory bool is_cart --- int8 error_code int32 n_ref_loaded	Interfaz srv que se utiliza para enviar una petición al servidor de trayectorias con la ruta del fichero de la trayectoria e indicando si es cartesiana o articular. Devuelve un código de error y el número de referencias cargadas.

Tabla 2: Interfaces implementados para el robot de 4 grados de libertad

pr_3dof_msgs	**Contenido**	**Uso**
msg		
PRArray	float[3] data	Mensaje para variables relevantes en el control (posición, velocidad, etc.)
PRArrayH	std_msgs/Header header builtin_interfaces/Time current_time float64[3] data	Mensaje con la misma estructura que PRArray, pero con cabecera y una estructura de tiempo
PRLongArrayH	std_msgs/Header header builtin_interfaces/Time current_time float64[9] data	Mensaje con cabecera y una estructura de tiempo para comunicar un vector de 9 componentes.
PRMat	uint32 rows uint32 cols float64[<=100] data	A través de este mensaje se pueden enviar matrices unidimensionalmente, indicando las filas y las columnas que tiene.
PRForceState	std_msgs/Header header builtin_interfaces/Time current_time float64[3] force float64[3] momentum	Se utiliza para enviar los 3 valores de fuerza en x,y,z obtenidos por el sensor de fuerza y los tres valores de par. Tiene una cabecera y una estructura de tiempo.
PRAeAiH	std_msgs/Header header builtin_interfaces/Time current_time pr_3dof_msgs/PRMat ae pr_3dof_msgs/PRMat ai	Este mensaje se usa para comunicar las matrices Ae y Ai que se utilizan para el cálculo de los términos gravitacionales. Incluye una cabecera y un valor de tiempo.
PRState	std_msgs/Header header builtin_interfaces/Time current_time	Este mensaje se utiliza en el controlador de Reinforcement

	pr_3dof_msgs/PRArray q pr_3dof_msgs/PRArray q_vel	Learning para comunicar el estado del robot (posición y velocidad), incluye una cabecera y un valor de tiempo.
PRPDState	std_msgs/Header header builtin_interfaces/Time current_time pr_3dof_msgs/PRArray q pr_3dof_msgs/PRArray q_vel pr_3dof_msgs/PRArray q_ref pr_3dof_msgs/PRArray q_ref_vel	Se utiliza en el controlador de aprendizaje por refuerzo para comunicar el estado del robot: posición, velocidad, referencia de posición y referencia de velocidad (todas en el espacio de articulaciones). Incluye también una cabecera y un valor de tiempo.
PRRLInfo	std_msgs/Header header builtin_interfaces/Time current_time pr_3dof_msgs/PRArray q pr_3dof_msgs/PRArray q_vel pr_3dof_msgs/PRArray q_error pr_3dof_msgs/PRArray q_vel_error float64 reward float64 current_score	Este mensaje se usa para enviar información relevante durante el entrenamiento de los algoritmos de aprendizaje por refuerzo. Incluye una cabecera y un valor de tiempo.
srv		
Trajectory	string path_trajectory bool is_cart --- int8 error_code int32 n_ref_loaded	Mensaje srv que se utiliza para hacer una petición en la que se envía la ruta de un fichero de referencias y se indica si esta está en espacio articular o cartesiano. La respuesta está formada por un código de error y un número que indica cuantas referencias se han cargado. Se usa para comunicar desde la aplicación web que comience un ejercicio.

Tabla 3: Interfaces implementados para el robot de 3 grados de libertad

5.2.2.2. LIBRERÍA EIGEN Y OPERACIONES CON MATRICES

Para el desarrollo de los controladores es necesario realizar operaciones con matrices, sumas, restas, productos matriciales, cálculo de inversas, etc. Para simplificar estas operaciones se ha utilizado la librería de C++ Eigen [15]. Esta es una "template library" definida a través de únicamente ficheros de cabeceras.

Como características, soporta matrices de todos los tamaños y también de tamaño dinámico y fijo. La implementación está optimizada para las matrices de un tamaño fijo, cuyas operaciones se realizan de una manera muy rápida. Esta librería tiene únicamente como dependencia las librerías estándares de C++. La librería Eigen se ha utilizado en otros proyectos de gran relevancia como Tensorflow, Point Cloud Library o ROS.

Así pues, Eigen permite desarrollar las mismas operaciones implementadas en los controladores de MATLAB/Simulink. Cada operación matricial de MATLAB tiene su equivalente en Eigen. Esta dependencia ha sido integrada en los paquetes de ROS2 gracias al script cmake mediante el cual se puede encontrar el paquete eigen y añadirlo como dependencia al fichero a construir.

En cada componente implementado, generalmente las matrices y vectores se han modelado como objetos de Eigen para operar con ellos de una manera sencilla. Sin embargo, la información de los mensajes de ROS2 está modelada a través de estructuras del tipo std::vector y std::array. Es por ello que en cada componente, al recibir un mensaje, se convierten los datos de estas estructuras a matrices y vectores de Eigen. Cuando se ha operado, se realiza la conversión inversa para poder enviar el mensaje de ROS2.

Un ejemplo de mensaje que envía una matriz es PRMatH. Este mensaje contiene la información de una matriz, pero unidimensionalmente, entonces es por este motivo que también se comunica el número de filas y columnas que contiene la matriz. ROS2 incluye un mensaje estándar para comunicar matrices Float32MultiArray, pero su uso ha quedado obsoleto.

En los componentes, los objetos de Eigen se instancian y se inicializan en el constructor y después, en los callbacks se modifica su valor. Se ha evitado declararlas en las callbacks para minimizar las operaciones de asignación de memoria en el bucle de control.

5.2.2.3. MESSAGE FILTERS Y MEDIDAS DE TIEMPOS

Se ha utilizado la librería Message Filters (propia de ROS2) en nodos que tienen diversas suscripciones. Cada suscripción del nodo tiene su propia callback, entonces, es necesario una nueva callback que se ejecute cuando se haya recibido un mensaje nuevo en cada uno de los topics suscritos. Esto es lo que resuelve Message Filters.

Entonces, la librería permite sincronizar mensajes de igual y de distinta frecuencia. Tiene dos políticas de sincronización, la de tiempo exacto y la de tiempo aproximado. La de tiempo exacto solamente ejecuta el callback global si los mensajes recibidos tienen exactamente el mismo valor de tiempo. Este valor de tiempo se almacena en la cabecera de los mensajes, entonces, todos los mensajes personalizados que se quieran utilizar con Message Filters deben tener una cabecera.

La política de tiempo aproximado permite que los mensajes no tengan exactamente el mismo tiempo para que el callback global se ejecute. Se puede añadir un valor de tiempo límite entre mensaje y mensaje para descartarlos y no ejecutar el callback.

La política utilizada en este proyecto ha sido la de tiempo exacto. Por una parte, es prácticamente imposible que dos mensajes se hayan enviado exactamente en el mismo tiempo, sin embargo, para asegurarse de que en una iteración del bucle de control se opere con mensajes únicamente enviados en esta iteración, se mide el tiempo en el nodo del Encoder (el primero) y en los siguientes mensajes se iguala el valor del tiempo de la cabecera al valor del Encoder. De esta manera no puede suceder que en una iteración del bucle de control se utilicen mensajes de otra iteración distinta y que haya un desfase. Sin embargo, esta política no es flexible en el sentido de que, si un nodo tarda demasiado en realizar los cálculos y no envía el mensaje a tiempo con respecto al periodo de control, el bucle de control entero se pierde, es decir, no hay acción de control en dicho periodo.

Esto no ha sido ningún problema en los controladores implementados en este proyecto puesto que se ha estudiado la respuesta temporal y se cumple siempre el periodo de muestreo, es decir, se emite una acción de control antes de que el temporizador del Encoder termine su espera.

Para controladores en los que ciertos nodos requieran un tiempo de cálculo mayor, se puede flexibilizar esta política con la de tiempo aproximado o también separando el callback asociado a la suscripción del topic que tenga una frecuencia menor y que la callback global y la anterior se comuniquen la información por medio de una variable de la clase.

Para medir los tiempos se ha utilizado el siguiente método perteneciente a la clase rclcpp::Node de ROS2 **get_clock()**, este devuelve un objeto del tipo rclcpp::Clock::SharedPtr mediante el cual aplicando el método **now()** se obtiene un stamp de tiempo actual cuyo valor tiene una componente en segundos y otra en nanosegundos.

Entonces, en el callback asociado al temporizador del componente del encoder, antes de publicar el mensaje correspondiente con la posición medida, se obtiene el stamp de tiempo actual mediante el método now() y se almacena en la cabecera del mensaje. Como se ha explicado anteriormente, en los siguiente callbacks, se sigue almacenando el mismo stamp para que todos los datos estén asociados a la misma iteración del bucle de control.

Además, para monitorizar los tiempos de ejecución de cada callback y estudiar la respuesta temporal del sistema, a cada mensaje se le ha añadido un nuevo campo que es un nuevo de stamp de tiempo. Este se utiliza para guardar el instante justo antes de publicar un mensaje (para que no se almacenen únicamente los instantes del encoder).

5.2.3. LIBRERÍAS PARA LOS ROBOTS

Se ha desarrollado una librería para el robot de 4 grados de libertad y otra para el de 3 grados (pr_lib y pr_3dof_lib). Ambas librerías son un paquete de ROS2 basado en C++ en el que en vez de crear un ejecutable o componente, se construye una librería de forma que otros paquetes pueden utilizar sus archivos precompilados y las cabeceras.

La función de estas librerías consiste en la implementación de las funciones relacionadas con los cálculos que tienen que ver con el modelo del robot, por ejemplo, la cinemática directa, cinemática inversa, jacobiano dependiente, etc. También se tienen funciones de más bajo nivel como puede ser la lectura de ficheros o la conversión de mensajes de ROS2 a otras estructuras.

A continuación, se van a listar todas las funciones implementadas (con pseudo-código) para la librería del robot de 4 grados de libertad y se van a explicar.

- PRModel::**InverseKinematics**(&Q, const &X, const &RParam)

Esta función recibe como entrada un vector (std::array) con los valores de la posición en coordenadas cartesianas de la plataforma móvil y resuelve la cinemática inversa a partir de los parámetros de configuración del robot, de forma que actualiza el valor de Q ya que se ha introducido como referencia a la función y esta sería la salida. La resolución de la cinemática inversa es analítica (trivial). Esta función calcula las posiciones de todas las articulaciones del robot (pasivas y activas) a partir de un valor de la posición de la plataforma móvil.

- PRModel::**InverseKinematicsPrism**(&Q, const &X, const &RParam)

Esta función es igual a la anterior pero solamente calcula la posición de las articulaciones activas, esta se utiliza para el generador de trayectorias, ya que no es importante conocer en ese punto la referencia de las articulaciones pasivas.

- PRModel::**ForwardKinematics**(&QA, &X, &X_ant, const &RParam, const tol, const iter_max)

Esta función resuelve la cinemática directa del robot de 4 grados de libertad. En los robots paralelos, la cinemática directa es más complicada de resolver que en los robots serie. En este caso, no se puede resolver analíticamente, entonces se ha implementado un algoritmo de resolución numérica. Este es el algoritmo de Newton-Raphson. Este valor parte de un valor inicial (la posición en coordenadas cartesianas anterior) y a partir de la posición actual en coordenadas de articulaciones, se calcula la nueva posición cartesiana. Este cálculo es iterativo y por eso también se le introduce a la función una tolerancia y un valor de iteraciones máximo. El diagrama de flujo es el siguiente:

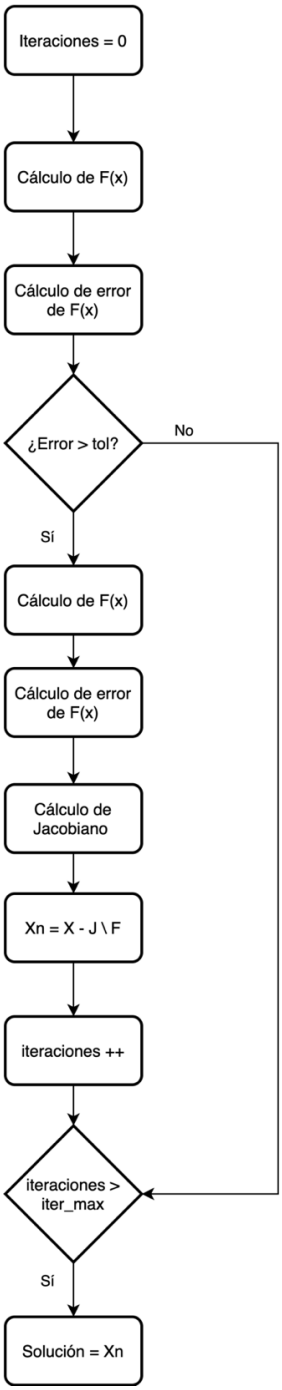


Figura 25: Diagrama de flujo de resolución de la Cinemática Directa

- PRmodel::FK::**EqPosition**(&f, const &X, const &QA, const &RParam)

Esta función se utiliza en la cinemática directa para calcular las ecuaciones de restricción.

- PRModel::FK::**Jacobian**(&FJ, const &X, const &RParam)

Con esta función se calcula también para la cinemática directa el Jacobiano Directo.

- PRModel::**DepJacobian**(&DepJ, const &Q, const &theta, const &psi, const &RParam)

A través de la función DepJacobian se calcula el Jacobiano Dependiente del robot necesario para calcular los términos gravitacionales para una compensación de la gravedad. El cálculo de cada elemento de la matriz Jacobiana es directo.

- PRModel::**IndJacobian**(&IndJ, const &Q)

Esta otra función calcula el Jacobiano Independiente del robot.

- PRModel::**RastT**(&RastT, const &DepJ, const &IndJ)

La función RastT calcula la matriz R* necesaria para el cálculo del vector de términos gravitacionales. Las ecuaciones implementadas son las siguientes:

$$R^*_{[1:11,:]} = -DepJ \backslash IndJ$$

$$R^*_{[12:15,:]} = I_{4x4}$$

$$RastT = R^{*T}$$

- PRModel::**QGravFunction**(&QGrav, const &RastT, const &theta, const &psi, const &Q, const &P)

Con esta función se obtiene el par necesario para compensar el efecto de la gravedad en cada articulación (términos gravitacionales). Esta depende del resultado de la función anterior, del valor de los ángulos Theta y Psi, de la posición de las articulaciones activas y pasivas, y de los parámetros de masas y centro de gravedad de los elementos del robot.

- PRModel::Optitrack::**PosOriPM**(&Coordinates, const &mf1, const &mf2, const &mf3, const &mm1, const &mm2, const &mm3, const &robot)

Esta función permite calcular la posición de la plataforma móvil dada la posición de tres marcadores colocados en la plataforma fija y de tres marcadores colocados en la plataforma móvil del robot. La función también requiere de una entrada que indica si se está utilizando la versión nueva del robot de 4 grados de libertad o la versión anterior, ya que el cálculo es diferente. Finalmente, se obtiene una matriz con las coordenadas de la plataforma móvil (posición y orientación en los 6 grados de libertad).

- PRModel::**ForwardJacobian**(&ForJ, &X, &RParam)

Con esta función se calcula también la matriz jacobiana directa del robot. Se ha utilizado para detectar si el robot está cerca de una singularidad ya que el valor de su determinante en dicho caso se encuentra cercano a 0.

- PRLimits::**LimActuators**()

Devuelve una matriz de tamaño 4x2 con el valor de los límites de cada articulación activa.

Articulación	Límite inferior	Límite superior
q13	0.6537	0.9337
q23	0.6474	0.9274
q33	0.6502	0.9302
q42	0.549	0.829

En todas las articulaciones, el límite superior corresponde con la suma del límite inferior y el valor de desplazamiento máximo (0,28).

- PRLimits::**LimAngles**()

Devuelve un vector de cuatro elementos que contiene los límites de las articulaciones esféricas (30°) y de la articulación universal de la cuarta pata (90°).

- PRSingularity::**CalculateAngOTS**(&theta, &psi, &q, &OTS_ant, &RParam, iter_OTS, tol_OTS)

Esta función calcula los ángulos OTS (ángulo formado por dos ejes instantáneos de giro de un par de torsores cinemáticos de salida) del robot a partir del valor de los ángulos theta, psi (coordenadas cartesianas), de la posición de las articulaciones y de los parámetros geométricos del robot. El cálculo de estos ángulos es iterativo, se ha resuelto con el método de Newton-Raphson de una manera similar a la resolución de la cinemática directa.

- PRSingularity::**EqOTSJacobian**(&J, &wx, &wy, &wz, &theta, &psi, &q, &op, &Rm1, &Rm2, &Rm3, &betaMD, &betaMI)

Esta función calcula el jacobiano de las ecuaciones para resolver los ángulos OTS. El cálculo de los elementos de la matriz es directo y analítico.

- PRSingularity::**EqOTS**(&f, &wx, &wy, &wz, &vx, &vz, &theta, &psi, &q, &op, &Rm1, &Rm2, &Rm3, &betaMD, &betaMI)

Se determinan las componentes de los ángulos OTS. El OTS se selecciona mediante la variable op.

- PRSingularity::**PosibleAngOTS**(&q_ind_mod, &X_cart, &OTS_med, &des_qind, i_qind, &minc_des, &ncomb, &tol_FK, &iter_FK, &iter_OTS, &tol_OTS, &RParam, &Mlim_q_ind, &Vlim_angp)

Se calculan las soluciones de los ángulos OTS para las cuatro combinaciones, devuelve un vector con las soluciones.

- PRSingularity::**CalculateQindModReleaser**(&X_cart, &q_ref, &ang_OTS, &sol_OTS, &minc_des, &fj_det, &RParam, &vc_des, &Mlim_q_ind, &Vlim_angp, &des_qind, &lim_Ang_OTS, &lmin_FJac, &tol_FK, &iter_FK, &tol_OTS, &iter_OTS, &enable)

Calcula los incrementos de posición necesarios a aplicar a la referencia original para que el robot escape de la singularidad de tipo II. A partir del mínimo ángulo OTS, se seleccionan las patas que están involucradas en la singularidad. Posteriormente, se generan ocho modificaciones de la referencia a partir de estas patas: Los dos actuadores se extienden, los dos se retraen, el primero se extiende y el segundo se retrae y viceversa, las otras cuatro combinaciones consisten en la extensión o retracción de una pata mientras que la otra que causa la singularidad no se mueve.

Para cada modificación, se determina el ángulo omega que se obtendrá mediante el siguiente proceso: Se resuelve la cinemática directa, después la cinemática inversa, solución del par de OTS involucrados en la singularidad y finalmente se calcula el ángulo omega. De las combinaciones anteriores, se escoge la que produzca el máximo ángulo omega para los OTS involucrados.

- PRUtils::**read_file**(&ref_matrix, const &file_path)

Esta función perteneciente a la librería pr_utils se utiliza para leer un fichero de referencias y guardar todos sus datos en una matriz. Entonces, en primer lugar, abre el fichero indicado en el argumento file_path y rellena la matriz dada como argumento (como referencia). Si no ha encontrado el fichero, devuelve un error, -1, si lo ha encontrado devuelve 1.

- PRUtils::**vector2matrix**(&matrix, &vec)

Convierte una matriz modelada como un objeto std::vector que contiene más objetos std::vector de manera que forman una estructura bidimensional de datos a una matriz modelada con la librería Eigen. Esta función se utiliza en la función anterior read_file.

En la libería desarrollada para el robot de 3 grados de libertad (pr_3dof_lib) las funciones y clases implementadas son las siguientes:

- PR3DOFModel::**InverseKinematics**(&retC3PRS, const &xx, const &yy, const &lmlrlb)

La función calcula la cinemática inversa del robot. Es decir, la posición de las articulaciones activas a partir de la posición cartesiana (vector xx). Como parámetros se requiere del vector yy y lmlrlb que son parámetros geométricos del robot. La solución de la cinemática inversa, tal como sucedía en el robot paralelo de cuatro grados de libertad, es también analítica.

- PR3DOFModel::**GenRefQ**(const &gamma, const &beta, const &zeta, const &lmlrlb, &q)

Esta función calcula también la cinemática inversa del robot, a partir del valor de las coordenadas cartesianas independientes de la plataforma del robot (gamma, beta y zeta). Como parámetros se requiere del vector geométrico lmlrlb. Esta función tiene en cuenta que la altura del robot en la posición en reposo con respecto al eje de coordenadas es 0.19191484 que la longitud de la barra de los actuadores es de 0.725. De esta manera, a la hora de diseñar una trayectoria cartesiana, la posición en reposo del robot corresponde con el punto z=0 y esto también corresponde con una longitud de las articulaciones activas de 0.

- PR3DOFModel::**Jacobian**(&Jac, &yy, &lmlrlb, &xx)

Esta función se utiliza para calcular el jacobiano directo del robot. Este tiene un tamaño 3x3 cuyo cálculo de cada elemento es directo a partir de los parámetros proporcionados.

- PR3DOFModel::**ForwardKinematics**(&Qind, &lmlrlb, &q2, &q7, &q9, &id2, &j, &retQs, tol, iter_max)

Mediante esta función se resuelve el problema de la cinemática directa de posición del robot de 3 grados de libertad. El procedimiento es el mismo que la función descrita para el robot de 4 DOF ya que esta también implementa el algoritmo de Newton-Raphson.

- PR3DOFModel::**QGravTerms**(&q, g, &lmlrlb, &Ae, &Ai, &phi1, &phi2, &phi3, &retG)

Calcula los términos gravitacionales en par necesarios para compoensar el efecto de la gravedad de cada articulación activa.

- PR3DOFModel::**fcn**(&q, &lmlrlb, &ret_Xe, &ret_Xi)

Obtiene las matrices Ae y Ai que se utilizan para el cálculo de términos dinámicos del robot (términos gravitacionales, coriolis, centrífugos e inerciales).

- PR3DOFModel::**Vel3DOFEmb**(&qp, &Ae, &Ai, &DQS)

Resuelve la cinemática de velocidades del robot.

- PR3DOFModel::**InerEmbCorr**(&q, &a, &Ae, &Ai, &lmlrlb, &phi010, &phi016, &phi017, &J, &ine0)

Esta función calcula los términos inerciales del robot.

- PR3DOFModel::**CEmbCorr**(&q, &dq, &Ae, &Ai, &lmlrlb, &phi010, &phi016, &phi017, &Fv, &Fc, &Cv)

Calcula los términos de par necesarios a aplicar en las articulaciones activas para contrarrestar los efectos de coriolis y centrífugos.

- PR3DOFModel::**ForwardJacobian::calculate**(&Jacobian, &force_control, &QS1, &lmlrlb)

Resuelve la matriz jacobiana directa.

- PR3DOFUtils::**Spline::Spline**(&init_point, &final_point, &n_iter)

Esta clase se utiliza para modelar una curva Spline. En el constructor se indica el punto inicial, el punto final y el número de iteraciones. Este constructor calcula los coeficientes de la Spline.

- PR3DOFUtils::Spline::**calculate_value**(&time)

Se calcula el valor de la salida de la Spline para un valor de entrada comprendido entre 0 y 1, si se introduce 0, se obtiene el primer punto con el que se ha construido la curva y si se introduce 1 se obtiene el último punto.

5.2.4. COMPONENTES PARA LOS SENSORES Y ACTUADORES DEL ROBOT

Se han creado los paquetes pr_sensors_actuators y pr_3dof_sensors_actuators para construir los componentes relacionados con los interfaces físicos de los robots, así como de sus versiones simuladas. Los componentes relacionados con el sistema de captura de movimiento se han separado en otro paquete que se ha llamado pr_mocap. Los dos primeros paquetes requieren como dependencias la librería BIODAQ de Advantech y el último la NatNet SDK.

5.2.4.1. ENCODERS

Se ha desarrollado un componente de ROS2 en lenguaje C++ para realizar la lectura de los cuatro encoders y publicar el valor de posición. El componente está modelado también por una clase.

En el constructor de la clase se realiza la configuración inicial, en la que se declaran dos parámetros. Uno es el tiempo de muestreo y el segundo es el valor inicial de posición. Este último valor es necesario ya que a través de los encoders se obtiene una medida incremental a partir del valor en el que inicialmente se encuentran los actuadores. Posteriormente, se crea el publicador que consiste en un mensaje del tipo PRArrayH. También se crea un subscriptor para que este nodo conozca en qué momento se ha terminado la trayectoria, esto es necesario puesto que este componente también se encarga de desactivar los frenos al comenzar la trayectoria y de activarlos al terminar.

Posteriormente, se realiza la configuración de las entradas digitales. Se instancian los objetos necesarios para el acceso a las entradas digitales y se selecciona el dispositivo. También se configura el conteo de los pulsos en modo “AbPhaseX1”. Por otro lado, también se instancia y configura una salida digital para manejar los frenos del robot y se desactivan para que el robot pueda moverse.

Otra cuestión importante es que en el constructor de la clase se crea también un temporizador cuyo valor de tiempo será el valor del parámetro del periodo de muestreo. Este temporizador tiene adjunto un callback que se ejecuta cada vez que vence el tiempo. En este callback se comprueba si la trayectoria ha finalizado. Si no lo ha hecho, se lee el valor de los encoders, se convierte el valor a un valor de posición y este se publica como un mensaje PRArrayH que incluye un valor de tiempo actual y el valor de la posición de las cuatro articulaciones. Entonces, el componente de los encoders es el que activa el periodo de muestreo. Se activa el callback del timer, se publica la información de posición y los siguientes nodos que estén suscritos, activarán sus respectivas callbacks, publicando también información hasta llegar a los actuadores y terminar con la cadena.

En este componente se conoce si la trayectoria ha finalizado o no puesto que hay un callback asociado a un topic que indica esta información. Si la trayectoria ha terminado, en el callback del temporizador, se activan los frenos.

En el siguiente esquema se puede observar la estructura del nodo de una manera más visual:

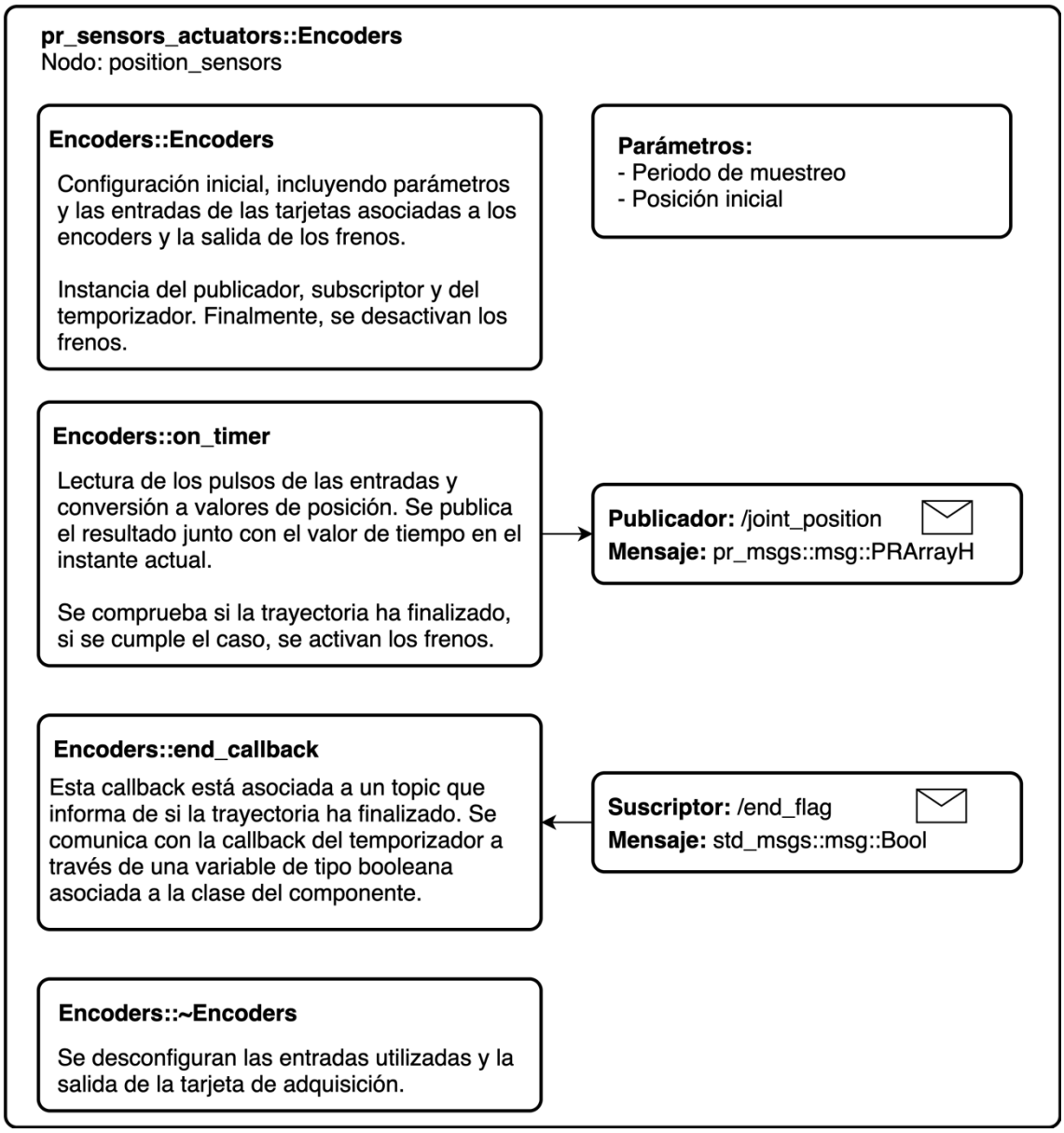


Figura 26: Esquema de componente de los Encoders

Este nodo es muy similar en el robot de tres grados de libertad. En este, como tiene una articulación menos, se realiza la lectura de tres encoders y como el robot no tiene freno, se elimina la parte que gestiona el freno del nodo.

También se ha desarrollado una versión simulada de este nodo cuya función es comunicar el modelo de Simulink que se tiene del robot. Por tanto, se elimina el acceso a la tarjeta de adquisición de datos y se sustituye este acceso por una comunicación entre Simulink y este nodo basada en topics de ROS2. Esto es posible porque Matlab dispone de una toolbox en la que se ha implementado funciones y bloques de Simulink para crear nodos, publicadores, suscriptores, etc. Esta toolbox es la “ROS Toolbox”.

5.2.4.2. SENSOR DE FUERZA

Como se ha mencionado en el apartado de hardware, para acceder a los valores del sensor de fuerza, es necesario establecer una comunicación UDP. Así pues, el componente relacionado con el sensor de fuerza está basado en una comunicación de este tipo desarrollada en C++.

En el constructor del componente, de nuevo, se realiza la configuración inicial. En este caso, se configura y se crea un socket UDP para comunicar el ordenador de control con el sensor de fuerza. A este socket se le añade un tiempo límite con el fin de que si se está esperando a recibir un mensaje, pero no se recibe, que la función no bloquee el callback. Se detecta el error y se pide de nuevo el dato en la siguiente iteración.

Este nodo tiene asociado un temporizador, entonces es independiente del temporizador del nodo de los encoders. Esto es porque con el fin de realizar estudios, se quiere que tener valores de fuerza con más frecuencia que los valores de posición.

Entonces, en el callback del timer, se envía un mensaje de petición y se espera a la respuesta que contiene los valores del sensor de fuerza. Estos valores son tres momentos y tres fuerzas. Entonces se construye el mensaje PRForceState con estos datos y se publica.

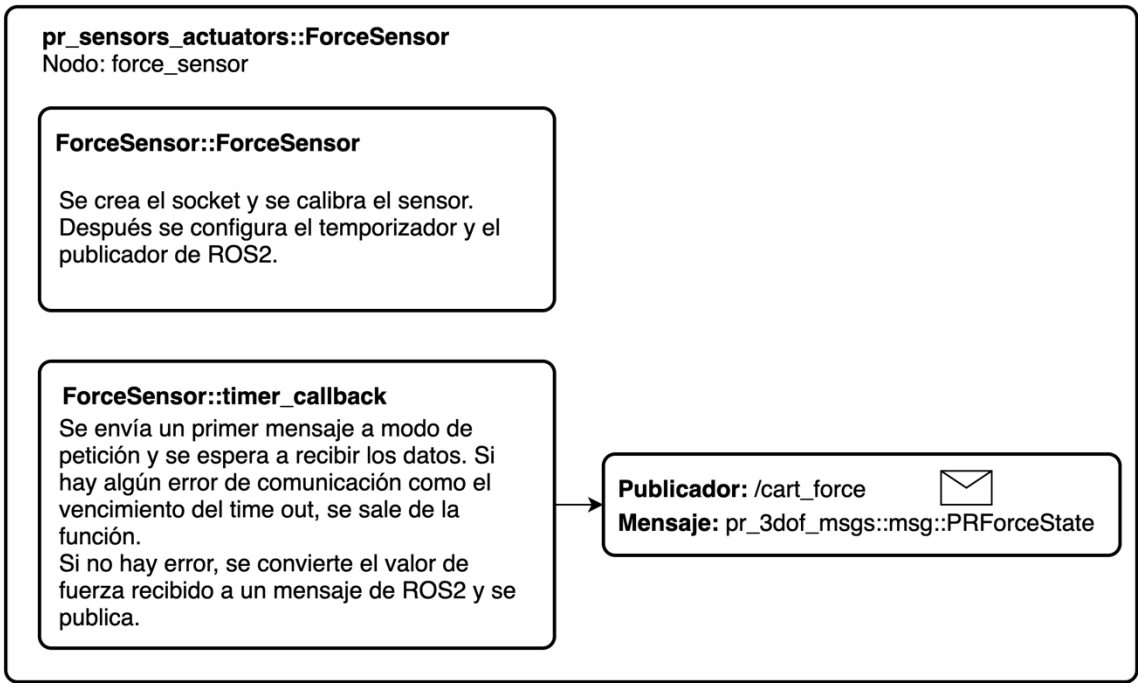


Figura 27: Esquema de componente del sensor de fuerza

5.2.4.3. SISTEMA DE CAPTURA DE MOVIMIENTO

Con el fin de poder realimentar el control con los datos de posición de los marcadores detectados por las cámaras, se ha desarrollado un componente cuya función es acceder a la red de las cámaras y publicar la posición en coordenadas cartesianas de la plataforma móvil del robot.

El componente construido pertenece al paquete pr_mocap y requiere como dependencia el kit de desarrollo de software de NatNet [16]. Esta SDK tiene una clase cliente con la que se puede acceder a los datos ofrecidos por las cámaras.

Así pues, en el constructor de la clase se declaran una serie de parámetros relacionados con la dirección IP del servidor, los puertos de comandos y de datos, el nombre de los marcadores del robot y el robot que se esté utilizando puesto que la posición de la plataforma móvil se calcula de una manera diferente si se está trabajando con la versión anterior o la nueva del robot de 4 grados de libertad.

Una vez leídos los parámetros se crea el publicador de la posición móvil de la plataforma. Después se instancia un cliente de NatNet y se adjunta a este objeto un callback que se ejecutará cada vez que haya un nuevo frame de datos de las cámaras. Se añaden los parámetros de la comunicación al cliente y se conecta. Posteriormente, se envía una petición para obtener las descripciones de los datos de Motive. De esta información se necesita el nombre de los marcadores para conocer en qué orden está y poder buscarlos posteriormente.

Una vez construido el programa, se ejecuta el callback asociado al cliente cada vez que hay un nuevo frame de datos. En este callback, se accede al valor de las coordenadas de x, y, z de cada marcador colocado en el robot y a partir de estos datos, se calcula la posición de la plataforma móvil. Una vez calculada, se publica a través de un mensaje PRMocap al topic x_coord_mocap de ROS2. Por último, en el destructor se desconecta y se elimina el cliente.

A continuación, se muestra el esquema del nodo:

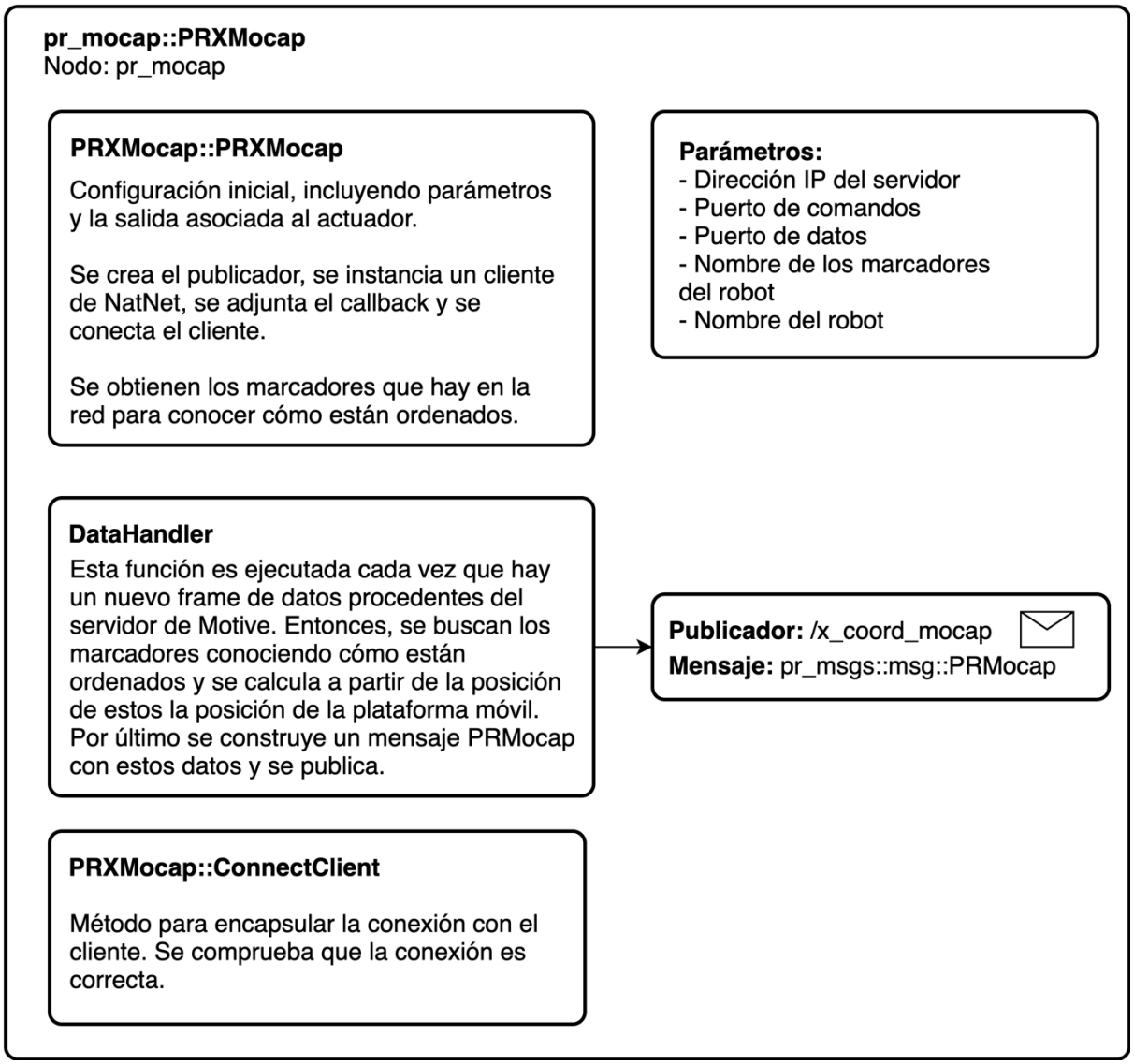


Figura 28: Esquema del componente de la captura de movimiento PRXMocap

5.2.4.4. ACTUADORES

Para el acceso a los actuadores se ha desarrollado otro componente a partir de una clase de C++. En el constructor se realiza la configuración inicial, se crean y leen dos parámetros, el voltaje de saturación y la conversión de par a voltaje. Esta es necesaria puesto que algunos actuadores emiten la acción de control en valores de par en vez de voltios. Después se crea la comunicación de ROS2. Se instancia una suscripción para recibir la acción de control y otra suscripción para saber si la trayectoria ha terminado. Por último, se configura la salida de la tarjeta y se escriben 0 Voltios.

En el callback asociado a la suscripción de la acción de control, si la trayectoria no ha finalizado, se obtiene el valor de la acción de control del actuador y se convierte a voltaje. Una vez convertido, se satura el valor si está por encima o por debajo del límite marcado por el parámetro de la saturación. Finalmente se escribe el voltaje en la salida.

Se tiene también otro callback asociado al topic que informa de que la trayectoria ha terminado. Si esta ha terminado, se escriben 0 voltios en la salida. Por último, en el destructor de la clase, se vuelven a escribir 0 voltios y se desconfigura la salida.

El esquema del nodo es el siguiente:

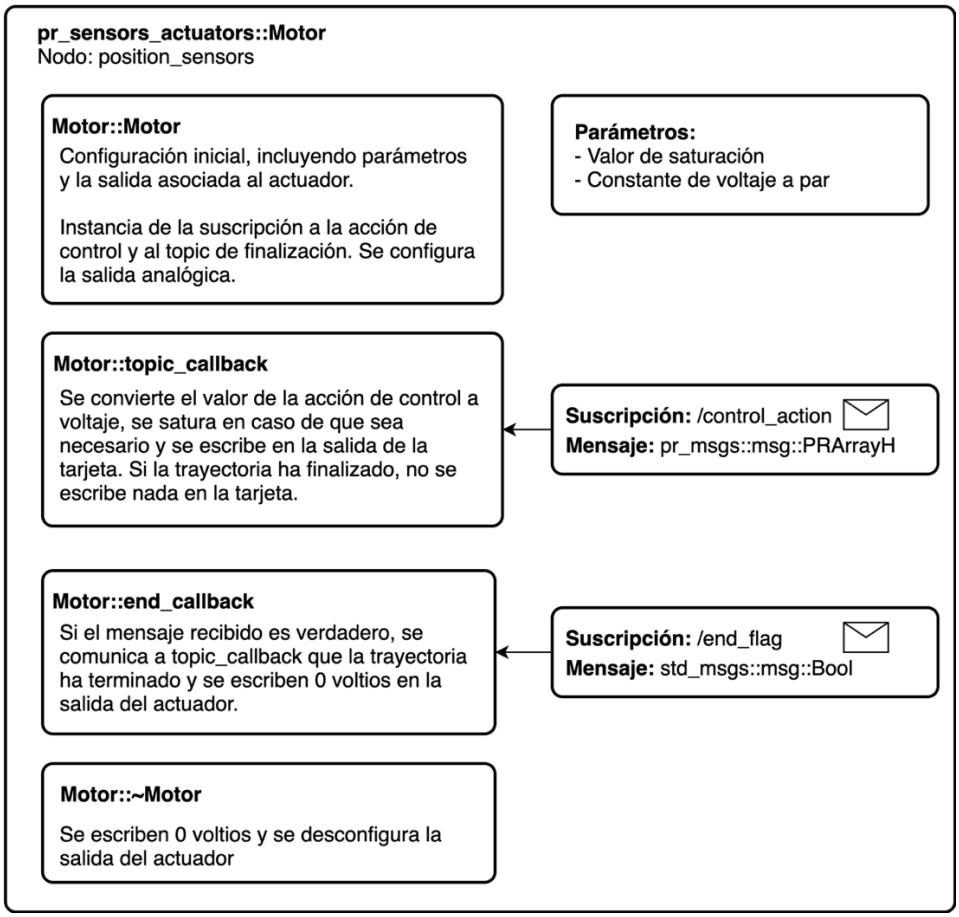


Figura 29: Esquema del componente del actuador, Motor

La única diferencia entre la implementación para el robot de 4 grados de libertad y el de 3 es el tipo de mensaje que se utiliza en el topic de la acción de control. Uno pertenece al paquete pr_msgs y el otro al paquete pr_3dof_msgs.

Entonces, este nodo solamente controla un actuador, pero se puede instanciar tantas veces como se necesite para abarcar los cuatro actuadores de un robot y los tres del otro. Cuando se instancia, es necesario renombrar el nodo de forma que el primer actuador es motor_1, el segundo motor_2 y así sucesivamente.

Como en el nodo de los encoders, también se ha creado una versión simulada que lee la información del topic de la acción de control /control_action y publica el valor en voltaje por un topic para comunicar esta acción con simulink. El nodo simulado abarca todos los actuadores, no son necesarias varias instancias como en el anterior.

5.2.5. GENERADOR DE REFERENCIAS

Para generar las referencias de una trayectoria en cada iteración del bucle de control se ha implementado otro componente de ROS2. Este componente pertenece al paquete pr_ref_gen y es una clase de C++.

Para el robot de 4 grados de libertad, en el constructor se declaran y se leen los parámetros necesarios. Posteriormente, se lee el fichero que contiene los valores de referencia de la trayectoria y se guardan todos los valores en una variable. El fichero puede tener los valores de referencia en coordenadas de articulaciones o en el espacio cartesiano. Entonces hay un parámetro que indica si estas referencias están en un espacio o en otro. En caso de que estén en espacio cartesiano, estas se convierten a través de la cinemática inversa.

Después se crean los publicadores y suscriptores de ROS2. Es necesario una suscripción al valor emitido por los encoders (topic /joint_position) puesto que el nodo de los encoders es el que maneja el tiempo del periodo de control. Así pues, se emite un nuevo valor de referencia cada vez que se recibe un nuevo valor de posición. Se crea también el publicador de la referencia y el publicador para informar al resto de nodos de que la trayectoria ha finalizado.

Entonces, como se ha mencionado anteriormente, en el callback asociado a la suscripción de /joint_position, se envía un nuevo valor de referencias. A través de una variable de la clase, se realiza el conteo de las referencias e incrementa cada vez que se ejecuta el callback. Si esta variable alcanza el valor del número de referencias totales que contenía el fichero de la trayectoria, se dejan de publicar referencias y se publica un valor true por el topic que informa de que la trayectoria ha terminado.

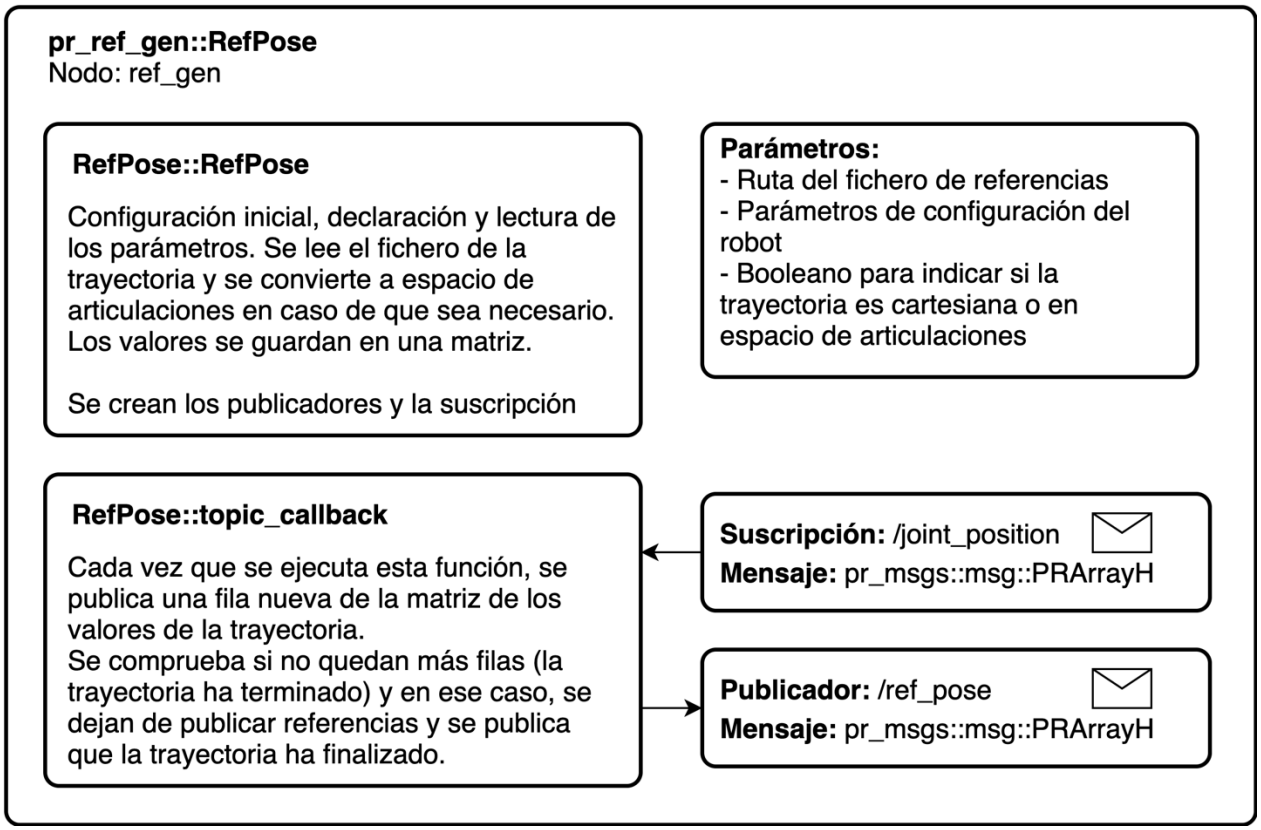


Figura 30: Esquema del componente del generador de referencias de posición, RefPose

La versión implementada para el robot de 3 grados de libertad es algo diferente. En este robot, todas las trayectorias comienzan a partir de un punto que no es la posición en la que se encuentra el robot (posición de reposo). Entonces hay que generar una trayectoria desde la posición de reposo al primer punto de la trayectoria del fichero. Esta trayectoria se ha definido como una curva Spline. En el constructor se resuelven los coeficientes de la Spline a partir de la posición en reposo del robot y de la primera posición del fichero. También es necesario resolver una segunda curva Spline entre la última posición del fichero de referencias y la posición en reposo del robot.

Entonces, durante las primeras iteraciones de la función topic_callback, se envían referencias que se corresponden con los puntos de la Spline. Cuando se llega al primer punto de la referencia, se comienzan a enviar los puntos del fichero de trayectorias. Después de que estos se acaben, a partir de la segunda Spline, se devuelve al robot a la posición de reposo.

En la siguiente imagen se puede visualizar un ejemplo de trayectoria en espacio de articulaciones:

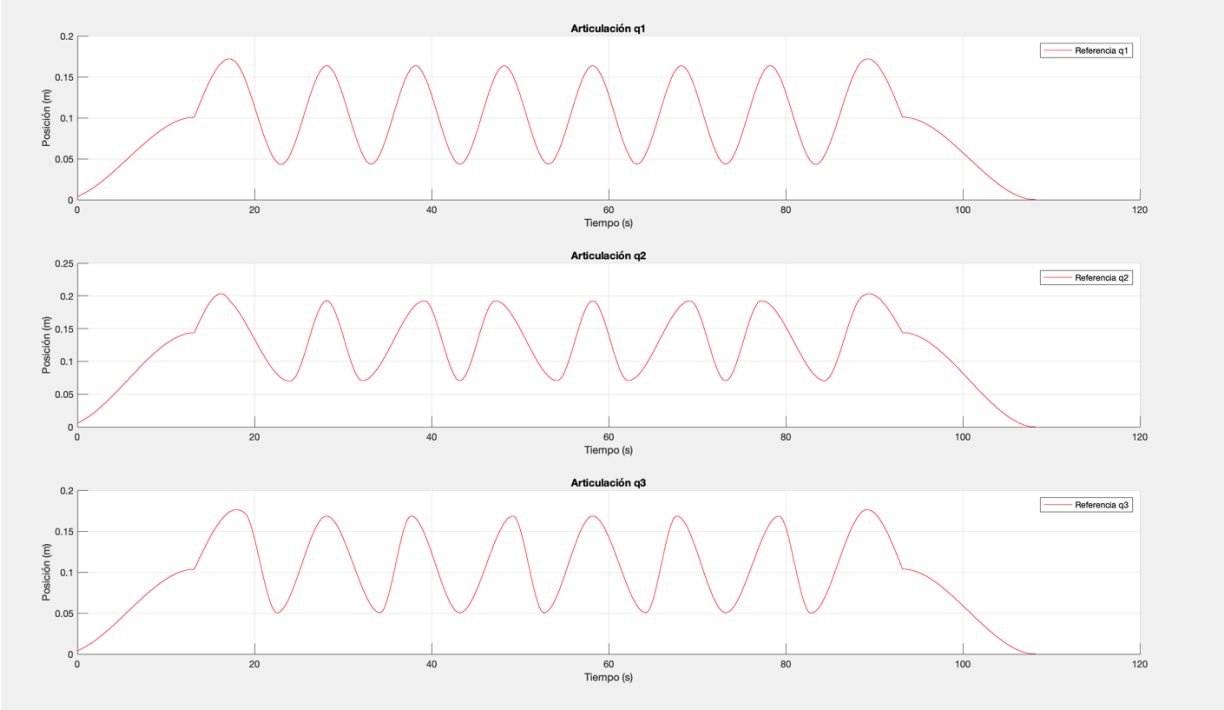


Figura 31: Ejemplo de trayectoria para el robot de 3DOF con una curva spline al comienzo y al final de la trayectoria

Como se han desarrollado también controladores de fuerza para el robot de 3 grados de libertad, el generador de referencia de este robot no solamente se puede utilizar para generar referencias de posición si no también de fuerza. El funcionamiento sería el mismo, se lee el fichero de referencias de fuerza y se almacenan los datos en una matriz que después, en el callback se recorren comenzando cuando termina la primera Spline.

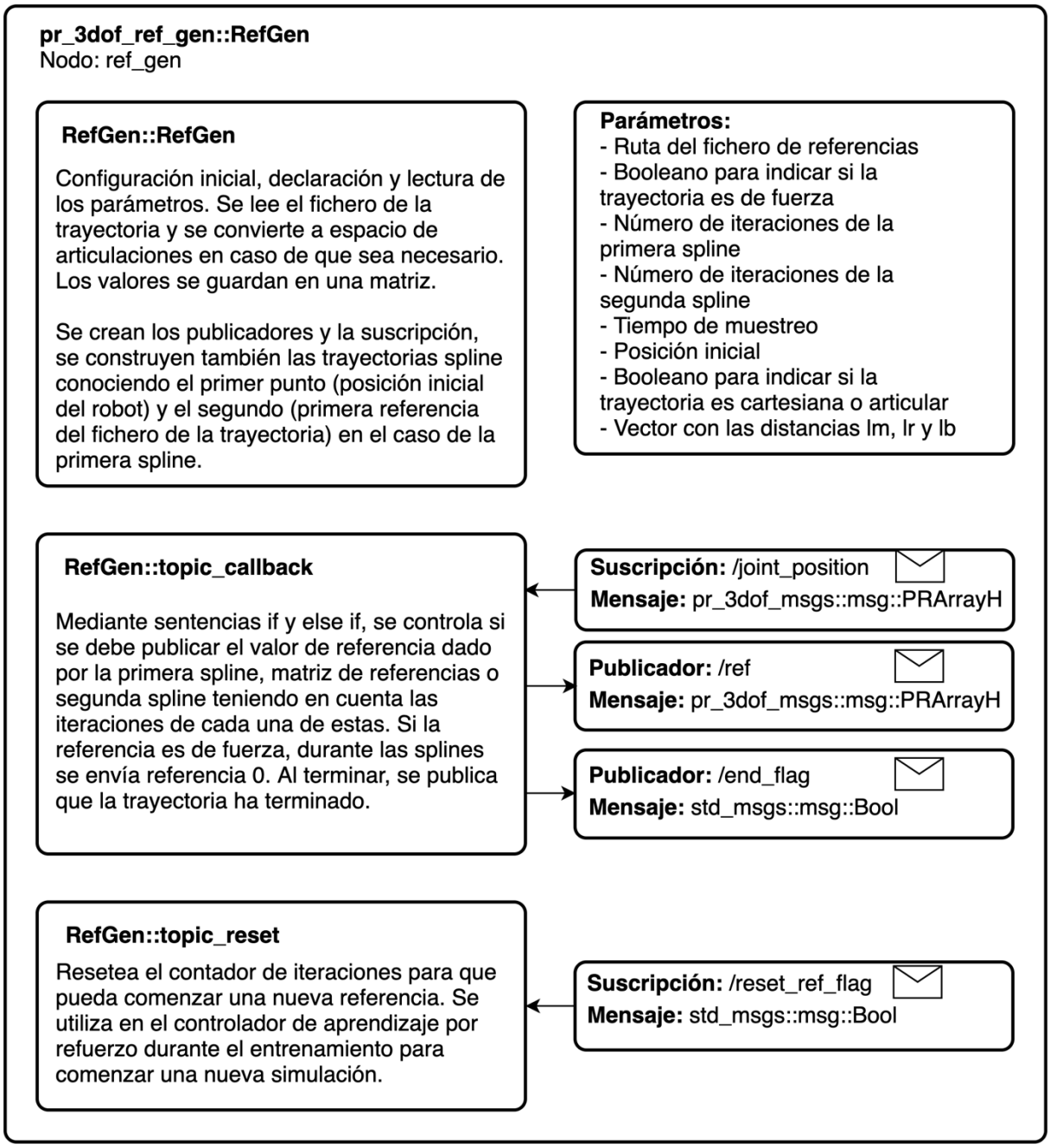


Figura 32: Esquema del componente del generador de referencias genérico para el robot de 3DOF

5.2.6. COMPONENTES AUXILIARES

Se ha creado un paquete para desarrollar componentes auxiliares tanto para los robots de 4 grados de libertad (pr_aux) como para el de 3 grados (pr_3dof_aux). Se han incluido componentes que hacen operaciones matemáticas que son independientes del modelo del robot, como puede ser un derivador, integrador, suma/resta, multiplicación, etc. A continuación, se muestra una lista con los componentes:

- Derivador

Este componente realiza una derivada discreta, necesita como parámetros el tiempo de muestreo y la posición inicial. Se suscribe a un topic del tipo pr_msgs::msg::PRArrayH en el caso de 4DOF y pr_3dof_msgs::msg::PRArrayH en el caso de 3DOF. En el callback calcula la derivada y la publica a través de un mensaje del mismo tipo. La ecuación de la derivada implemntada es la siguiente:

$$salida = \frac{valor_{actual} - valor_{anterior}}{T_s}$$

También tiene un callback que resetea el valor anterior (lo iguala a 0). Este se utiliza en aprendizaje por refuerzo para comenzar una nueva simulación.

- Integrador

Se ha implementado otro nodo que hace la función de integrar una variable, como parámetros necesita el valor inicial y el tiempo de muestreo. La operación que realiza es la siguiente:

$$salida = valor_{acumulado} + T_s \cdot (valor_{actual} - valor_{previo})$$

- GainAdd

Este nodo se suscribe a dos topics (vectores de 3 ó de 4 componentes) y cuando recibe los valores, los multiplica por una ganancia (que puede ser distinta para cada vector) y finalmente los suma. La operación es la siguiente:

$$salida = g_1 \cdot entrada_1 + g_2 \cdot entrada_2$$

Las ganancias se introducen como un parámetro del nodo y está en forma de vector de dos elementos.

- AddTerms

Se suscribe a tres variables, las suma y publica su valor. Se ha utilizado en el robot de tres grados de libertad para implementar un controlador por dinámica inversa y este componente suma los términos gravitacionales, de coriolis e inerciales.

- Replayer

Este nodo es diferente a los anteriores en el sentido que no realiza una operación matemática. Se ha utilizado este componente para reproducir experimentos y verificar que la acción de control emitida es la correcta. Así pues, este componente es similar al generador de trayectorias. En el constructor lee un fichero generalmente con posiciones (posiciones reales de un experimento o simuladas). Después crea un temporizador, y en su callback recorre los valores medidos y los publica en cada iteración.

5.2.7. COMPONENTES RELACIONADOS CON EL MODELO DEL ROBOT

Se han construido paquetes para implementar componentes que realizan operaciones relacionadas con el modelo de los robots. Estos componentes utilizan las librerías asociadas a cada robot pr_lib o pr_3dof_lib para realizar los cálculos. Entonces, la implementación del modelo realmente se encuentra en estas librerías, mientras que estos componentes implementan la interfaz de ROS2. La ventaja de haber desarrollado una librería es que esta es independiente del framework que se utilice. Si en un futuro se desea utilizar otro framework distinto a ROS2, siempre que este utilice el lenguaje C++, se podrán utilizar las librerías.

5.2.7.1. ROBOT DE 4 GRADOS DE LIBERTAD

Los componentes relacionados con el modelo del robot de 4 grados de libertad se han implementado en el paquete llamado pr_modelling. Cada uno de estos utiliza la función correspondiente de la librería. En la siguiente tabla se muestran los componentes implementados con sus suscripciones y publicaciones (nombres por defecto):

Nombre del componente	**Explicación**	**Suscripciones**	**Publicaciones**	**Parámetros**
ForwardKinematics for_kin	Bloque que calcula la cinemática directa.	joint_position	x_coord	- Configuración del robot - Posición inicial - Tolerancia - Iteraciones máximas
InverseKinematics inv_kin	Calcula la cinemática inversa, se obtiene la posición de las articulaciones pasivas y activas.	x_coord	q_sol	- Configuración del robot

ForwardJacobian for_jac	Calcula el determinante del jacobiano directo.	x_coord	for_jac_det	- Configuración del robot
DependentJacobian dep_jac	Publica el jacobiano dependiente.	x_coord q_sol	dep_jac	- Configuración del robot
IndependentJacobian ind_jac	Publica el jacobiano independiente.	q_sol	ind_jac	
RastT rast_t	Calcula el complemento ortogonal entre el jacobiano dependiente e independiente.	dep_jac ind_jac	rast_t	
QGrav q_grav	Calcula los términos gravitacionales en cada articulación a partir de parámetros relacionados con centros de gravedades y masas.	x_coord q_sol rast_t	q_grav	- p11 - p12 - p21 - p22 - p31 - p32 - p41 - p42 - pm
AngOTS ang_ots	Obtiene los ángulos OTS a partir de un método de cálculo iterativo.	x_coord	ang_ots	- Configuración del robot - Iteraciones máximas - Tolerancia - Ángulos iniciales - Coordenadas cartesianas iniciales
StatePublisher state_publisher	Recoge la información del estado del robot y la publica.	joint_position joint_velocity	pr_state	- pos_max - pos_min - vel_max - vel_min

Tabla 4: Componentes relacionados con el modelo del robot de 4DOF

5.2.7.2. ROBOT DE 3 GRADOS DE LIBERTAD

Los componentes implementados para el modelo del robot de 3 grados de libertad se han organizado en el paquete pr_3dof_modelling y utilizan funciones y clases de la librería pr_3dof_lib.

Nombre del componente	Explicación	Suscripciones	Publicaciones	Parámetros
ForwardKinematics for_kin	Bloque que calcula la cinemática directa.	joint_position	x_coord	- lmlrlb - q2q7q9 - Posición Inicial - Tolerancia - Iteraciones máximas
InverseKinematicsGBZ inv_kin_gbz	Calcula la cinemática inversa, se obtiene la posición de las articulaciones activas.	x_gbz_coord	q_coord	- lmlrlb
FCN fcn	Calcula las matrices Ae y Ai necesarias para el cálculo de los términos gravitacionales, de coriolis e inerciales.	x_coord	ae_ai	- lmlrlb
QGrav q_grav	Calcula los términos gravitacionales en cada articulación a partir de parámetros relacionados con centros de gravedades y masas.	x_coord ae_ai	q_grav	- g - phi_10_16_17
VelKin vel_kin	Implementa la cinemática de velocidad del robot.	joint_velocity ae_ai	vel_x	
CorrEmb corr_emb	Calcula los términos de coriolis y centrífugos.	vel_x x_coord ae_ai	corr_terms	- lmlrlb - phi_10_16_17 - fv - fc

InerEmb iner_emb	Cálcula los términos interciales.	joint_acceleration x_coord ae_ai	iner_terms	- lmlrlb - phi_10_16_17 - J
ForJacPos for_jac_pos	Obtiene primero la velocidad de las articulaciones a partir de la velocidad cartesiana y de la matriz jacobiana. Después integra este valor y publica la posición.	x_vel x_coord	ref_pose_inc	- lmlrlb - Periodo de muestreo
StatePublisher state_publisher	Publica el estado del robot (posición y velocidad de las articulaciones).	joint_position joint_velocity	pr_state	- pos_max - pos_min - vel_max - vel_min
StatePDPublisher	Publica el estado del robot, añade el error de posición y el error de velocidad.	joint_position joint_velocity ref_pose ref_vel	pr_pd_state	- pos_max - pos_min - vel_max - vel_min

Tabla 5: Componentes relacionados con el robot de 3DOF

5.2.8. COMPONENTES RELACIONADOS CON LOS CONTROLADORES

Se han creado componentes para ambos robots que recogen implementan las operaciones básicas de suma y resta para calcular la acción de control final a partir de términos ya calculados anteriormente. Estos se han creado en los paquetes llamados pr_controllers y pr_3dof_controllers.

Con más profundidad, el componente PDGController, en ambos robots se suscribe a la posición, velocidad, referencia y términos gravitacionales y aplica la siguiente ecuación:

$$u = k_p \cdot (q_{ref} - q) - K_v \cdot q_p + q_{grav}$$

Una vez calculada, publica en un mensaje de ROS2 que contiene un vector con 4 ó 3 elementos en función del robot la información correspondiente a la acción de control de cada actuador. Estos componentes tienen parámetros relacionados con las ganancias del controlador.

Otro componente implementa un controlador algebraico cuya ecuación se explicará en los siguientes apartados. Para el robot de 3 grados de libertad se ha

implementado un controlador de fuerza basado en un modelo de admitancia cuyo componente es AdmittanceController. En este componente, primero en el suscriptor se calcula el modelo discreto, ya que los parámetros se introducen en el modelo continuo para que sean más intuitivos de modificar. Posteriormente, en la callback asociada a las suscripciones con message filters, se actualiza el valor de la salida a partir de la entrada y se publica. Esto se explicará también con más profundidad en los próximos apartados.

5.2.9. CONTROLADORES IMPLEMENTADOS

5.2.9.1. PD CON COMPENSACIÓN DE LA GRAVEDAD

Se ha implementado un controlador PD con compensación de la gravedad para el robot de 4 grados de libertad y otro para el robot de 3 grados. La ecuación que define la acción de control es la siguiente (siendo G(q) los términos de voltaje que anula el efecto de la gravedad):

$$u = K_p \cdot (q_{ref} - q) - k_v \cdot q_p + G(q)$$

El esquema del controlador dinámico es el siguiente:

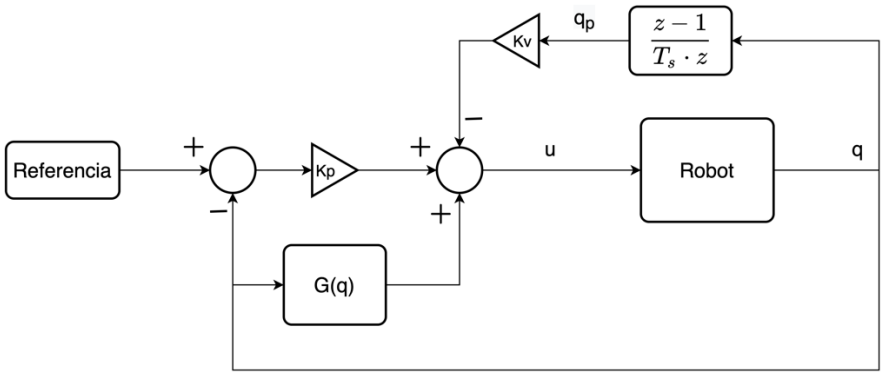


Figura 33: Diagrama dinámico de un controlador PD+G

El controlador PD+G es adecuado para ambos robots puesto que es preciso y suave, además, no se requiere de términos inerciales ni de coriolis puesto que estos robots realizan movimientos lentos en los que la dinámica inercial y de coriolis se puede despreciar en el controlador.

En el siguiente esquema se muestran los nodos que forman el controlador para el robot 4DOF así como de los topics que intercambian información.

4 DOF

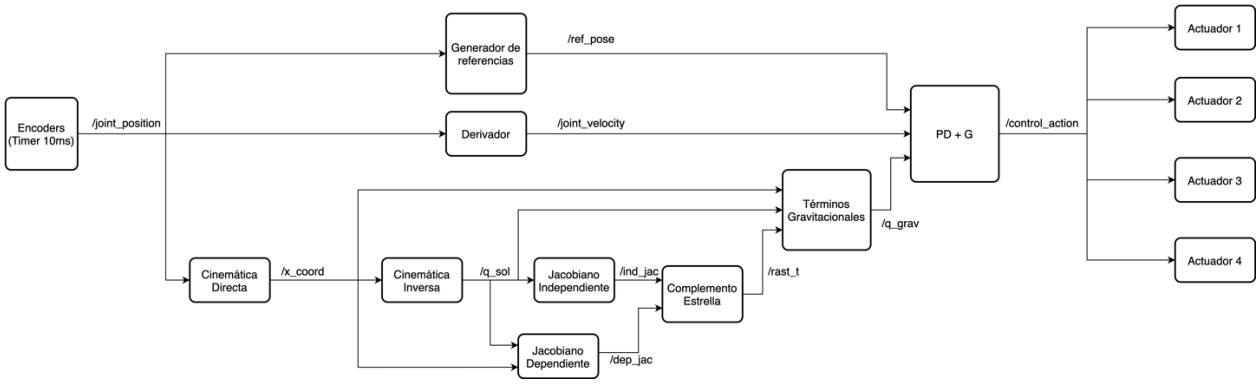


Figura 34: Esquema de ROS2 del controlador PD+G para el robot de 4DOF

La implementación para el robot de 3DOF es la siguiente:

3 DOF

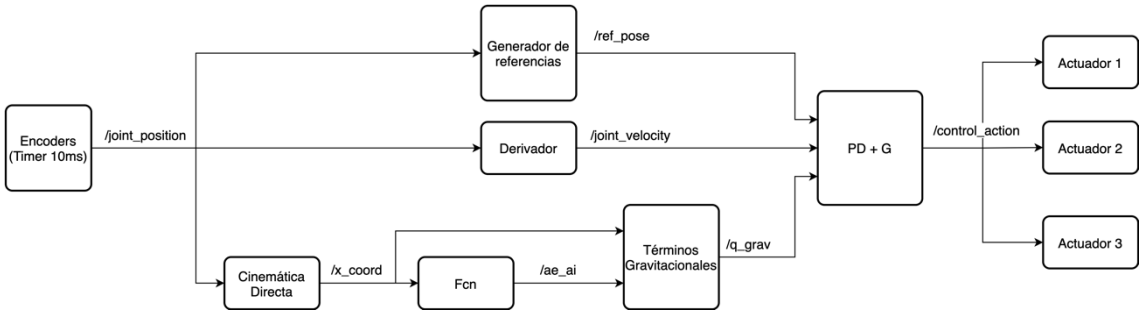


Figura 35: Esquema de ROS2 del controlador PD+G para el robot de 3DOF

5.2.9.2. CONTROLADOR ALGEBRAICO

El controlador algebraico de posición se basa en las siguientes ecuaciones discretas:

$$u_1 = \frac{q_{ref} - k_1 \cdot \left(q_{ref(k-1)} - q_{(k-1)}\right) - q_{ref(k-1)}}{T_s}$$

$$u = \frac{u_1 - k_2 \cdot \left(u_{1(k-1)} - q_p\right) - q_p}{T_s}$$

Las ganancias de este son:

$$k_1 = 0.75$$

$$k_2 = 0.5$$

En el siguiente esquema se muestra el diagrama dinámico del controlador. El bloque del controlador Algebraico incluye las ecuaciones descritas anteriormente.

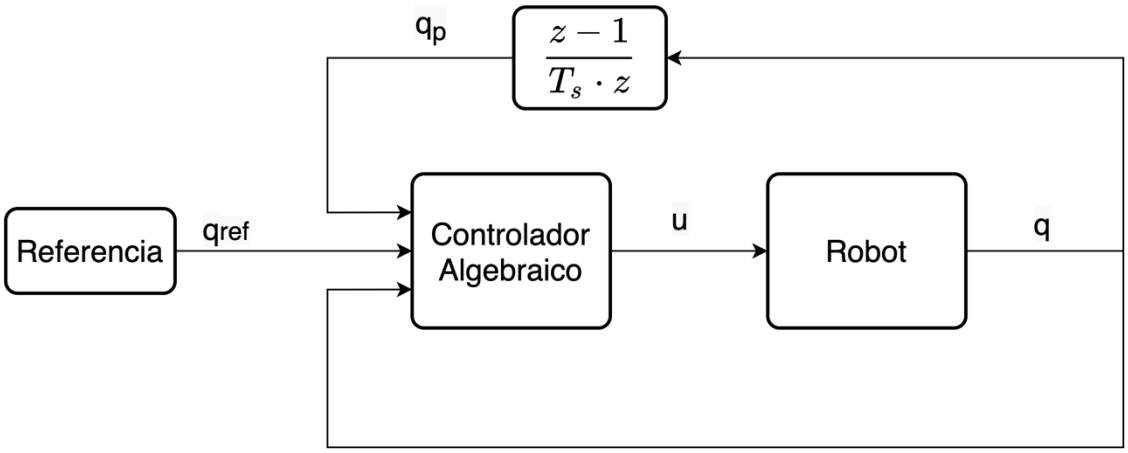


Figura 36: Diagrama dinámico de un controlador algebraico

En la siguiente figura se representa el diagrama de nodos y de topics de ROS2 que componen el controlador. La modularidad de la implementación permite que los componentes Encoders, Generador de referencias, Derivador y los Actuadores sean exactamente los mismos que los del controlador anterior para el robot de 4 grados de libertad.

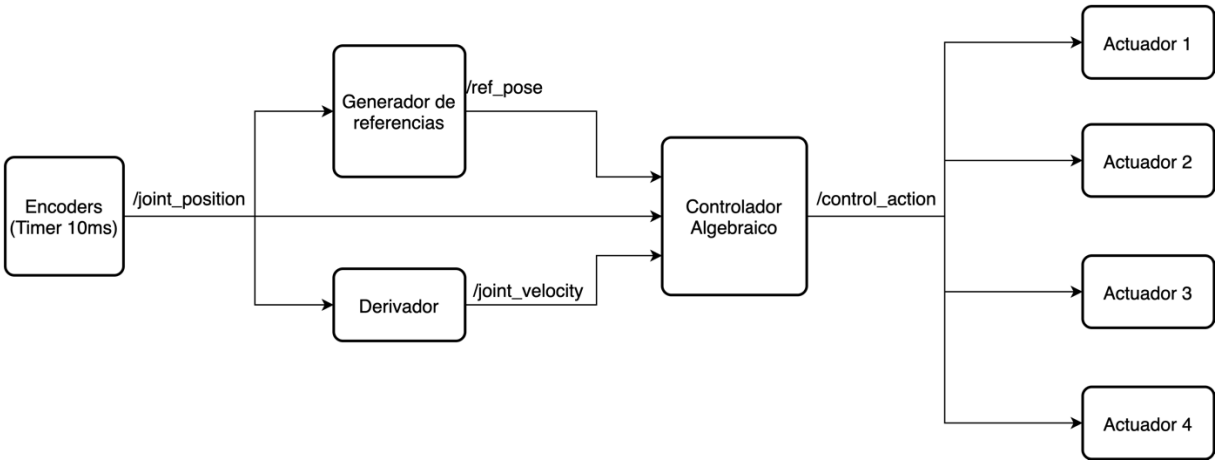


Figura 37: Esquema de ROS2 del controlador algebraico del robot de 4DOF

5.2.9.3. ESCAPE DE UNA SINGULARIDAD CON REALIMENTACIÓN DE LAS CÁMARAS

Una de las desventajas de los robots paralelos es que presentan singularidades. Estos puntos se caracterizan por un valor cercano a cero del determinante de la matriz Jacobiana. Existen dos tipos de singularidades. En las de tipo I, el robot puede perder como mínimo un grado de libertad y estas se dan en los límites del espacio de trabajo. Por otro lado, las de tipo II son aquellas en las que el robot gana como mínimo un grado de libertad. Estas se pueden dar en el propio espacio de trabajo [17].

Una singularidad de tipo II que suceda dentro del propio espacio de trabajo es peligrosa para el ejercicio de rehabilitación. Se puede perder el control debido a que el robot gana como mínimo un grado de libertad. Por tanto, la plataforma se puede mover incluso si la velocidad de los actuadores es 0.

Por este motivo, es necesario establecer un método para escapar de estos puntos singulares. En primer lugar, se debe detectar que el robot se encuentra en una singularidad y qué patas están afectando a esta. Para lo primero, se tiene la información del determinante del Jacobiano y para lo segundo, los ángulos OTS.

Para calcular los ángulos OTS, es necesario resolver el problema de la cinemática directa del robot. Sin embargo, dado que en un punto singular el determinante del Jacobiano está muy próximo a 0, no se pueden obtener valores fiables a partir del modelo, sería peligroso una realimentación puesto que el cálculo numérico inestabilizaría.

Es por este motivo, por lo que se ha realimentado el controlador para escapar de la singularidad con la información procedente de las cámaras de Optitrack. Estos sensores sí que proporcionan una posición fiable de la plataforma móvil del robot en todo momento, incluyendo los puntos singulares.

El controlador propuesto está formado por dos bucles de control, en el bucle interno, se ha implementado un controlador algebraico de posición, el mismo que el explicado en el punto anterior, gracias a la implementación modular de la arquitectura a través de ROS2, no ha sido necesario desarrollar más código para este bucle interno.

El bucle externo tiene la finalidad de detectar que el robot se encuentra en una posición singular, conocer qué patas están afectando y desplazar al robot de la posición singular a una que no lo sea. Es decir, modifica la referencia original sumando un incremento de posición. El esquema del controlador es el siguiente:

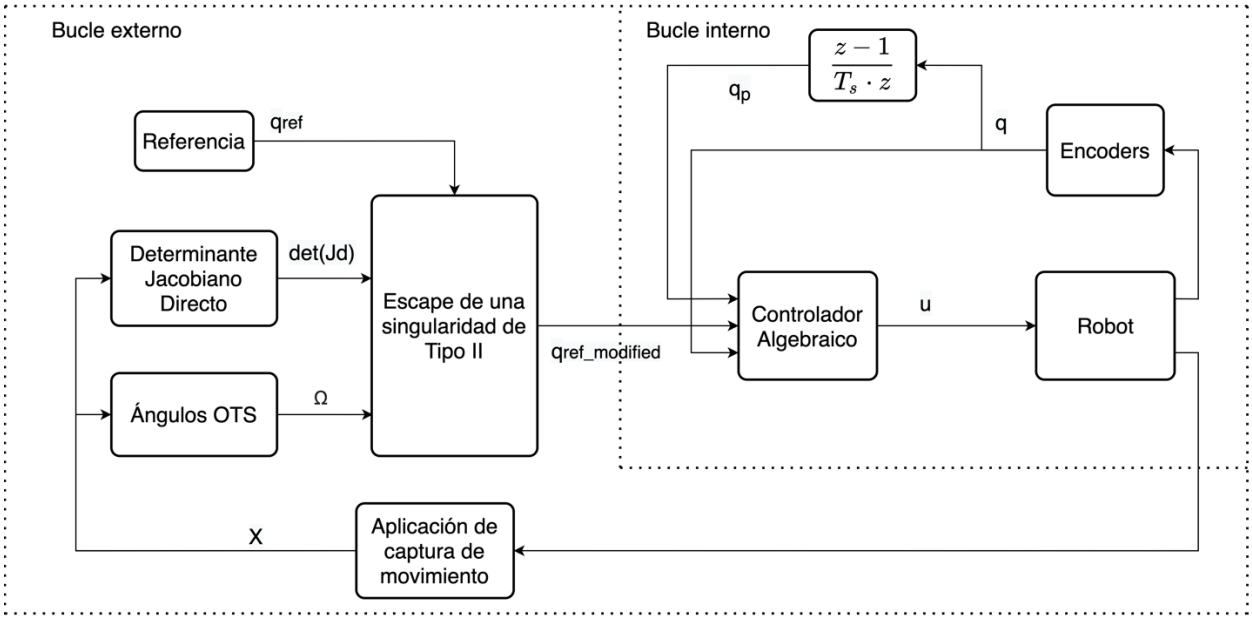


Figura 38: Diagrama dinámico del controlador para escapar de una singularidad

En la siguiente imagen se muestra el esquema de nodos y topics de ROS2 que representa este controlador:

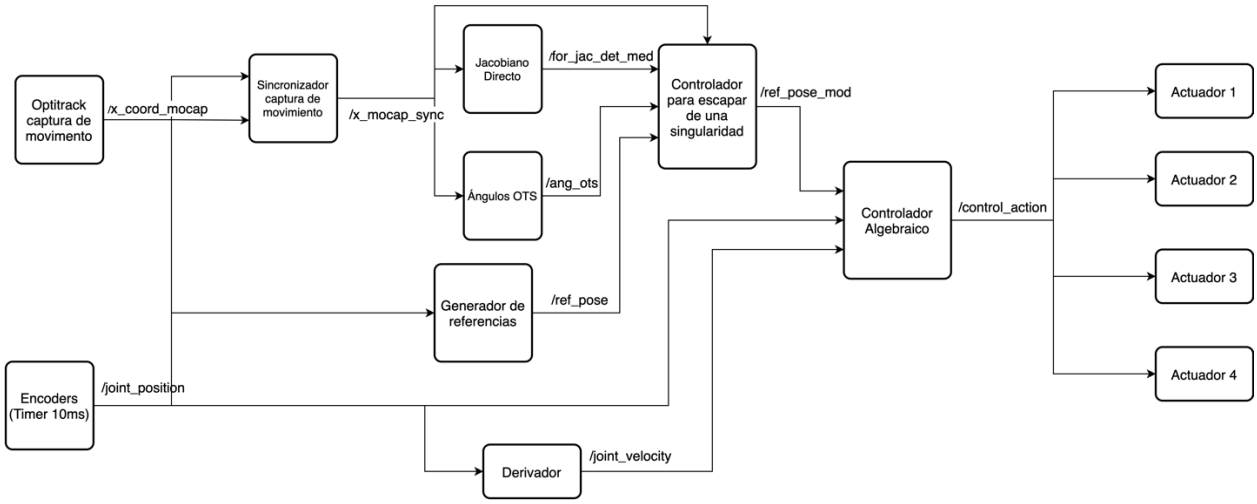


Figura 39: Esquema de ROS2 del controlador para escapar de una singularidad

Como se ha mencionado anteriormente, el nodo encargado de recibir la información del servidor Motive, publica los datos en mensajes de ROS2 cada vez que se recibe un nuevo frame de datos de Motive. Entonces, no tiene la misma frecuencia que el bucle de control (10 ms). Para sincronizar estos datos, se ha construido otro nodo sincronizador publica la posición de la plataforma móvil dada por el sistema de captura de movimiento cada vez que se recibe un dato de posición.

5.2.9.4. CONTROLADORES DE FUERZA PARA EL ROBOT DE 3DOF

Para desarrollar ejercicios de rehabilitación de una manera segura, es necesario conocer y controlar la fuerza que el robot ejerce sobre el paciente. Esto se puede realizar realimentando el sistema de control a través de los datos proporcionados por el sensor de fuerza y diseñando un nuevo controlador que funcione externamente a los controladores de posición ya implementados. Es decir, a través del enfoque propuesto, se tiene un bucle interno para controlar la posición del robot y un bucle externo que controla la fuerza que este ejerce. Esto permite también diseñar ejercicios de rehabilitación resistivos.

5.2.9.4.1. MODELO DE ADMITANCIA

Se ha utilizado un modelo de admitancia para el control de la fuerza. A este modelo se le introduce el error de fuerza (la diferencia entre la fuerza de referencia y la fuerza medida) y se obtienen velocidades o posiciones cartesianas. El hecho de utilizar este tipo de modelo conlleva una serie de ventajas. Este control puede ser utilizado tanto para el robot sin contacto con la pierna como con la pierna en el arnés, también es seguro de utilizar en el instante de contacto. Otra ventaja es que se puede modificar la dinámica con la que responde el controlador de fuerza con tres parámetros (por eje que se controla) y la modificación de estos parámetros es intuitiva. Se pueden modificar los parámetros de forma que cada eje tiene una respuesta diferente, el eje Z puede ser menos rígido que el eje Beta por ejemplo. La función de transferencia es la siguiente:

$$\frac{V_{(s)}}{F_{(s)}} = \frac{1}{m \cdot s^2 + d \cdot s + k}$$

Se denomina control por admitancia porque el robot reacciona frente a fuerzas de interacción modificando la trayectoria con respecto a la referencia original [18]. El modelo está descrito por parámetros que se pueden ajustar.

El parámetro k, influye en la cantidad se desplaza el robot frente a un error de fuerza, es decir, si este es más o menos rígido. Si es más rígido, es más difícil de mover con respecto a la referencia original. Los parámetros m y d afectan más al transitorio de la respuesta. Con esta función de transferencia realmente se está modelando un control con una masa, un coeficiente de fricción viscosa y una rigidez ficticias.

La implementación de este modelo se ha realizado realmente a través de un modelo en espacio de estados de la forma siguiente:

$$\dot{x} = A \cdot x + B \cdot u$$

$$y = C \cdot x + D \cdot u$$

El modelo continuo con las matrices construidas para este problema se muestra a continuación (los estados x1, x2, x3 son posiciones y sus derivadas velocidades, las

salidas y se corresponden con el valor de los estados puesto que la matriz de observación C es la identidad):

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \ddot{x}_1\\ \ddot{x}_2\\ \ddot{x}_3\end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ \frac{-k_1}{m_1} & 0 & 0 & \frac{-d_1}{m_1} & 0 & 0\\ 0 & \frac{-k_2}{m_2} & 0 & 0 & \frac{-d_2}{m_2} & 0\\ 0 & 0 & \frac{-K_3}{m_3} & 0 & 0 & \frac{-d_3}{m_3}\end{bmatrix} \cdot \begin{bmatrix}x_1\\ x_2\\ x_3\\ \dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix} + \begin{bmatrix} 0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\\ \frac{1}{m_1} & 0 & 0\\ 0 & \frac{1}{m_2} & 0\\ 0 & 0 & \frac{1}{m_3}\end{bmatrix} \cdot \begin{bmatrix}F_{e1}\\ F_{e2}\\ F_{e3}\end{bmatrix}$$

$$\begin{bmatrix}y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\end{bmatrix} = \begin{bmatrix}1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1\end{bmatrix} \cdot \begin{bmatrix}x_1\\ x_2\\ x_3\\ \dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix} + \begin{bmatrix}0&0&0\\ 0&0&0\\ 0&0&0\\ 0&0&0\\ 0&0&0\\ 0&0&0\end{bmatrix} \cdot \begin{bmatrix}F_{e1}\\ F_{e2}\\ Fe_3\end{bmatrix}$$

Sin embargo, para poder implementar este modelo, hay que discretizarlo. Esto se puede realizar a partir de operar con las matrices discretas que lo forman para obtener las matrices A, B, C, D discretas. Las ecuaciones son las siguientes:

$$A_d = e^{A \cdot T_s}$$

$$B_d = A^{-1} \cdot (A_d - I) \cdot B$$

$$C_d = C$$

$$D_d = D$$

La discretización, se ha realizado antes de comenzar con el bucle de control. La operación que se realiza en el bucle es la siguiente:

$$x_{k+1} = A_d \cdot x_k + B_d \cdot u_k$$

$$y_k = C \cdot x_k + D_d \cdot u_k$$

Todas las operaciones matriciales involucradas anteriormente se han realizado a través de la librería Eigen, tanto la discretización como la actualización de la salida y el estado a partir del error de fuerza.

Después de haber realizado diversos experimentos y haber modificado el valor de las constantes del modelo de admitancia de cada eje, las seleccionadas finalmente son las siguientes:

Eje\Parámetro	m	d	k
Gamma	25	111,75	125
Beta	25	111,75	125
Zeta	100	447	500

Tabla 6: Constantes del modelo de admitancia para el controlador de fuerza

5.2.9.4.2. CONTROLADOR BASADO EN VELOCIDAD (VERSIÓN 1)

En la primera versión del controlador, del modelo de admitancia se obtiene la velocidad. Esta es una velocidad cartesiana (de la plataforma móvil del robot), entonces es necesario convertirla al espacio articular a partir de la matriz Jacobiana Directa. Posteriormente, se integra esta velocidad para obtener los incrementos de posición necesarios. Estos incrementos se suman a la referencia de posición y se obtiene la referencia modificada que es la que se introduce en el controlador de posición (PD+G). El esquema se muestra en la siguiente imagen:

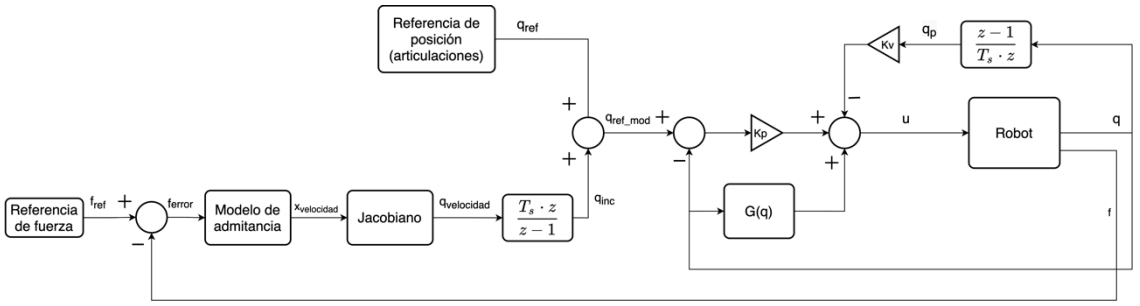


Figura 40: Diagrama dinámico del controlador de fuerza basado en el modelo de admitancia, versión 1

La implementación en ROS2 de este controlador es la siguiente:

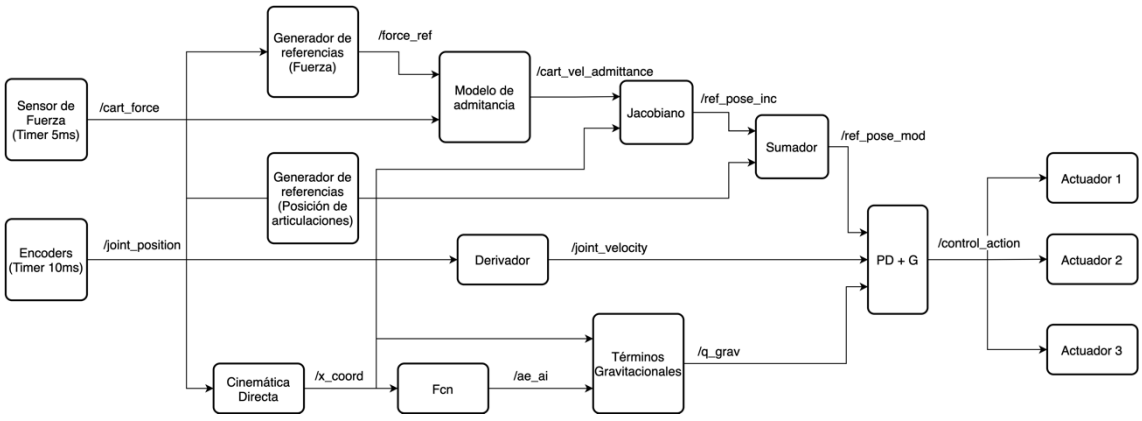


Figura 41: Esquema de ROS2 del controlador de fuerza versión 1

Se ha instanciado dos veces el componente de generación de referencias genérico, uno para generar la referencia de posición y otro para generar la referencia de fuerza. El sensor de fuerza tiene su propio temporizador con menos frecuencia puesto que era de interés estudiar la respuesta del sensor. Su valor del periodo se puede modificar para que coincida con los Encoders. En el nodo de la admitancia, se sincroniza la fuerza y se emite un valor de velocidad y posición cada vez que se recibe un valor de referencia de fuerza, entonces, este nodo está sincronizado con el temporizador de los Encoders.

5.2.9.4.3. CONTROLADOR BASADO EN POSICIÓN (VERSIÓN 2)

La segunda versión del controlador se diferencia en la primera en que esta vez se obtiene la posición del modelo de la admitancia. Este incremento de posición está en el espacio cartesiano, entonces se ha sumado este con la referencia cartesiana. Posteriormente, se obtiene el valor de la referencia modificada de las articulaciones gracias a calcular la cinemática inversa en cada iteración del bucle de control. Se puede observar mejor en el siguiente esquema:

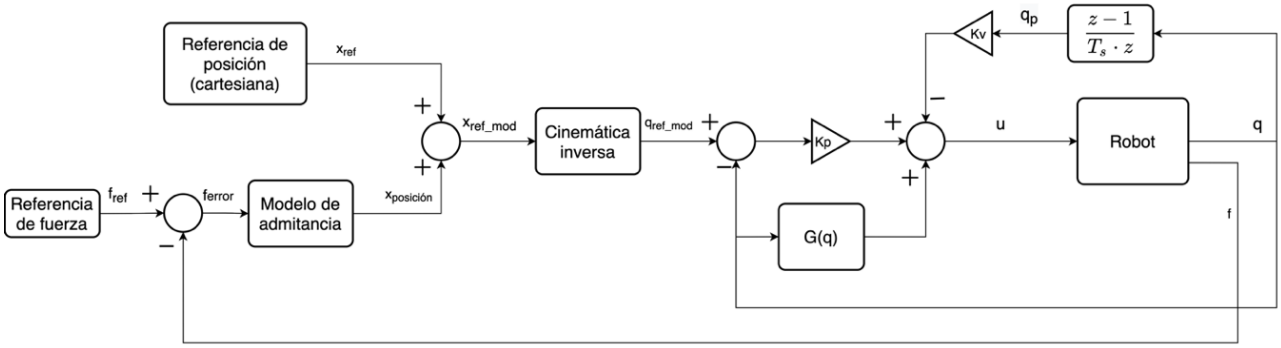


Figura 42: Diagrama dinámico del controlador de fuerza versión 2

Este esquema dinámico se ha traducido a ROS2 con la siguiente implementación:

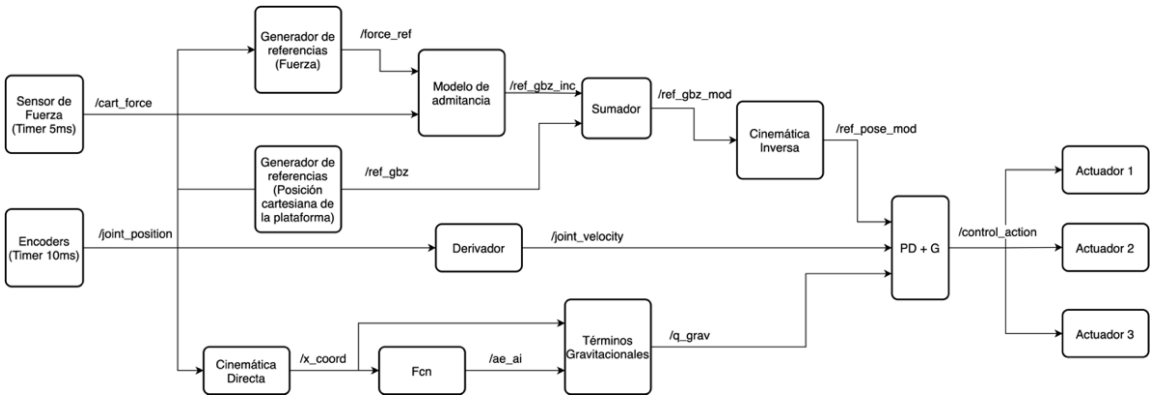


Figura 43: Esquema de ROS2 del controlador de fuerza versión 2

Esta vez, el generador de referencia de posición publica las referencias en el espacio cartesiano (posición y orientación de la plataforma móvil del robot). Después, de la función de la admitancia se obtiene directamente la posición y después de sumarla a la referencia original, se calcula la cinemática inversa y se obtiene la nueva referencia modificada que es la entrada del bucle de control de posición (PD+G).

5.2.9.5. CONTROLADOR BASADO EN APRENDIZAJE POR REFUERZO

Este controlador se ha implementado sobre la simulación del robot de 3 grados de libertad. Este robot tiene un modelo más sencillo libre de singularidades que permite que los algoritmos aprendan más rápidamente y establemente.

5.2.9.5.1. ENTORNOS DESARROLLADOS

Para poder implementar un controlador basado en aprendizaje por refuerzo es importante desarrollar un entorno mediante el cual los algoritmos de aprendizaje pueden entrenar en este, de manera que se puedan ejecutar muchas simulaciones o experimentos reales. La interfaz o formato de entornos más utilizada son los entornos de OpenAI Gym [19].

Gym es un conjunto de herramientas que permite entrenar diversos algoritmos de aprendizaje por refuerzo de forma que se puede comparar la respuesta de estos. Tiene entornos ya construidos de diferente naturaleza como control clásico, robótica o videojuegos sencillos.

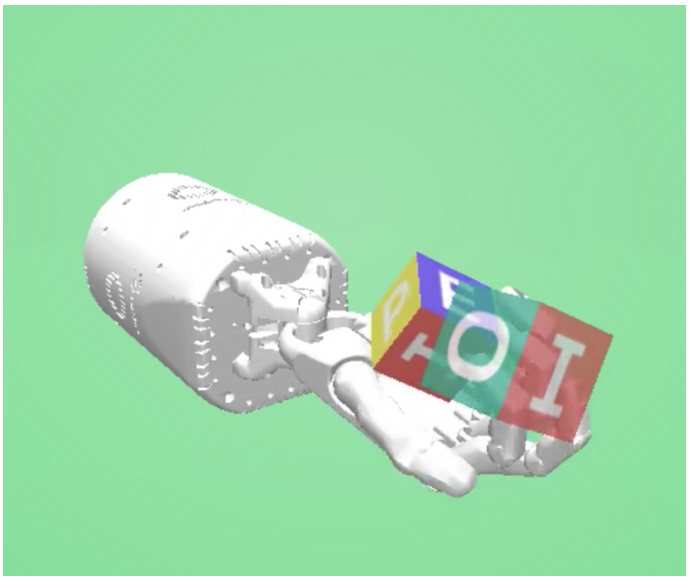


Figura 44: Ejemplo de entorno ya construido cuyo objetivo es orientar correctamente el cubo. Fuente: https://gym.openai.com/envs/HandManipulateBlock-v0/

En Gym también se pueden desarrollar entornos personalizados, esto es lo que se ha implementado en este proyecto. Este entorno encapsula todas las operaciones necesarias para que después el algoritmo de aprendizaje entrene sobre este a partir de

la ejecución de múltiples episodios. El entorno implementado se basa en una clase de Python cuyos métodos más importantes son los siguientes:

- Step: Este método tiene como argumento la acción que aplica el algoritmo. Devuelve la siguiente observación y recompensa. Entonces este método encapsula un paso de la simulación o del experimento.
- Reset: Esta función prepara todo lo necesario para que comience un nuevo episodio. Devuelve la primera observación.

El entorno implementado interacciona con MATLAB/Simulink debido a que en este software es donde se tiene implementado el modelo del robot. La comunicación entre MATLAB y la clase de gym se ha realizado a través de topics de ROS2. Entonces, el entorno construye también un nodo de ROS2. La siguiente figura muestra un esquema simple del agente y del enrono:

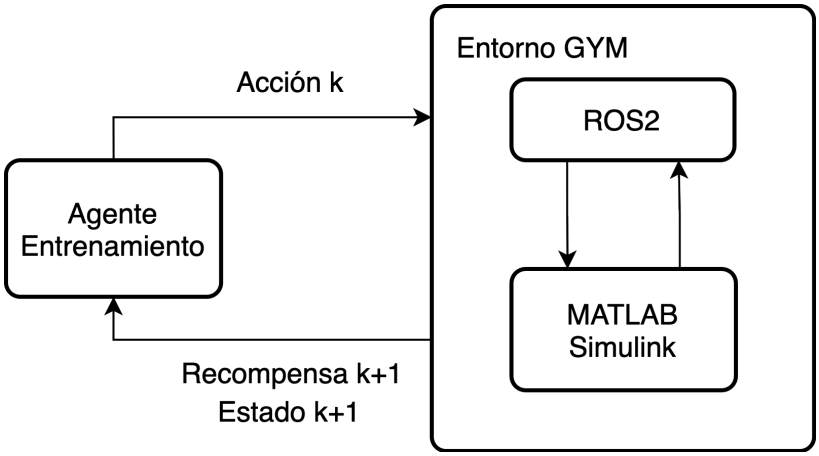


Figura 45: Esquema del entorno y del agente del controlador basado en Aprendizaje por Refuerzo

El controlador implementado es una combinación de un control PD convencional al cual se le suma la acción de control de un algoritmo de aprendizaje por refuerzo. Esto es lo que se muestra en el siguiente esquema. Gracias al controlador PD, el agente aprende más rápidamente a disminuir el error de posición. En estos dos artículos se propone la combinación de técnicas de aprendizaje por refuerzo con técnicas de control convencionales para que el algoritmo aprenda a compensar la dinámica no modelada [20] [21].

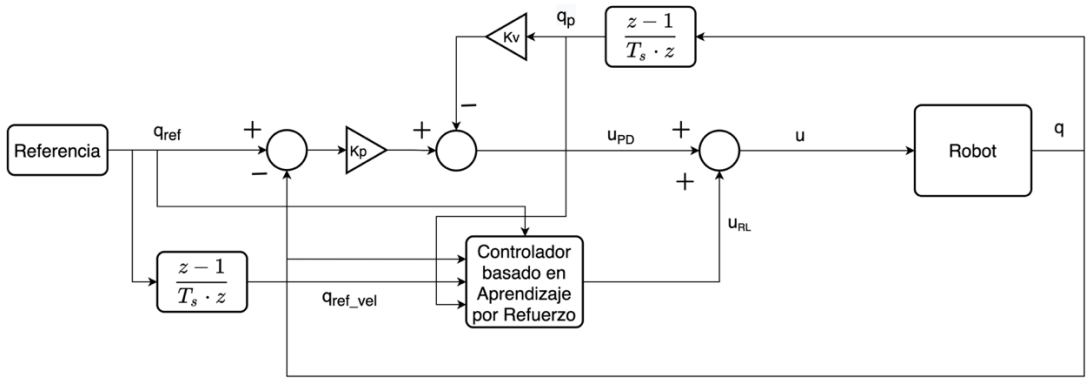


Figura 46: Diagrama dinámico controlador basado en Aprendizaje por Refuerzo

Los nodos y topics que forman el controlador se muestran a continuación:

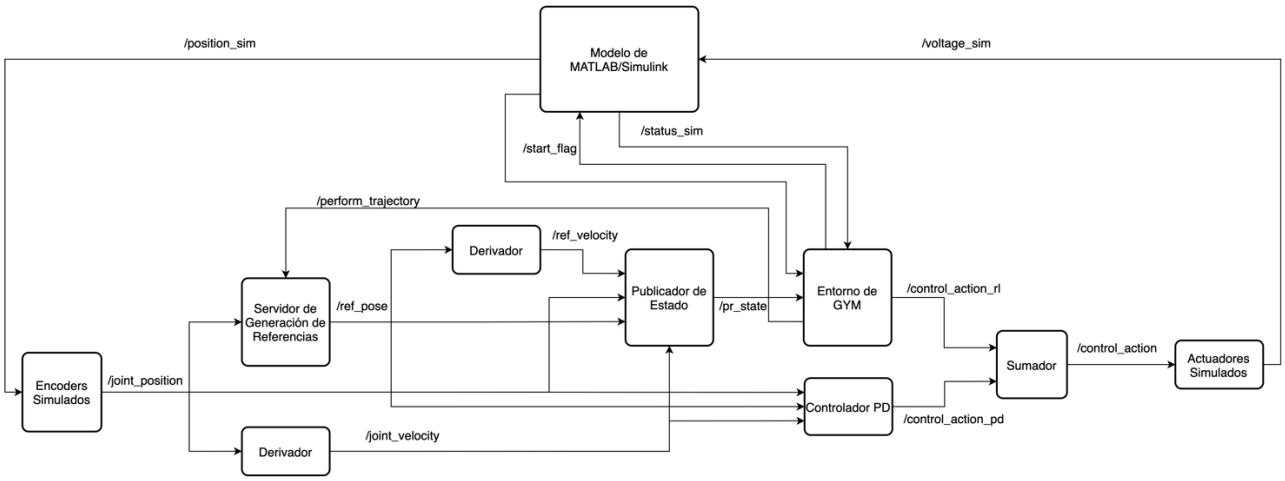


Figura 47: Esquema de ROS2 del controlador basado en Aprendizaje por Refuerzo

En este esquema se puede observar, en primer lugar, que se ha utilizado el modelo de MATLAB/Simulink del robot junto con los nodos de los encoders y actuadores simulados para entrenar el agente. Posteriormente, se ha utilizado el servidor de trayectorias para ejecutar una trayectoria aleatoria en cada episodio dentro de un conjunto de 39 trayectorias diferentes. Después se encuentra el publicador de estado, que recoge los datos de referencia de posición y de velocidad; y posición y velocidad reales. La salida de este nodo se introduce en el entorno de gym. Finalmente se suman las acciones de control del algoritmo de aprendizaje y la del algoritmo PD.

En cuanto al entorno personalizado de gym, lo más importante que hay que implementar son las funciones step y reset. La función reset hace todo lo necesario para que comience una nueva simulación. La función step sería un paso del entorno. Se le introduce una acción y devuelve la siguiente observación y recompensa. Entonces, estas funciones hacen también todo lo posible para comunicarse con el programa de Matlab/Simulink para que una nueva simulación y un paso de ella sea posible. Esta

comunicación se ha realizado por medio de topics de ROS2. La observación también la recibe de topics de ROS2, entonces, esta clase también implementa un nodo.

En la siguiente figura se muestra un esquema con todos los elementos que forman el entorno de gym. Se han implementado diferentes entornos, pero todos parten de la misma base y son muy similares.

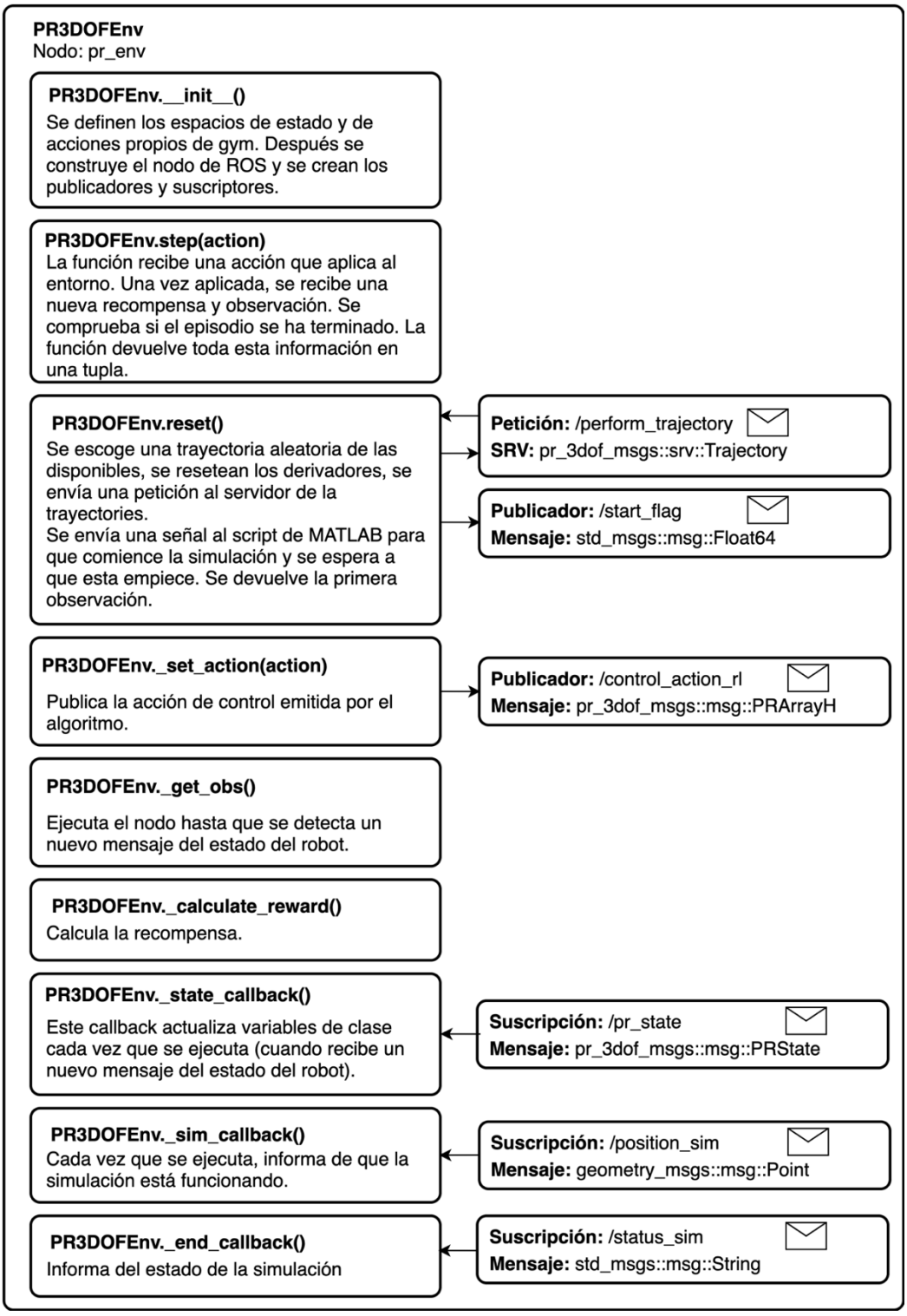


Figura 48: Esquema del entorno GYM desarrollado con Python, implementa también un nodo de ROS2

5.2.9.5.2. ENTRENAMIENTO DE ALGORITMOS

Los algoritmos no se han construido programando, si no que se ha utilizado la librería Stable Baselines. Esta librería implementa algoritmos de aprendizaje por refuerzo cuyo funcionamiento está asegurado. Stable Baselines parte de la librería de aprendizaje Baselines creada por OpenAI y añade una serie de ventajas. Una de las más importantes es que se puede visualizar a través de tensorboard el proceso de aprendizaje para comprobar de una manera sencilla que el algoritmo está aprendiendo, que converge o que ha inestabilizado.

La ventaja de utilizar esta librería también es que es muy sencillo cambiar un algoritmo por otro, dado que el entorno está encapsulado con el formato correcto para que cualquier algoritmo de esta librería lo utilice (siempre que sus espacios de estados y de acciones sean continuos). También es sencillo modificar los hiperparámetros de los algoritmos y del aprendizaje.

Los algoritmos con los que mejores resultados se han obtenido son el algoritmo TRPO (Trust Region Policy Optimization 2015 [22]) y PPO (Proximal Policy Optimization 2017 [23]). Estos son algoritmos relativamente nuevos que tratan de resolver el siguiente problema: Se desea entrenar lo más rápido posible, pero pasos muy grandes en el gradiente de la policy, si estos son malos, puede conllevar a que no se pueda recuperar lo aprendido, debido también a que es altamente no lineal. Un mal comportamiento dará lugar también a malas experiencias [24].

Ambos algoritmos mejoran a los anteriores en el sentido de mejorar la estabilidad y consistencia del gradiente de la policy y se han obtenido buenos resultados en tareas de control.

En cuanto a la señal de recompensa, ya se ha mencionado que esta es crucial para conseguir un buen comportamiento del agente. Se han utilizado dos variantes diferentes (n es el número de articulaciones activas):

$$Reward_1 = -w_{iae} \cdot \frac{\sum_{i=0}^{n}\left(qref_{(k)i} - q_{(k)i}\right)^2}{n} - w_{iadu} \cdot \frac{\sum_{i=0}^{n}\left(u_{(k)i} - u_{(k-1)i}\right)^2}{n}$$

Esta primera función de recompensa penaliza en primer lugar el error de posición y, en segundo lugar, los incrementos de las acciones de control. No solo se requiere minimizar el error de posición, si no que también el algoritmo ofrezca acciones de control realizables. Se tienen también dos coeficientes para penalizar más el error de posición o los incrementos de la acción de control.

$$Reward_2 = -w_{iae} \cdot \frac{\sum_{i=0}^{n}\left(qref_{(k)i} - q_{(k)i}\right)^2}{n} - w_{iaevel} \cdot \frac{\sum_{i=0}^{n}\left(qref\_vel_{(k)i} - qvel_{(k-1)i}\right)^2}{n}$$

La segunda variante, en vez de penalizar los incrementos de la acción de control, penaliza el error de velocidad. De esta manera también se consiguen acciones de control razonables.

Otra cuestión relacionada con la señal de recompensa es la recompensa que se obtiene cuando se termina el episodio. Si el episodio termina porque ha habido un error en la simulación o se han superado los límites del entorno significa que el comportamiento del agente es incorrecto. Hay que penalizar fuertemente estos estados terminales porque si no, el agente puede descubrir que terminando la simulación antes obtiene más recompensa, pero no lograría el comportamiento deseado.

Entonces se ha añadido esta recompensa cuando el episodio termina por un fallo de la simulación que depende del número de iteraciones transcurridas. Si han transcurrido pocas iteraciones se penaliza más.

$$R = -a \cdot (1 + e^{-\frac{t}{b}})$$

Las variables a y b son coeficientes que ajustan la curva. El valor de a debe ser grande para garantizar que el agente no descubra que es mejor terminar la simulación que desarrollar un buen comportamiento.

En cuanto al entrenamiento del algoritmo, la interfaz de programación de Stable Baselines es sencilla de utilizar gracias a haber creado el entorno con el formato gym. Cada 40000 iteraciones, se guardan los parámetros del modelo para posteriormente cargarlos y poder seguir entrenando en cualquier punto. Para los dos algoritmos (TRPO y PPO) se han utilizado modelos de la policy personalizados. Para modelar a la función actora se ha utilizado una red neuronal formada por 3 capas de 128, 64 y 64 neuronas respectivamente. La función de valor está modelada por la misma estructura de redes neuronales.

Entonces, a la hora del entrenamiento, para valorar si el algoritmo está mejorando, un buen factor a tener en cuenta es la recompensa total obtenida después de un episodio (el sumatorio de las recompensas de cada iteración de un episodio). La evolución de este dato sirve para ver si el algoritmo mejora conforme entrena o no. En la siguiente imagen se puede ver la representación de la recompensa total obtenida en cada episodio para un entrenamiento de 10e6 iteraciones para el controlador TRPO+PD:

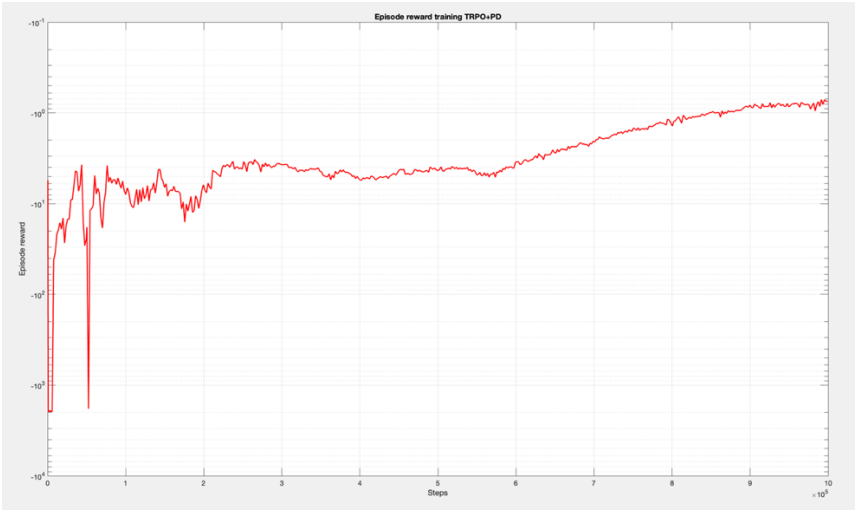


Figura 49: Entrenamiento de agente TRPO+PD, evolución de la recompensa total obtenida en cada episodio

Al principio del entrenamiento, el algoritmo tiene un comportamiento aleatorio que da lugar a acciones equivocadas. Por este motivo se obtiene una mala recompensa. Conforme transcurren las iteraciones, tiende a obtener una mejor recompensa en cada episodio, hasta llegar a recompensas totales menores que -1 (el máximo es 0).

En la siguiente imagen se muestra las recompensas totales obtenidas en un entrenamiento de más de 1e6 iteraciones para el controlador PPO+PD:

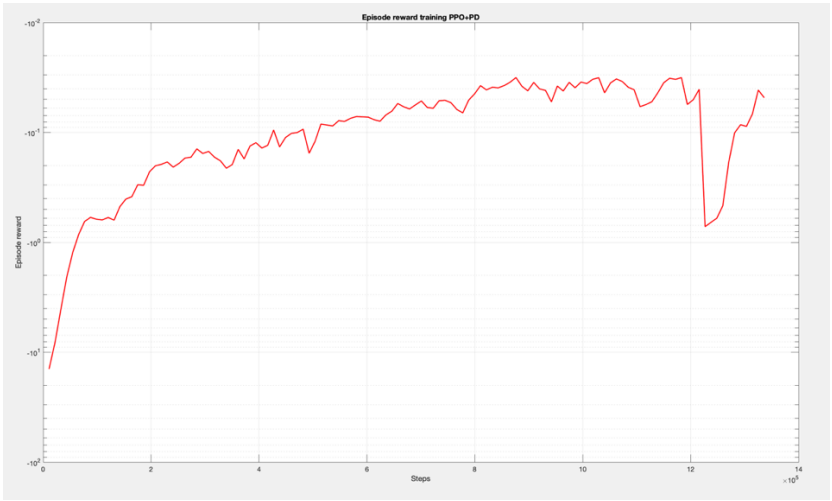


Figura 50: Entrenamiento de agente PPO+PD, evolución de la recompensa total obtenida en cada episodio

El hecho de guardar cada cierto tiempo el modelo sirve para posteriormente cargar los parámetros y seguir el entrenamiento (o utilizarlo solamente para predecir sin entrenar). Después de este entrenamiento, se han cargado los parámetros del último modelo guardado y se ha hecho un segundo entrenamiento con la siguiente evolución de la recompensa total.

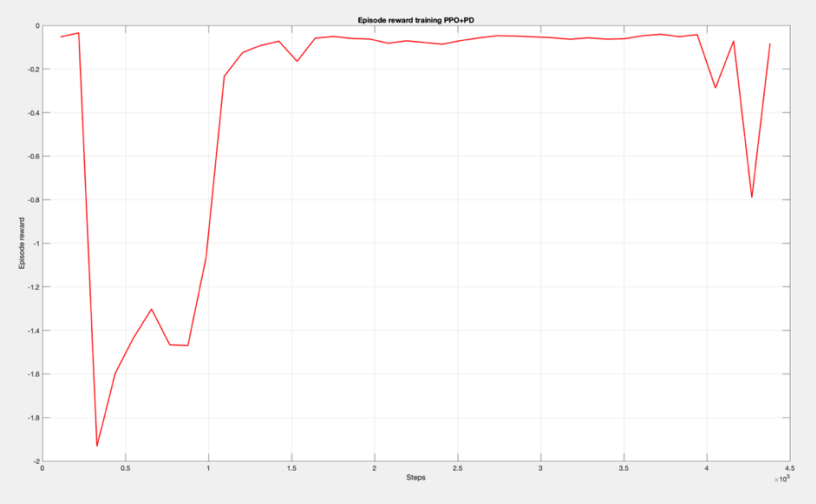


Figura 51: Continuación del entrenamiento del agente PPO+PD, evolución de la recompensa total obtenida en cada episodio

Para los segundos entrenamientos, se ha disminuido el ratio de aprendizaje para que el aprendizaje sea más estable a costa de ser más lento. Finalmente, después de estos entrenamientos, se han guardado los parámetros finales del controlador y se pueden utilizar para cargar un controlador que solamente prediga las acciones sin entrenar. De esta manera, los recursos computacionales requeridos son mucho menores.

5.2.10. Almacenamiento de datos en un segundo ordenador

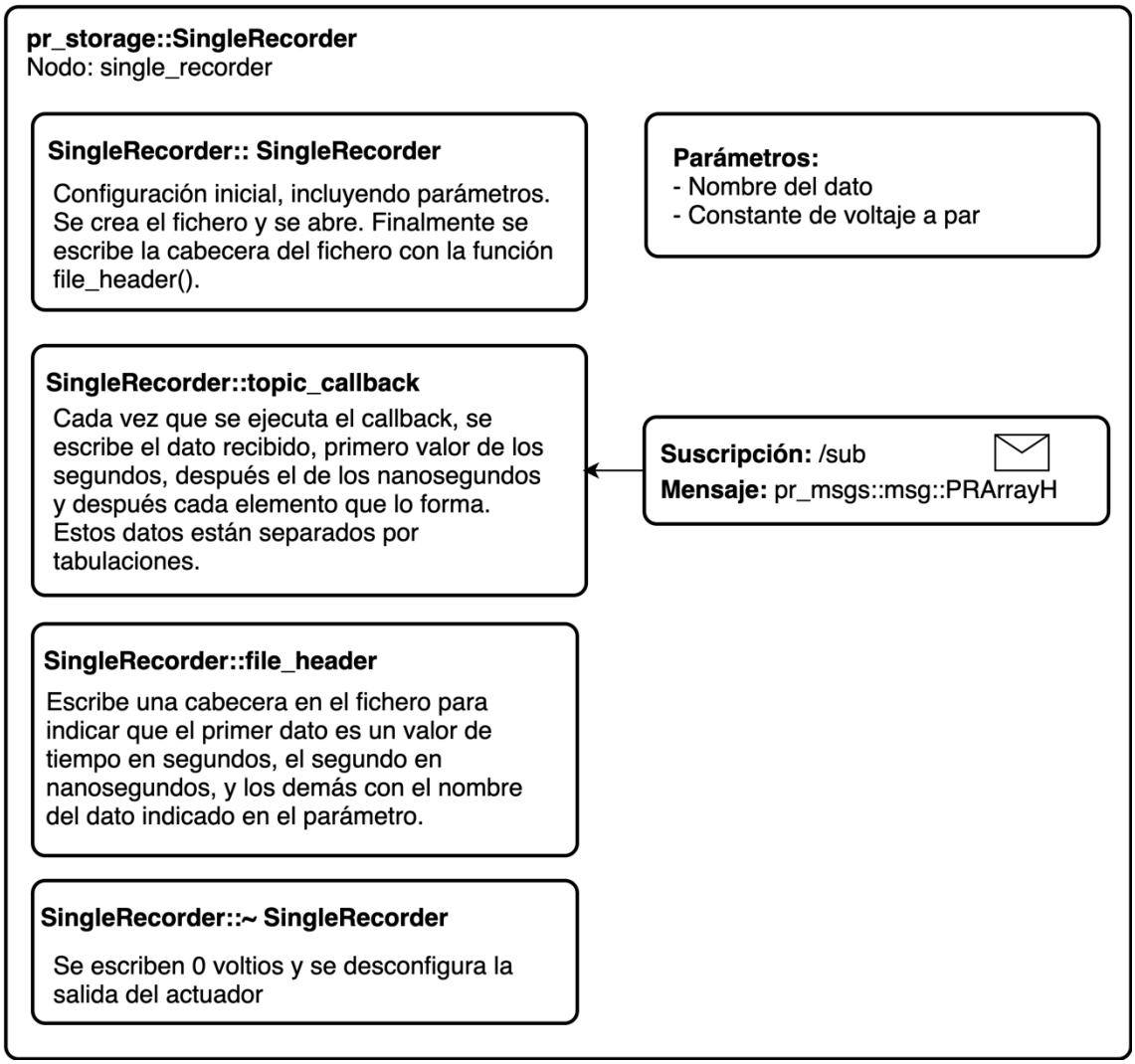
Con el fin de estudiar la respuesta de cada controlador y monitorizar la evolución de los pacientes, es necesario guardar los datos de cada iteración del bucle de control. En cada iteración del bucle, se podría escribir en un fichero los datos, sin embargo, las operaciones de escritura en disco duro no son deterministas y no es recomendable realizarlas online.

Otra alternativa sería guardar en memoria RAM cada iteración para que cuando el ejercicio termine, se escriban todos los datos en el disco duro. El inconveniente es que los ejercicios de rehabilitación pueden ser muy duraderos y se podría consumir bastante memoria RAM.

Así pues, se ha aprovechado una de las características de ROS2 para comunicar un ordenador auxiliar con el ordenador de control para que el auxiliar realice las operaciones de acceso a disco de manera que no disminuya el rendimiento del control. Gracias al sistema de distribución de datos que presenta ROS2 en las capas inferiores es posible que dos máquinas que comparten la misma red descubran los nodos, topics, servicios y acciones que se están ejecutando en ellas. Para que esto sea posible, hay que mantener la variable de entorno ROS_LOCALHOST_ONLY=0 (su valor por defecto es 0). Si fuera 1, solamente se trabaja en modo local.

Se ha instalado también ROS2 en este ordenador auxiliar y se ha desarrollado un entorno de trabajo pr_pc en el que se han construido paquetes para desarrollar programas que se suscriban al topic indicado y que guarden la información de cada iteración en un fichero. Se han creado dos paquetes, pr_storage y pr_3dof_storage además de otros dos que corresponden con los mensajes ya implementados en el ordenador de control pr_msgs y pr_3dof_msgs.

Los componentes implementados se suscriben al topic indicado en el archivo de lanzamiento (remapeado) y cada vez que reciben un nuevo dato, lo escriben es el fichero. El componente más representativo de todos es SingleRecorder que se puede suscribir a mensajes del tipo PRArray (el más común en la red de ROS2 implementada). La estructura del nodo es la siguiente:



Este componente en concreto es para el robot de 4 grados de libertad y para el tipo de mensaje de PRArray. Se han implementado otros componentes para otros mensajes, así como para el robot de 3 grados, la estructura de estos es la misma que la mostrada anteriormente.

Los paquetes pr_storage y pr_3dof_storage también contienen archivos de lanzamiento para ejecutar los componentes que sean necesarios y guardar todos los datos de un experimento. Entonces, antes de ejecutar la trayectoria, se debe ejecutar el archivo de lanzamiento para que los componentes estén preparados para guardar los datos.

En el script de lanzamiento, antes de añadir los componentes para la ejecución, crea una nueva carpeta cuyo nombre es la fecha y la hora de ese momento, dentro de ella es donde se guardan los archivos de cada variable del bucle de control.

5.2.11. PAQUETES PARA EL LANZAMIENTO DE CONTROLADORES

Ya se han introducido los archivos de lanzamiento, estos son scripts en lenguaje Python que permiten ejecutar diversos nodos para que no haya que ejecutarlos individualmente. Como la implementación realizada está basada en componentes, en el archivo de lanzamiento, primero se lanza un nodo contenedor de componentes capaz de recogerlos en tiempo de ejecución y ejecutarlos. Después de lanzar el contenedor, se añaden los componentes necesarios para el controlador implementado.

Entonces, se han organizado los archivos de lanzamiento en los paquetes pr_bringup y pr_3dof_bringup. El primero para el robot de 4 grados y el segundo para el de 3 grados de libertad. En estos archivos también se puede cambiar el nombre de las suscripciones y publicadores que tiene un nodo por defecto (remapear). También se puede cambiar el nombre del nodo que tiene por defecto para poder instanciar el mismo componente varias veces (tal como se ha hecho con los actuadores o generadores de referencias).

Otra cuestión con relación a los paquetes de lanzamiento es que se ha añadido una carpeta para almacenar parámetros en archivo yaml. Se ha almacenado información relativa a las configuraciones geométricas de los robots, tiempo de muestreo, voltaje de saturación, constantes de los controladores, ruta del archivo de la trayectoria de referencia, nombre de los marcadores del robot, etc.

El script de Python entonces lee el archivo yaml indicado y añade los parámetros a los nodos que los requiera. Estos archivos ayudan a tener los parámetros en un único fichero de forma que se puede cambiar fácilmente.

Por otro lado, ya se ha mencionado que se ha trabajado con dos versiones distintas del robot de 4 grados de libertad. La modularidad de la arquitectura permite que únicamente sea necesario cambiar el nombre del robot en los archivos yaml para trabajar con uno u otro (los componentes son exactamente los mismos, solamente cambian los parámetros geométricos, de masas y el nombre de los marcadores).

Cada controlador implementado tiene su propio archivo de lanzamiento, para ejecutarlo, es necesario ejecutar la siguiente sentencia en el terminal: ***ros2 launch <nombre_del_paquete> <nombre_del_archivo.launch.py>***.

5.2.12. INTERFAZ GRÁFICA DE USUARIO

En los objetivos del proyecto se ha propuesto también el desarrollo de una interfaz gráfica de usuario que permita visualizar las variables más relevantes del proyecto y ejecutar una trayectoria. Se ha decidido que la interfaz esté basada en una aplicación web debido a que, de esta manera, hay mucha más versatilidad en el sentido del dispositivo que ejecuta la aplicación. Tan solo es necesario un navegador web. Si estuviera basada en una aplicación Windows, por ejemplo, podría haber problemas de compatibilidad con aplicaciones Mac y Linux.

Por otro lado, ROS2 ofrece una serie de herramientas que permiten el desarrollo de una aplicación web para un robot que esté basado en este software. Estas herramientas se van a introducir a continuación.

5.2.12.1. HERRAMIENTAS NECESARIAS

En primer lugar, es necesario utilizar una aplicación que se denomina ROS2-Web-Bridge [25]. Esta aplicación se debe ejecutar en un ordenador y convierte la información de ROS2 (topics y servicios) a WebSockets que se pueden comunicar con un navegador. Para utilizar esta aplicación, es necesario instalar Node.JS. Esta está basada en este lenguaje de programación. Entonces, se clona el repositorio y se instalan las dependencias de este a través del gestor npm. Para utilizar el puente, es necesario ejecutar la siguiente sentencia en el terminal: ***node bin/rosbridge.js***.

Para que un cliente pueda acceder a la información de los WebSockets del puente de ROS2 es necesario utilizar la librería de JavaScript ROSLibJS. A través de esta librería, un navegador (cliente) puede conectarse con el puente y crear topics, suscribirse, publicar, enviar peticiones a un servicio, etc. De esta manera es como se soluciona la interfaz entre el robot de ROS2 y el cliente final.

En la siguiente figura se puede observar de una manera más esquemática todos los elementos que componen la interfaz gráfica:

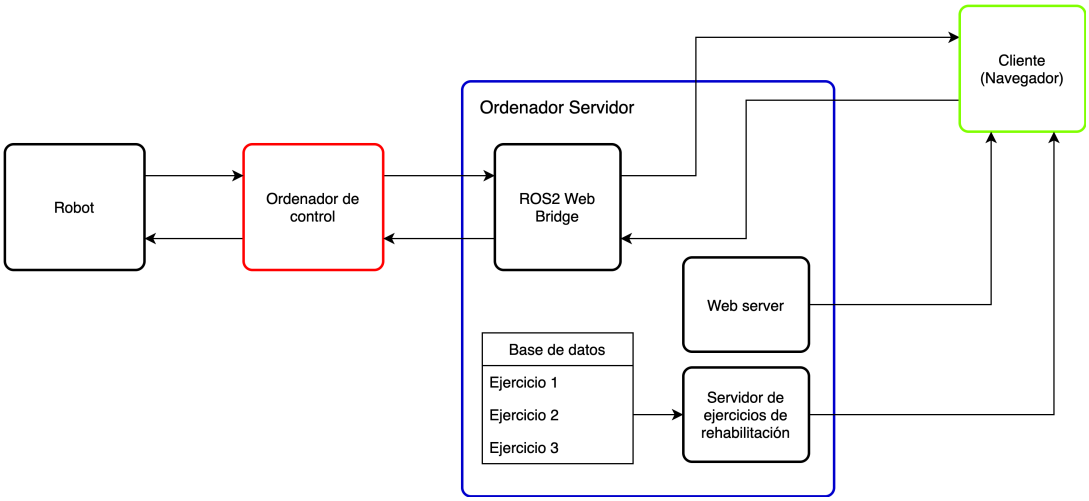


Figura 52: Esquema interfaz gráfica

En el esquema anterior se pueden diferenciar hasta tres máquinas diferentes, en primer lugar, se encuentra el ordenador de control que ejecuta los controladores del robot y tiene los interfaces de entrada y salida. Después se tiene un ordenador servidor, que ejecuta la aplicación ROS2-Web-Bridge además del servidor web y de otro servidor que accede a una base de datos y devuelve una lista con los ejercicios que hay disponibles. Por último, está el cliente, el cual puede conectarse desde cualquier máquina que tenga acceso a la red del ordenador servidor. También podría ser el propio ordenador servidor. Esto permite el uso de no solamente ordenadores, si no dispositivos más portables como móviles o tabletas.

5.2.12.2. APLICACIÓN BASADA EN REACTJS

El cliente final se trata de una aplicación basada en React [26]. React es una librería de JavaScript desarrollada por Facebook la cual facilita el desarrollo de interfaces de usuarios interactivas de una manera modular basada en componentes. Una de las principales características de React es que renderizará solamente los componentes de la pantalla que cambien, no es necesario actualizar la página web entera. Por otro lado, los componentes encapsulan operaciones y su propio estado.

Para crear el proyecto basado en React se ha ejecutado la siguiente sentencia: ***npx create-react-app pr_web.*** En el directorio del proyecto se puede ejecutar **npm start**, esta sentencia, llama a un script de lanzamiento que ejecuta el servidor de la aplicación, por defecto escucha en el puerto 3000.

La aplicación web a grandes rasgos está formada por una página principal y un componente que se corresponde con los gráficos en tiempo real. Los gráficos de este componente están basados en la librería Plotly.js. Esta librería permite implementar en un cliente gráficos visuales estadísticos, 2D, 3D y en tiempo real que es lo adecuado para este proyecto.

El componente del gráfico está formado por una función que construye un temporizador que se activa cada 100ms y modifica el estado tiempo (nueva fecha) del componente. Después, se tiene otra función que se ejecuta cada vez que el estado tiempo cambia. En esta función se actualiza el gráfico. Entonces, los 100ms anteriores es el tiempo que tarda el gráfico en actualizar sus datos (este valor se puede cambiar).

Posteriormente, la página principal define los estados necesarios, y primero se conecta al WebSocket proporcionado por ROS2-Web-Bridge. Una vez conectado, construye los suscriptores necesarios para poder adquirir los datos y visualizarlos, y establecer la lógica para los botones necesarios. El resultado de la aplicación es el siguiente:

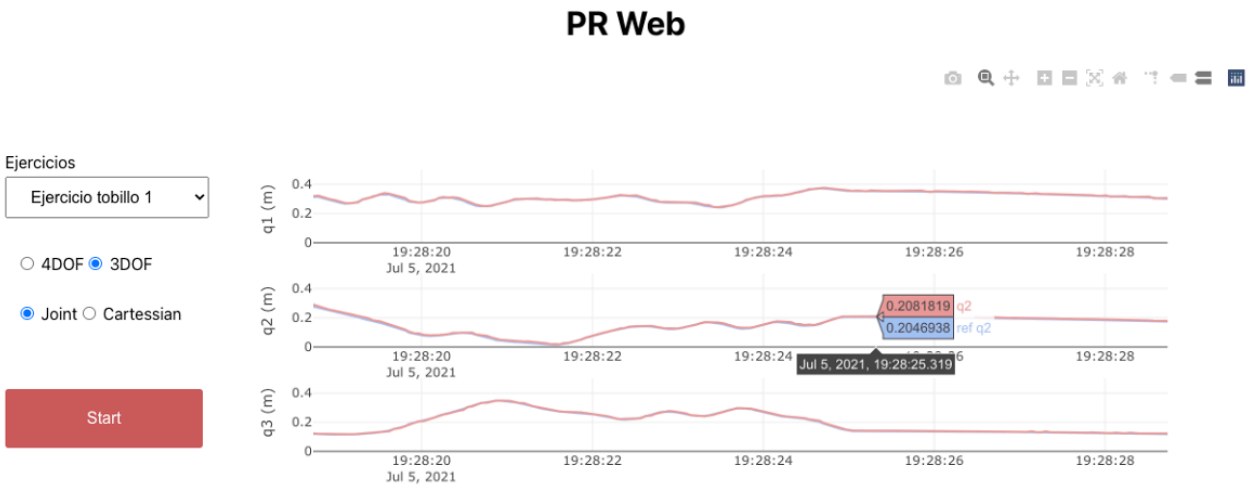


Figura 53: Interfaz gráfica basada en aplicación web

Se puede observar que hay una serie de controles. El primer selector se utiliza para seleccionar el ejercicio de rehabilitación que se desee ejecutar. Después hay dos pares de botones radio, los primeros dos son para seleccionar si se quieren visualizar variables del robot de 4 grados de libertad o del de 3 grados. Los otros dos son para seleccionar si se quiere representar los datos de la posición de las articulaciones o si se prefiere representar en el espacio cartesiano. Por último, hay un botón de inicio, que envía la trayectoria seleccionada al ordenador de control y este la ejecuta.

El estilo de la aplicación se ha modificado a través del archivo de estilos App.css. Todo el código fuente correspondiente con la aplicación web se ha guardado en un repositorio de github llamado pr_web.

5.2.12.3. SERVIDOR DE TRAYECTORIAS BASADO EN NODE

En primer lugar, se ha creado una base de datos a través del programa MySQL. Esta base de datos contiene el nombre de los ejercicios de rehabilitación disponibles. Estos ejercicios son trayectorias para el robot.

Entonces, para acceder a esta base de datos se ha desarrollado una aplicación en lenguaje Node que implementa un servidor HTTP a través de la API Express. El programa desarrollado, cuando se ejecuta a través de la sentencia **node bin/demo_db.js**, crea un servidor que escucha peticiones por el puerto 8000 de la máquina. Si se realiza una petición a la dirección <servidor>:8000/references, el servidor en primer lugar accede a la base de datos de MySQL y devuelve a través de un objeto JSON el nombre y características de todos los ejercicios disponibles.

El código fuente de este programa se ha añadido a github por medio de un nuevo repositorio llamado pr_db_server.

5.2.12.4. SERVIDOR DE ROS2 PARA EJECUTAR TRAYECTORIAS

Se han comentado diversos elementos de la arquitectura, sin embargo, falta que el ordenador de control sea capaz de recibir qué ejercicio debe ejecutar y realizarlo. Esto se ha solucionado a través de un servicio de ROS2 implementado en un componente.

El servidor implementado sustituye al anterior generador de trayectorias. Este incluye un callback que se ejecuta cuando recibe una petición. La petición se realiza con el mensaje Trajectory el cual ya ha sido mencionado en la tabla de mensajes y servicios de ROS2. Este callback, primero comprueba si hay una trayectoria ejecutándose, y si no la hay, lee los datos del fichero de trayectorias cuyo nombre se encuentra en el campo path_trajectory del mensaje. Como sucedía en el anterior generador de referencias, si las referencias están en el espacio cartesiano, se convierten aplicando la cinemática inversa al espacio articular. Una vez convertidas, establece una variable de la clase a verdadero y termina el callback.

Entonces, hay un segundo callback que se ejecuta cada vez que hay un nuevo mensaje procedente del nodo de los Encoders. Cuando se detecta que la variable mencionada anteriormente está en verdadero, se recorre en cada ejecución del callback la matriz de referencias. Cuando la matriz se termina, se mantiene siempre la última referencia. En el siguiente esquema se puede observar mejor los métodos y campos de la clase del servidor de generador de referencias:

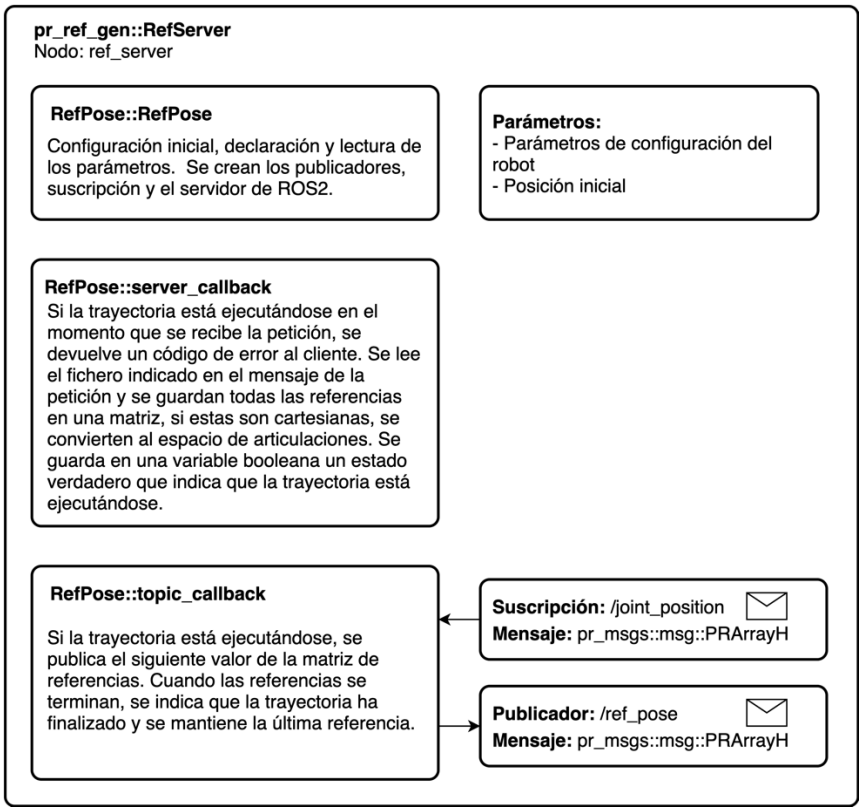


Figura 54: Esquema de componente del servidor de referencias

5.2.13. HERRAMIENTAS PARA EDICIÓN Y CONTROL DE CÓDIGO FUENTE

Para la edición del código, se ha utilizado el editor Visual Studio Code de Microsoft [27]. Visual Estudio Code es un editor de código ligero, pero con una gran eficacia, puede funcionar tanto en sistemas operativos Linux como Mac y Windows. En este proyecto se ha utilizado este editor para desarrollar el código fuente con todos los lenguajes que se han mencionado anteriormente: C++, Python, JavaScript y Node.

Visual Studio Code incorpora extensiones que facilita el desarrollo con cada lenguaje de programación y también se pueden añadir a la ruta del editor las librerías necesarias para desarrollar los programas de ROS2 (rclcpp) y que el autocompletado detecte las clases y métodos propios de estas librerías.

Por otro lado, se ha utilizado la herramienta git para el control del código fuente local [28]. La herramienta git permite tener un histórico de puntos de guardado del código fuente. Si, por ejemplo, se ha cometido un error, se puede devolver el código a como estaba en un punto anterior, de esta forma, se cometen menos errores graves. Cuando se controla el código de un directorio con git, se construye lo que se denomina un repositorio local. Se pueden crear varias ramas en este repositorio local que parten del mismo punto, pero en cada una se realizan diferentes implementaciones. Entonces, en un momento futuro, se pueden volver a fusionar estas ramas prestando atención a los cambios que han ocurrido en cada una y si hay conflictos (zonas del código que han sido modificadas en las dos ramas).

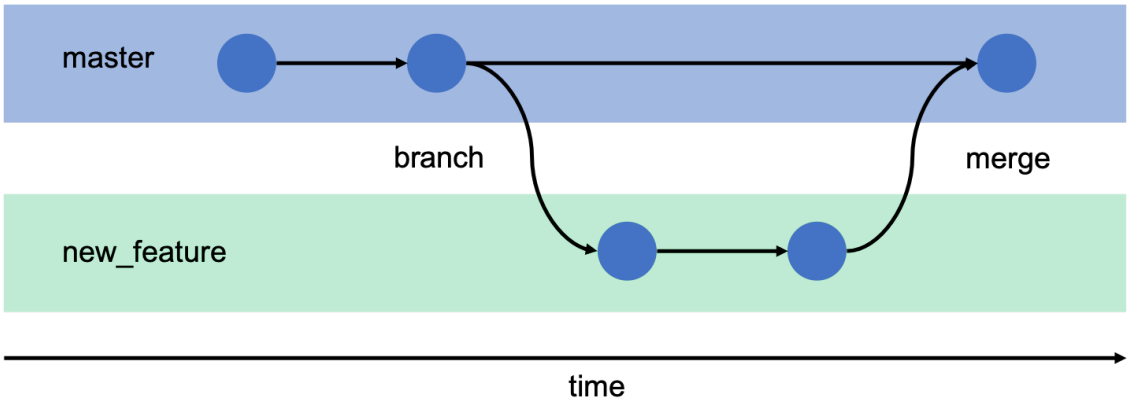


Figura 55: Ramas de git. Fuente: https://gitbookdown.dallasdatascience.com/branching-git-branch.html

No solamente se ha trabajado con git de manera local, si no que gracias a github, se han construido también repositorios remotos. El uso de github y de tener un repositorio remoto permite poder descargar el código en diferentes máquinas de una manera muy sencilla y, por tanto, también se puede desarrollar un proyecto con diversos ordenadores. Otra ventaja es la de poder colaborar y trabajar con múltiples desarrolladores al mismo tiempo.

6. RESULTADOS OBTENIDOS

En los siguientes apartados se van a mostrar los experimentos realizados con los controladores explicados anteriormente diferenciados entre los robots de 4 grados de libertad y el de 3 grados. En estos experimentos se pueden visualizar las variables de más interés de estudio como son la referencia de posición y la posición real de las articulaciones, el error de posición y la acción de control.

6.1. CONTROL DEL ROBOT DE 4 GRADOS DE LIBERTAD

6.1.1. PD CON COMPENSACIÓN DE LA GRAVEDAD

Se han desarrollado experimentos para validar la implementación del controlador PD con compensación de la gravedad. Estas trayectorias se han realizado con el nuevo robot de 4 grados de libertad. Además, se ha añadido la posición obtenida con las cámaras de la plataforma móvil para compararla con la posición estimada por el modelo. Entonces, en este experimento, no solamente se está realizando una prueba con el controlador PD con compensación de la gravedad, si no que también se está validando la lectura de la posición de la plataforma móvil a partir del sistema de visión. La compensación de la gravedad se está realimentando a partir de la estimación de la cinemática directa.

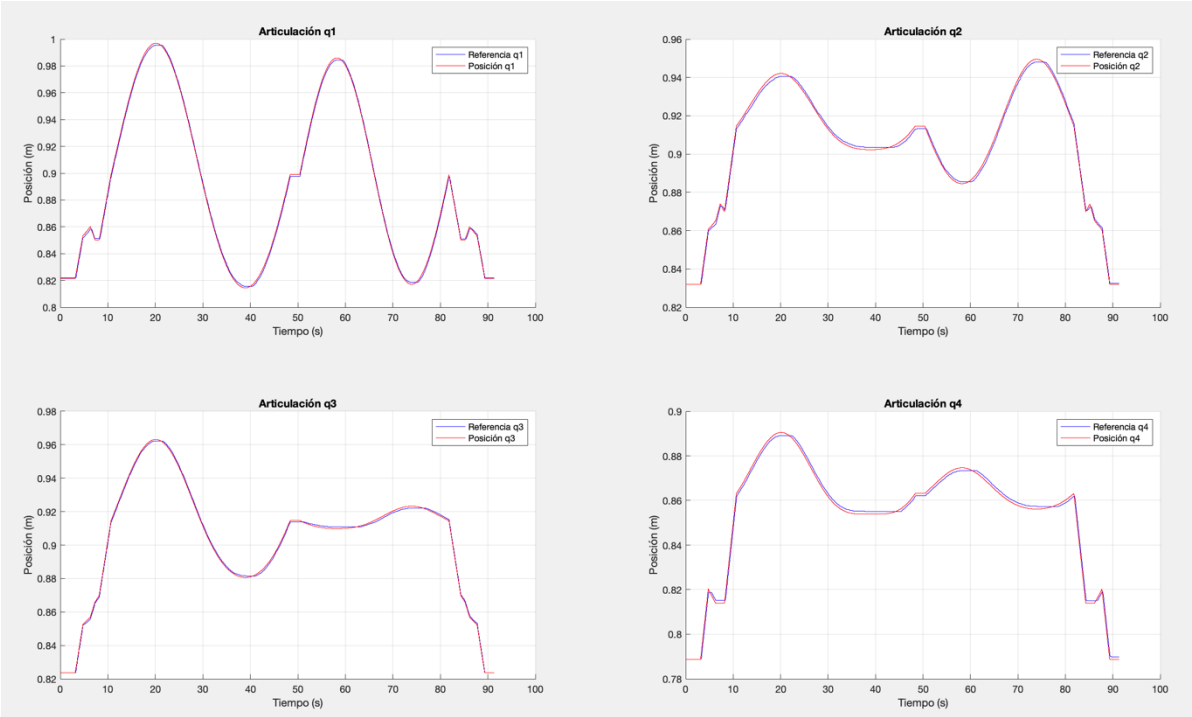


Figura 56: Experimento PD+G, trayectoria 1, posición de las articulaciones

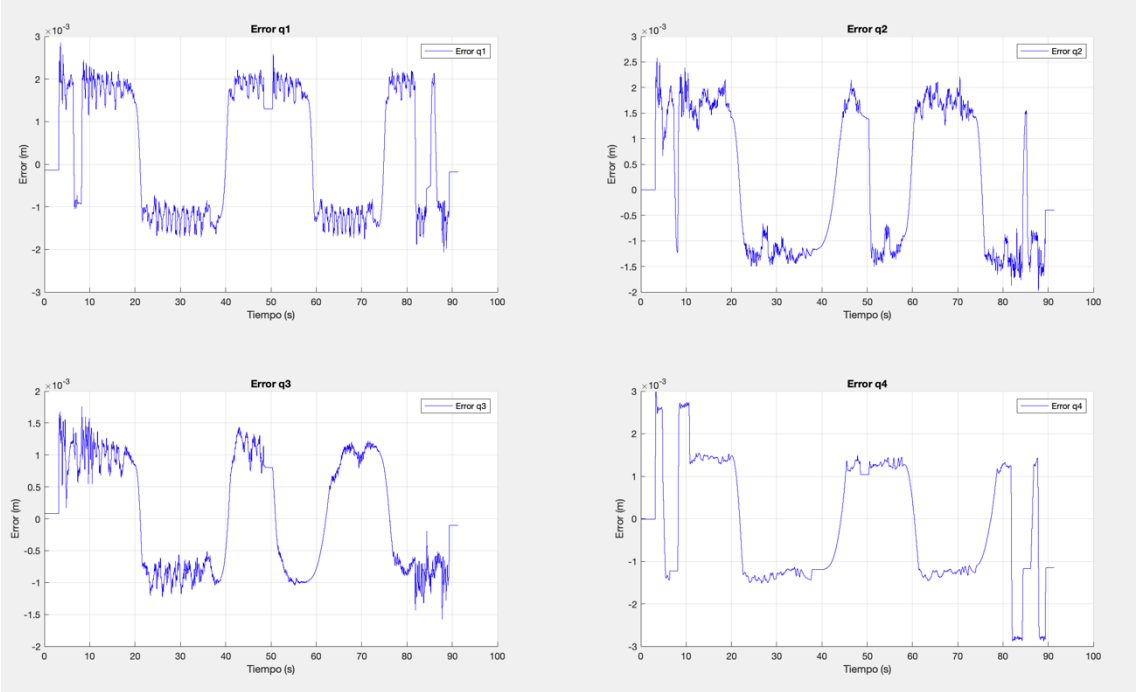


Figura 57: Experimento PD+G, trayectoria 1, error de posición

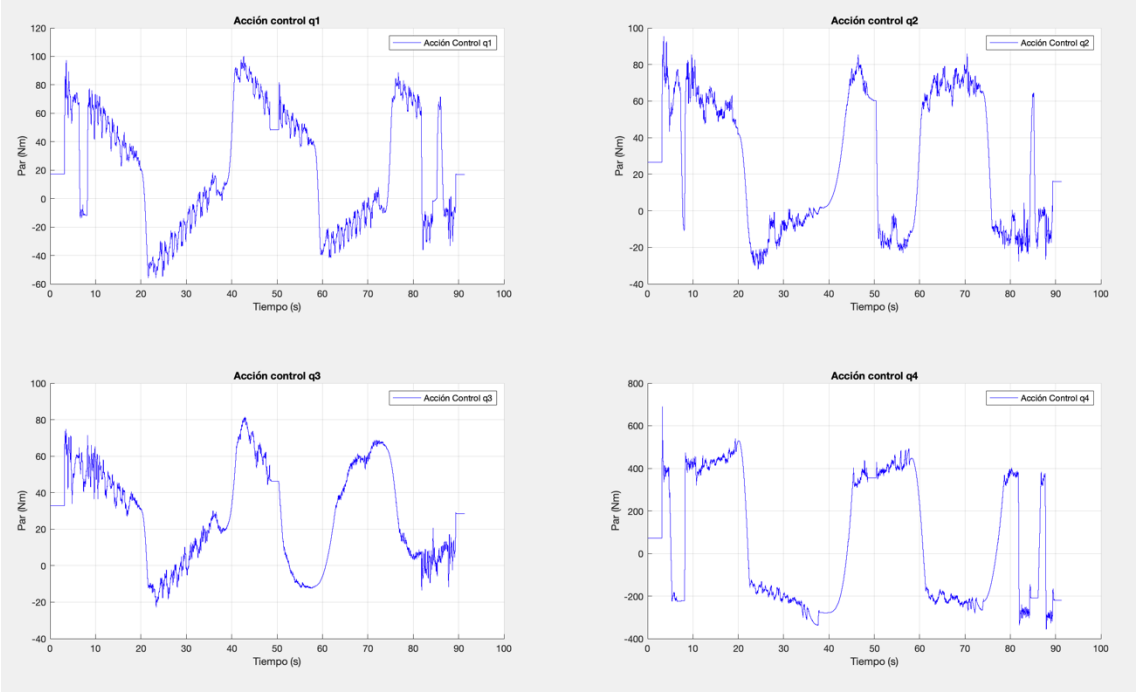


Figura 58: Experimento PD+G, trayectoria 1, acción de control

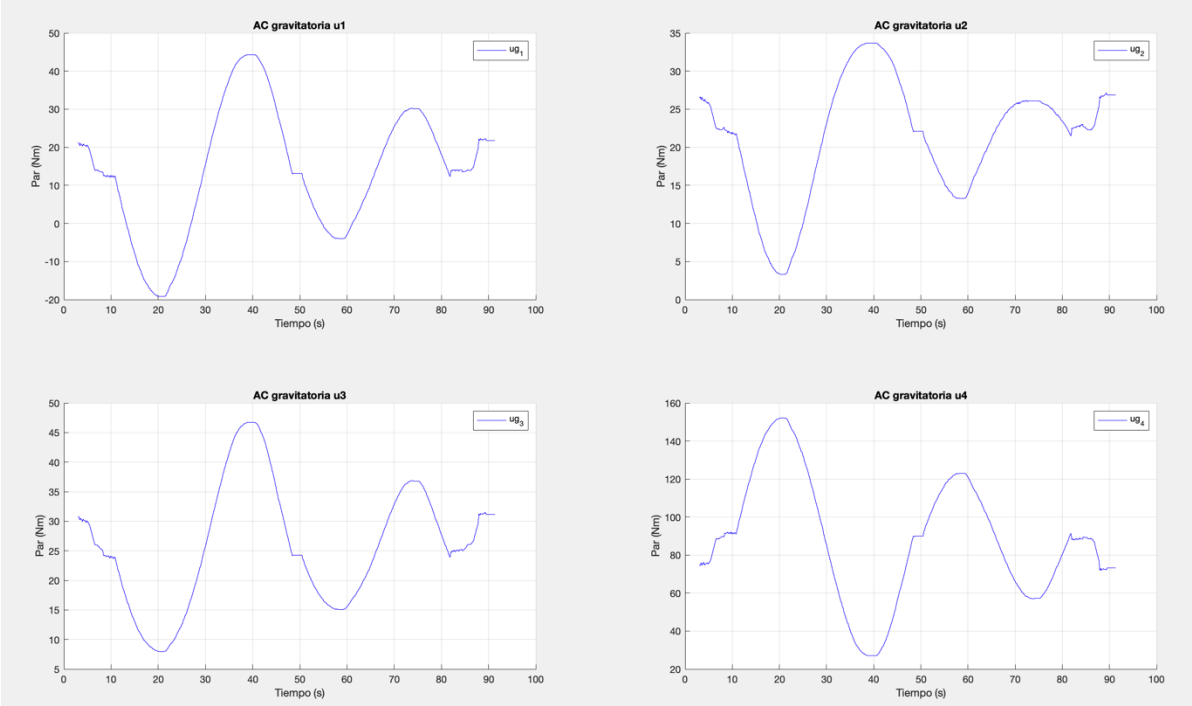


Figura 59: Experimento PD+G, trayectoria 1, términos gravitacionales

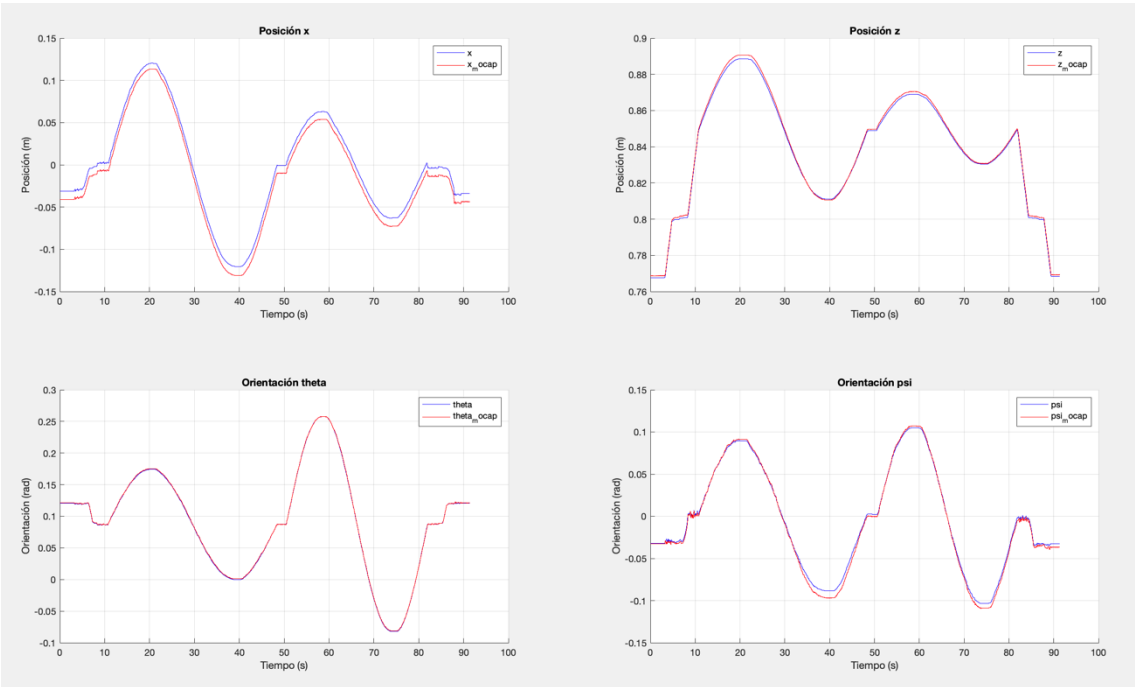


Figura 60: Experimento PD+G, trayectoria 1, posición de la plataforma móvil (modelo y obtenida por el sistema de captura de movimiento)

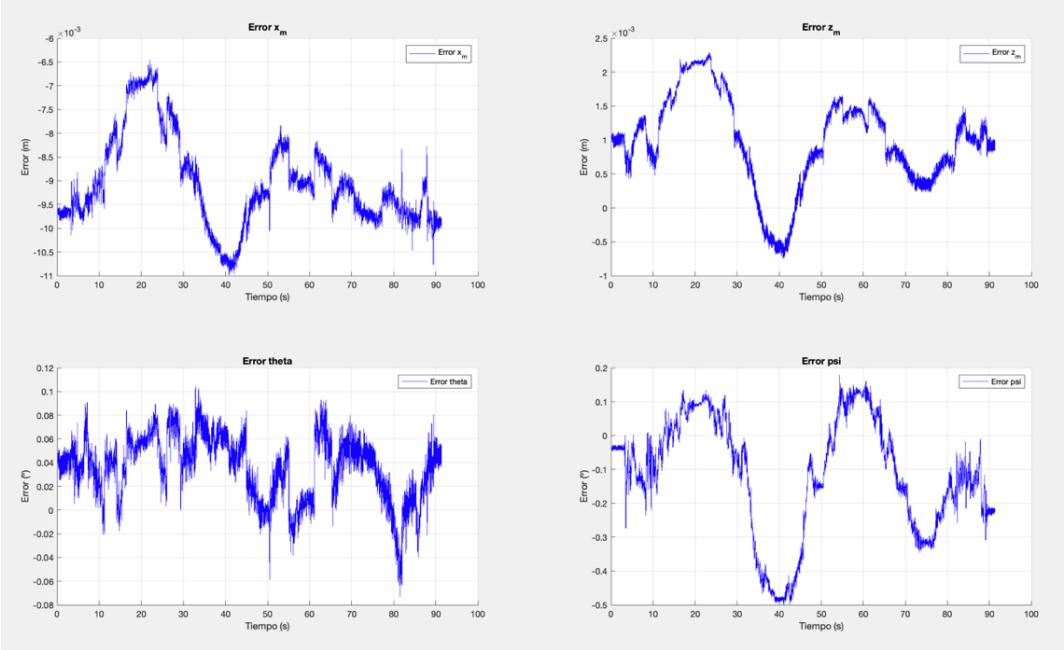


Figura 61: Experimento PD+G, trayectoria 1, error entre la posición cartesiana estimada por el modelo y la obtenida por Optitrack

En la siguiente gráfica se muestran los tiempos que tiene cada mensaje de ROS2. El tiempo de la acción de control debe ser el mayor puesto que es el último mensaje de la cadena que se transmite. Se puede observar que para este controlador hay una latencia de aproximadamente 1,6 ms entre que se leen los encoders y el momento en que se aplica la acción de control. A pesar de algunos instantes en los que el tiempo se demora más, los valores son adecuados y hay margen con respecto a los 10ms de periodo de muestreo.

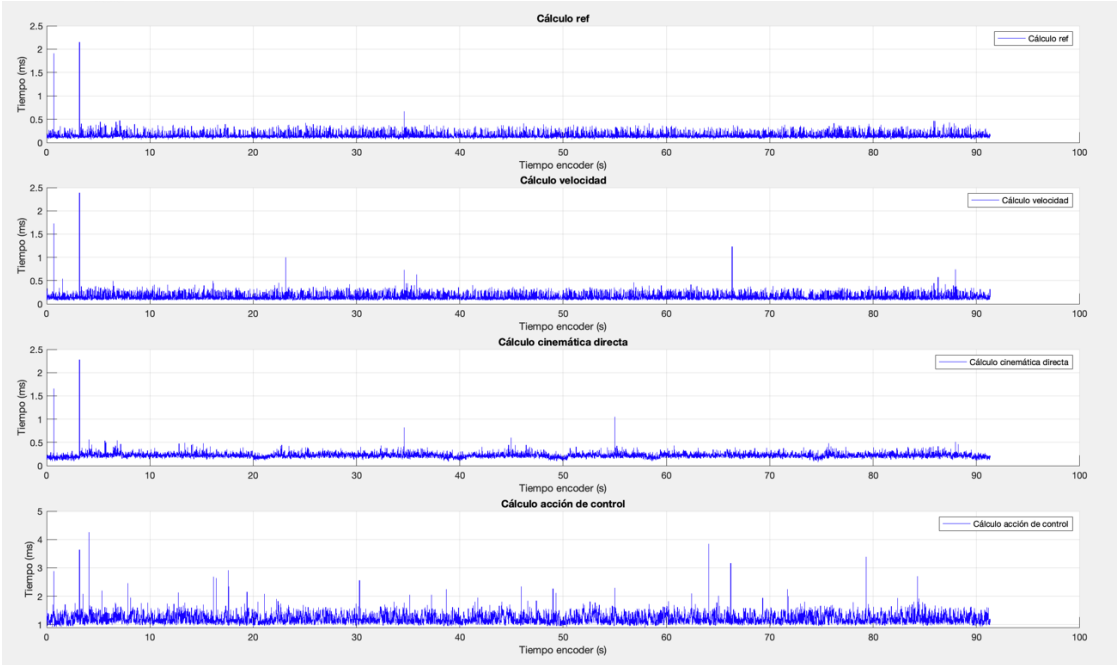


Figura 62: Experimento PD+G, trayectoria 1, tiempo de cálculo

Se ha probado el controlador con una segunda trayectoria en la que también se leen los datos de posición obtenidos por la captura de movimiento.

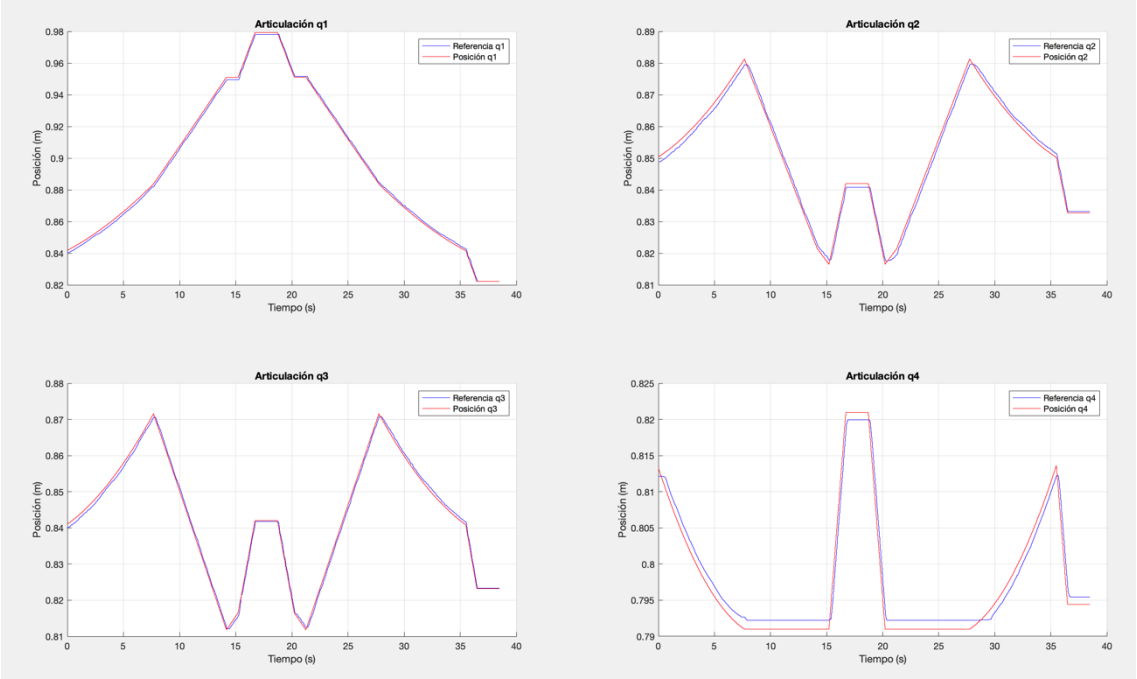


Figura 63: Experimento PD+G, trayectoria 2, posición de las articulaciones

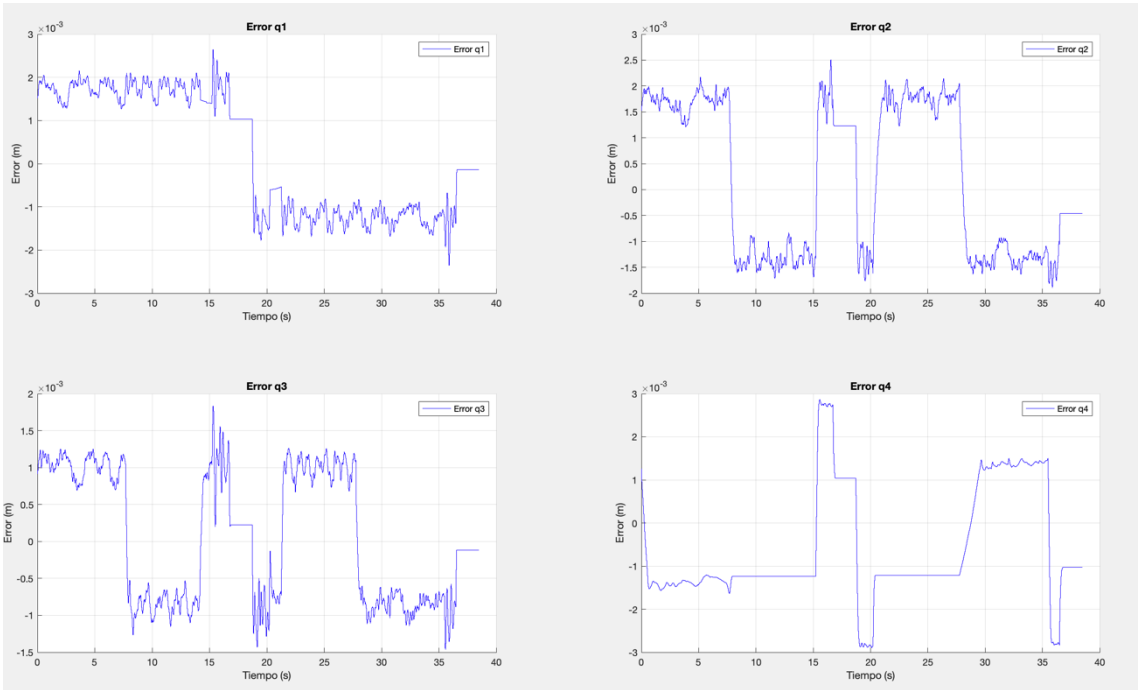


Figura 64: Experimento PD+G, trayectoria 2, error de posición

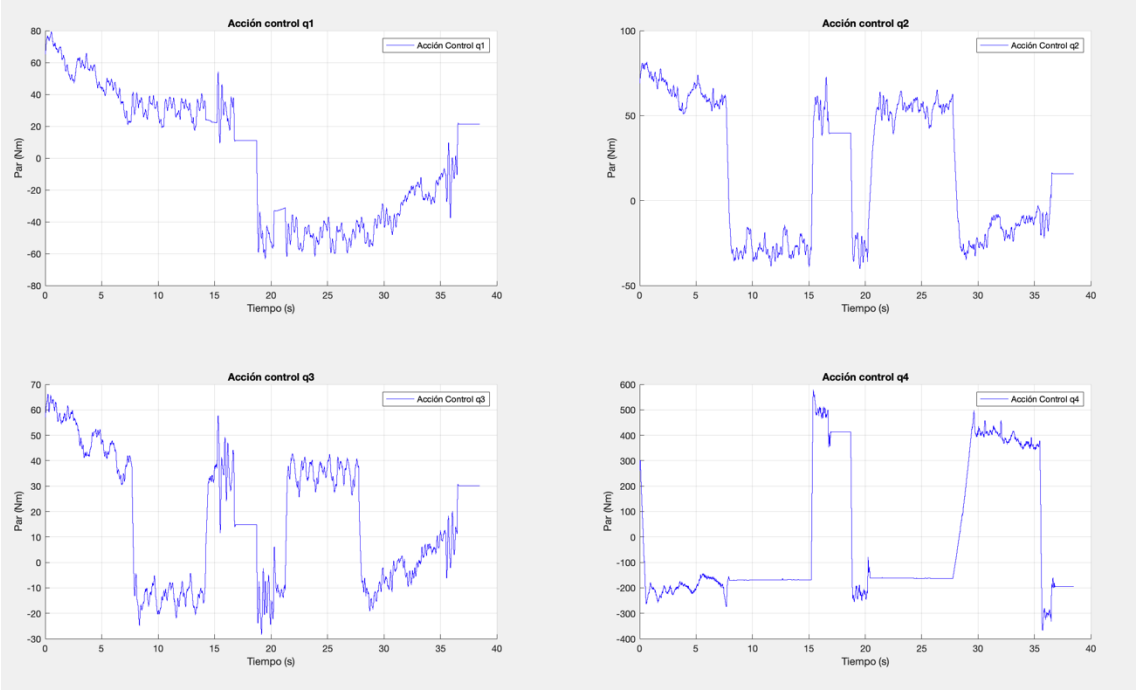


Figura 65: Experimento PD+G, trayectoria 2, acción de control

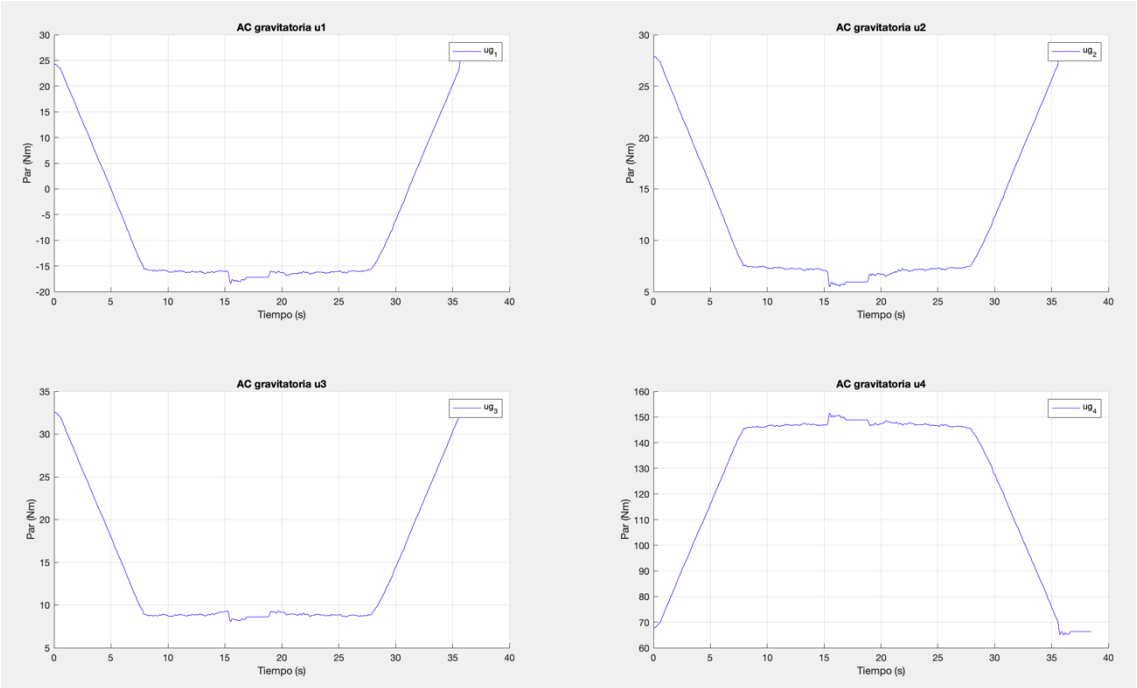


Figura 66: Experimento PD+G, trayectoria 2, términos gravitacionales

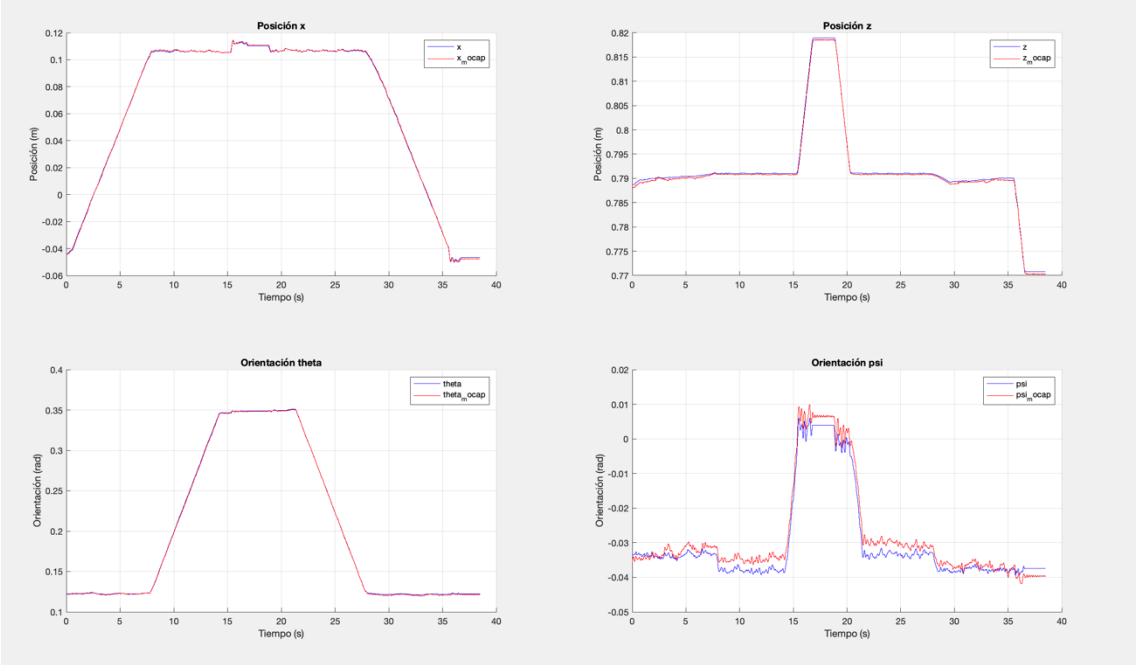


Figura 67: Experimento PD+G, trayectoria 2, posición de la plataforma móvil (modelo y obtenida por el sistema de captura de movimiento)

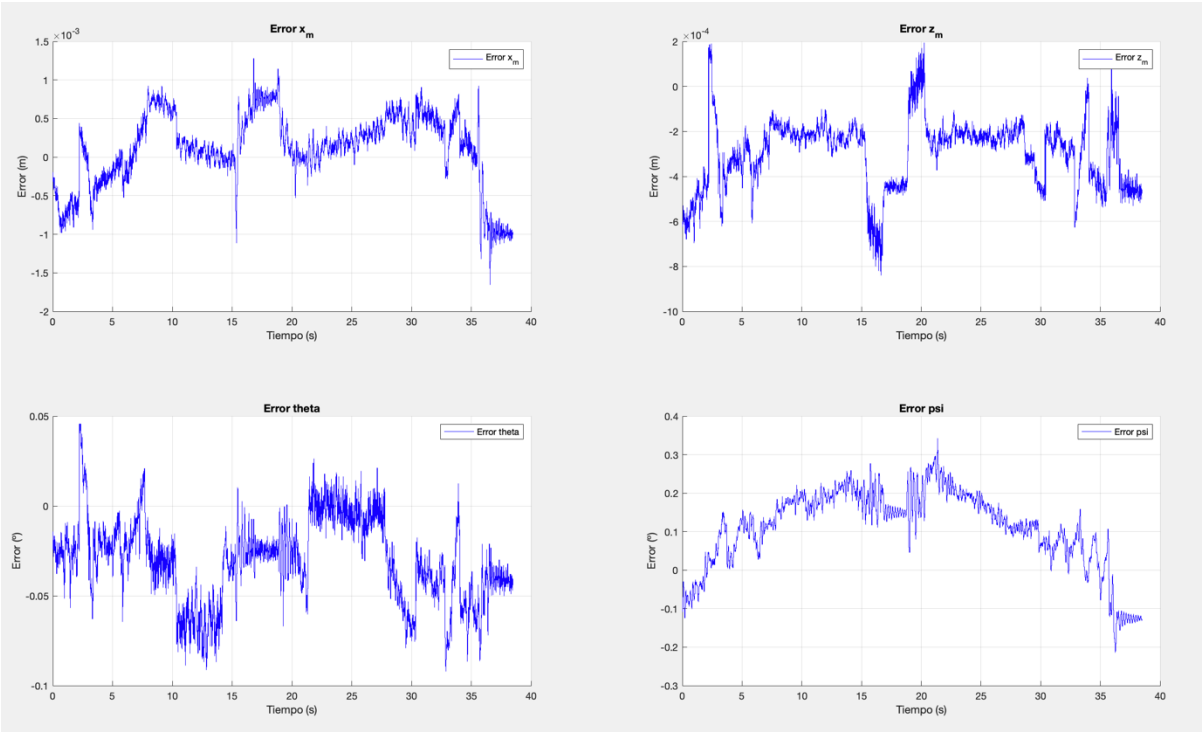


Figura 68: Experimento PD+G, trayectoria 2, error entre la posición cartesiana estimada por el modelo y la obtenida por OptiTrack

En cuanto a los tiempos de cálculo, se han obtenido valores similares a los de la trayectoria anterior, con un valor de latencia aproximado de 1,5 ms con algunos valores más altos.

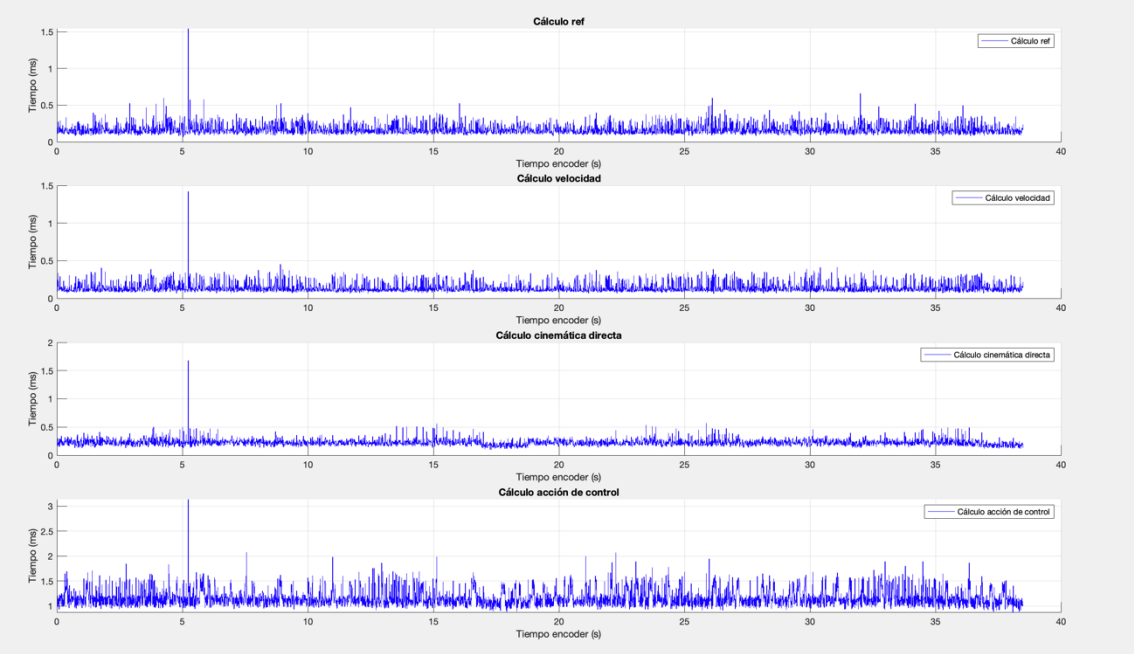


Figura 69: Experimento PD+G, trayectoria 2, tiempo de cálculo

6.1.2. CONTROLADOR ALGEBRAICO

La nueva versión del robot de 4 grados de libertad fue montada durante el trascurso del proyecto. Antes de colocar la plataforma móvil, se comprobó que el control de cada pata funciona utilizando el controlador algebraico. El resultado del movimiento de la pata 1 es el siguiente:

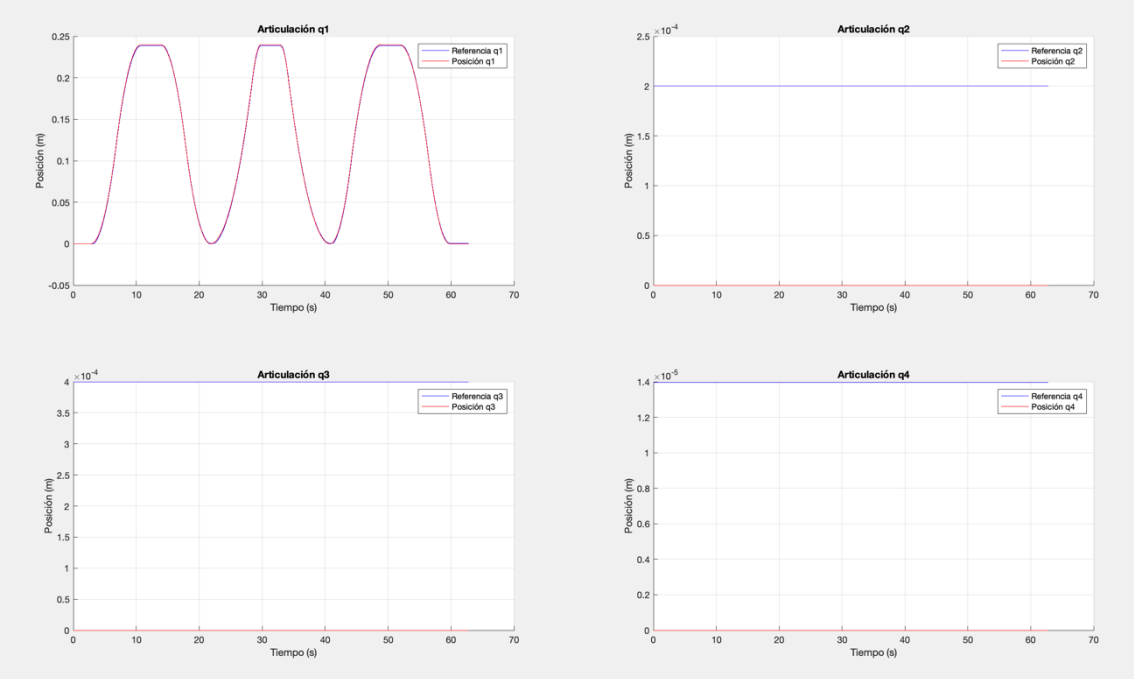


Figura 70: Experimento controlador algebraico, trayectoria pata 1, posición

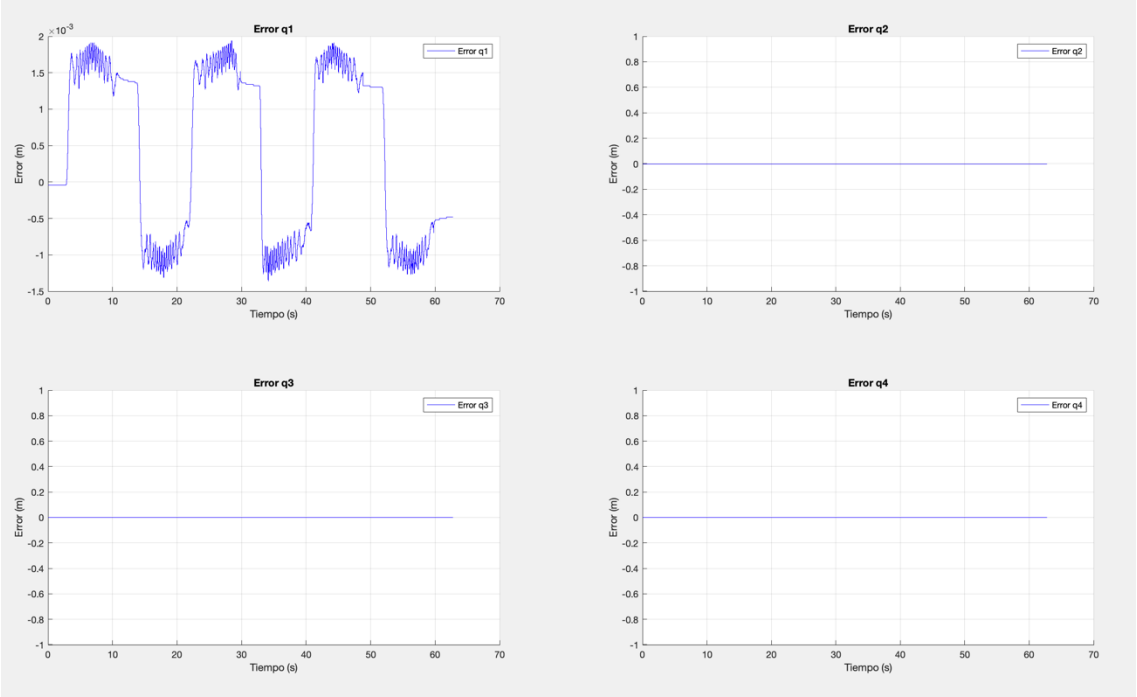


Figura 71: Experimento controlador algebraico, trayectoria pata 1, error de posición

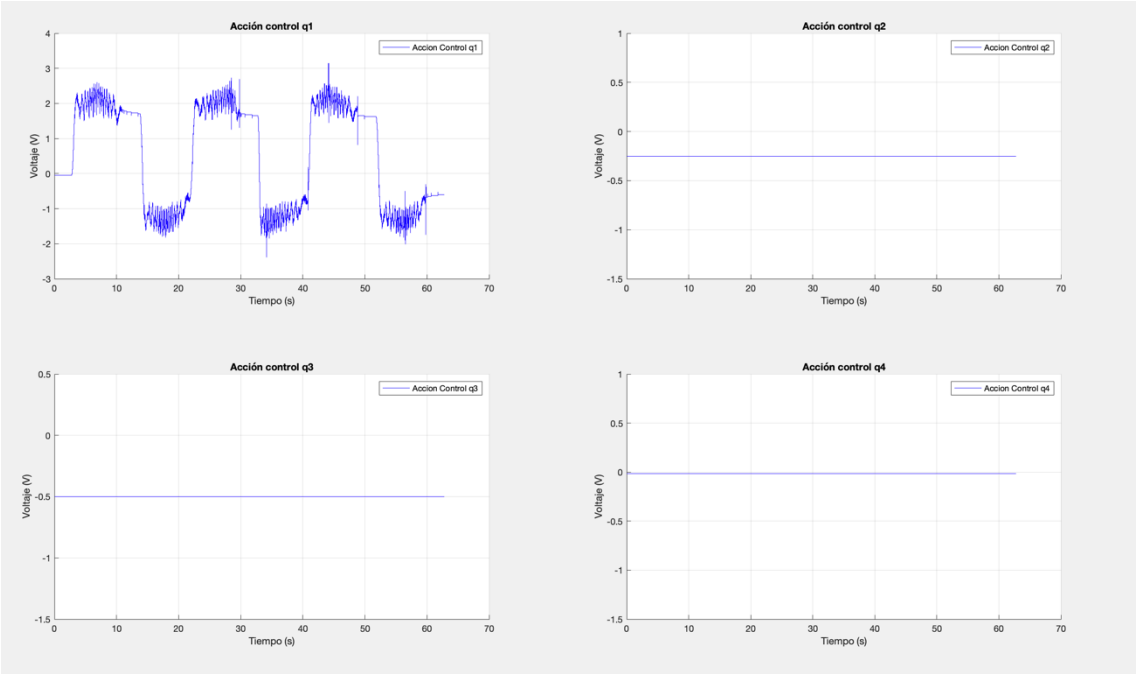


Figura 72: Experimento controlador algebraico, trayectoria pata 1, acción de control

Este controlador obtiene valores de latencia algo menores a los del controlador PD+G puesto que este es más sencillo, tiene menos nodos y los cálculos son más directos. La latencia se puede observar que es de aproximadamente 1,1ms.

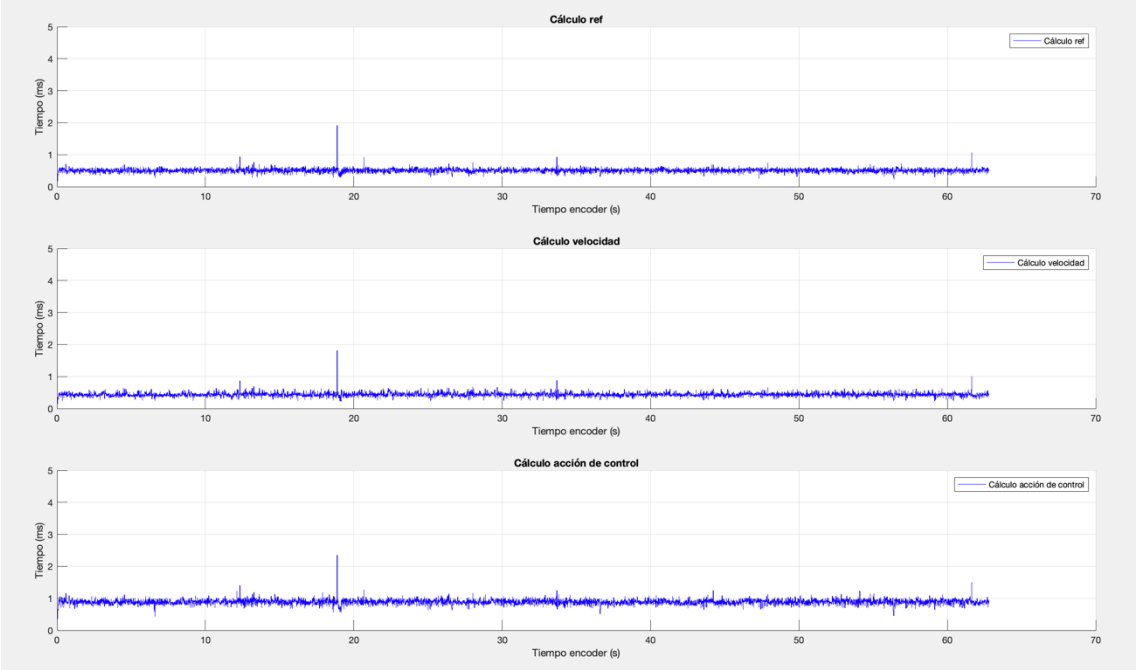


Figura 73: Experimento controlador algebraico, trayectoria pata 1, tiempo de cálculo

Una vez comprobado el funcionamiento de las 4 patas por separado, se montó la plataforma y se realizó una trayectoria que mueve la plataforma en el eje z:

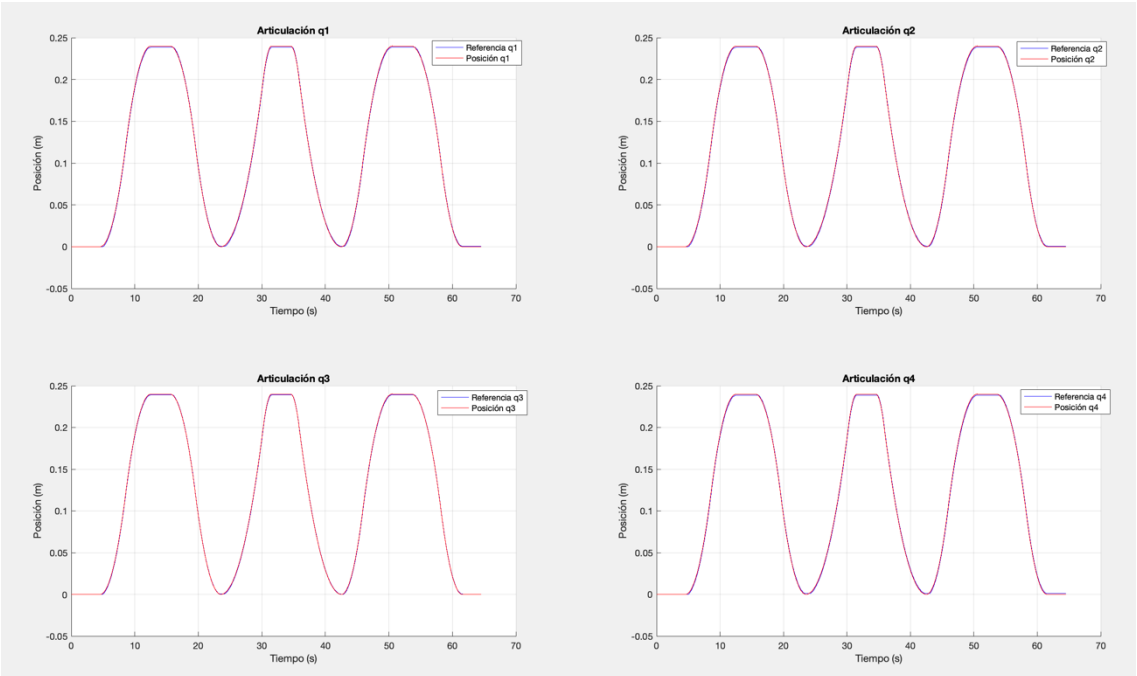


Figura 74: Experimento controlador algebraico, trayectoria en z, posición

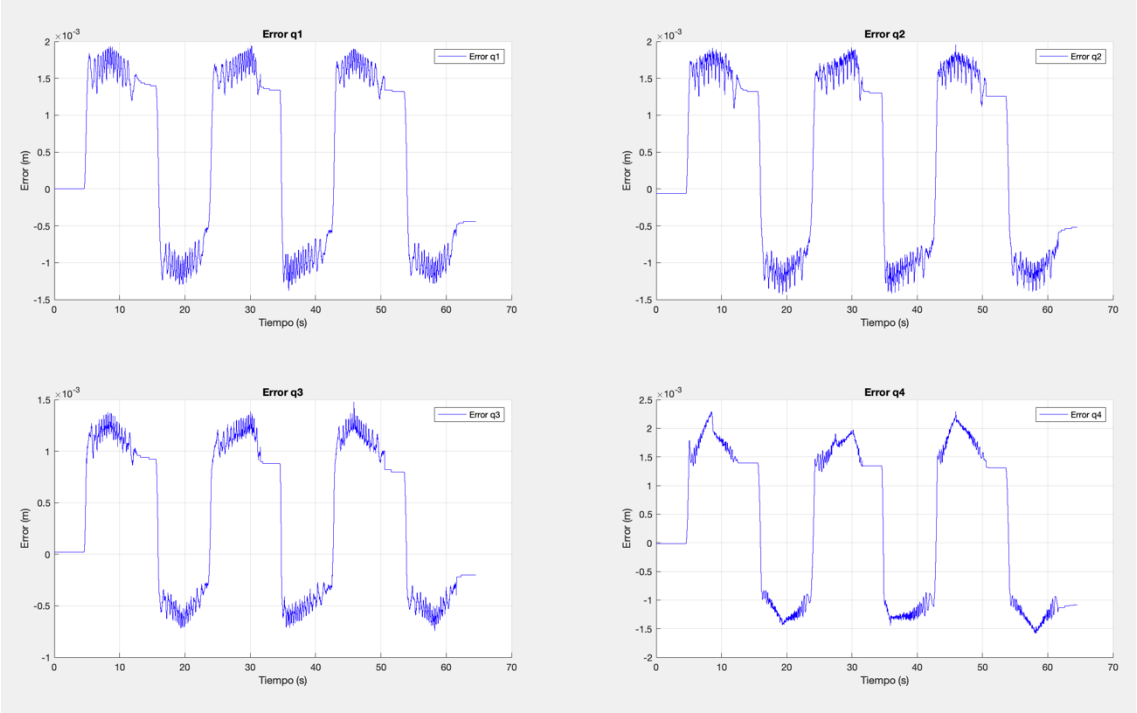


Figura 75: Experimento controlador algebraico, trayectoria en z, error de posición

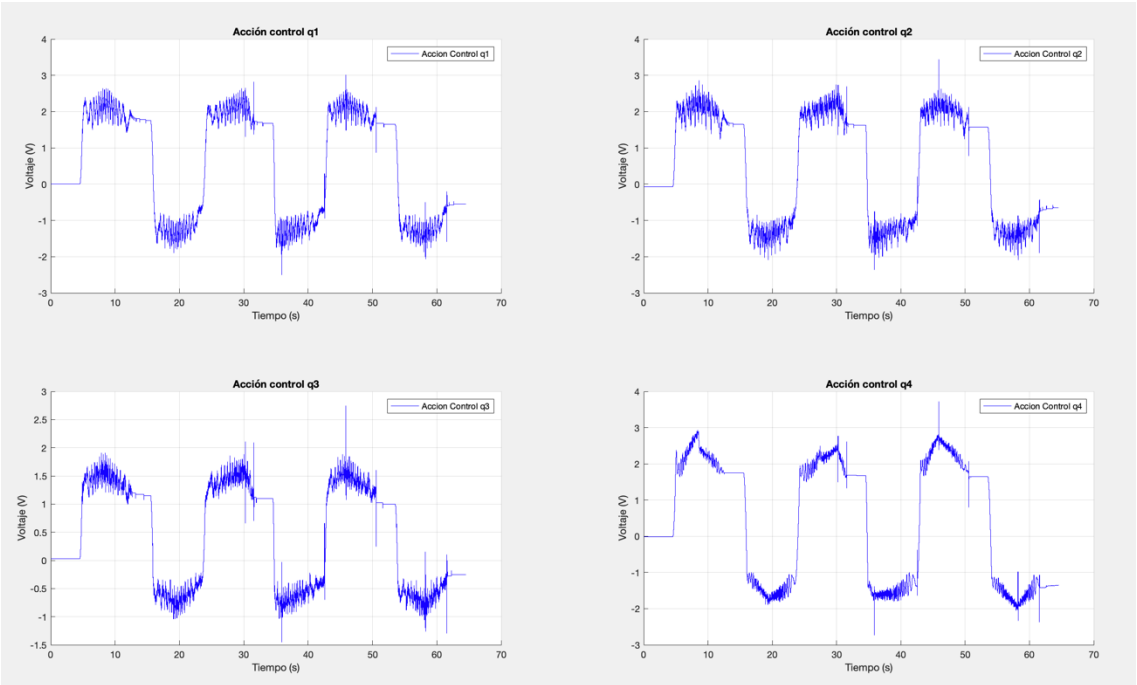


Figura 76: Experimento controlador algebraico, trayectoria en z, acción de control

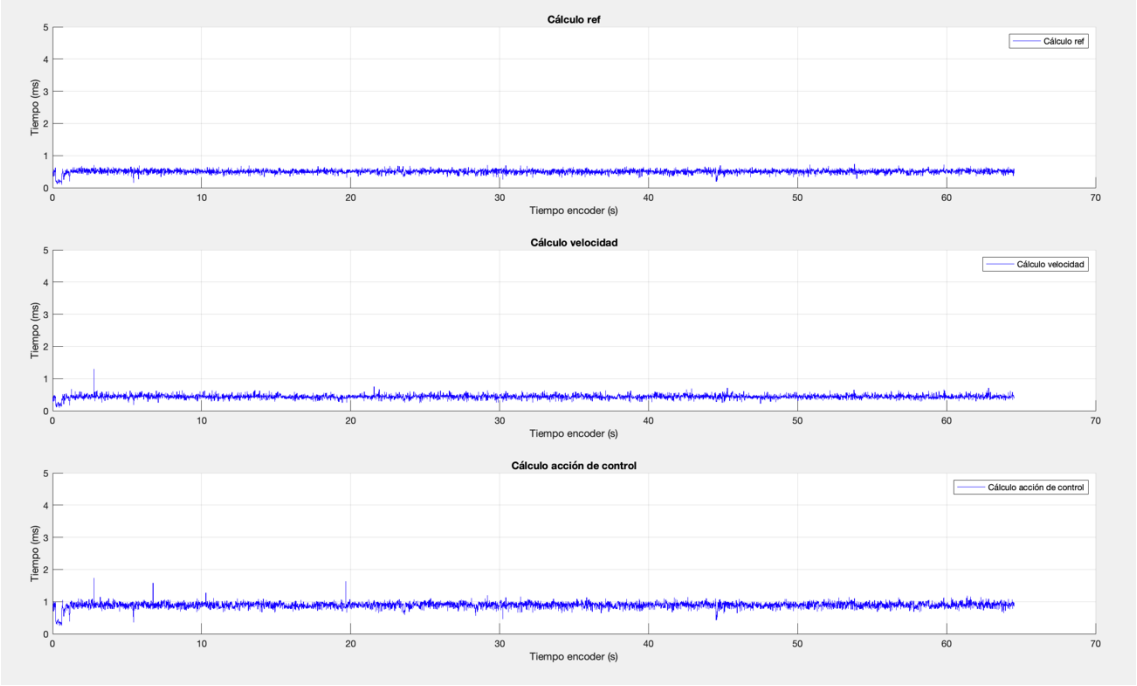


Figura 77: Experimento controlador algebraico, trayectoria en z, tiempo de cálculo

Este controlador algebraico es el que se ha utilizado en el siguiente apartado para controlar el bucle interno de posición en el controlador para escapar de una singularidad.

6.1.3. ESCAPE DE UNA SINGULARIDAD

Se ha validado experimentalmente el controlador para salir de una singularidad en el robot de 4 grados de libertad. Se ha probado el funcionamiento con diferentes trayectorias que se van a mostrar a continuación. Como se ha mencionado anteriormente, el bucle interno utiliza el controlador algebraico de posición.

Antes de cada experimento, se han calibrado las cámaras del sistema de captura de movimiento de Optitrack debido a que, aunque estas siempre se encuentren en la misma posición, son muy sensibles a pequeños movimientos como vibraciones. El proceso de calibración dura alrededor de 5 minutos y es necesario mover alrededor de la escena una barra con marcadores y después, colocar una escuadra de calibración con marcadores que darán lugar al origen de coordenadas del mundo. Los marcadores de estas piezas tienen que ser vistos por la mayor parte de cámaras posibles.

Figura 78: Barra de calibración de Optitrack. Fuente: https://www.optitrack.com/accessories/calibration-tools/

Figura 79: Escuadra de calibración. Fuente: https://www.optitrack.com/accessories/calibration-tools/

En los tres experimentos realizados, la metodología es la misma, el robot parte desde una posición que no es singular y se traslada a una que sí que lo es. Una vez esté en la posición singular, se activa el controlador para escapar de ella, este detecta que efectivamente la posición es singular y traslada el robot a una posición segura. El momento en el que se activa el controlador coincide con el instante en el que hay un cambio de referencia en dos de las patas con respecto a la referencia original de posición.

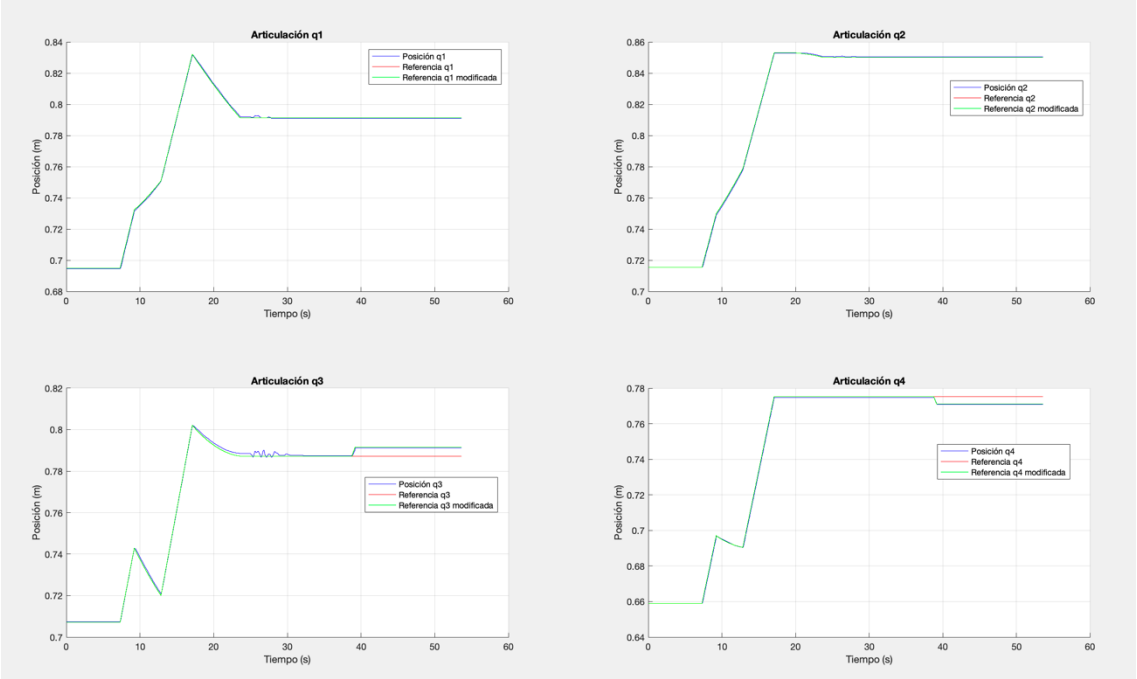


Figura 80: Escape de una singularidad, experimento 1, posición de las articulaciones

En la siguiente gráfica, se compara la estimación del modelo de la posición cartesiana de la plataforma móvil y de la posición detectada por el sistema de captura de movimiento (con estos datos son con los que se realimenta el sistema). En el segundo 25, se movió la plataforma móvil manualmente para comprobar que el robot se encuentra en una singularidad (gana un grado de libertad y este movimiento no produce un desplazamiento en los actuadores).

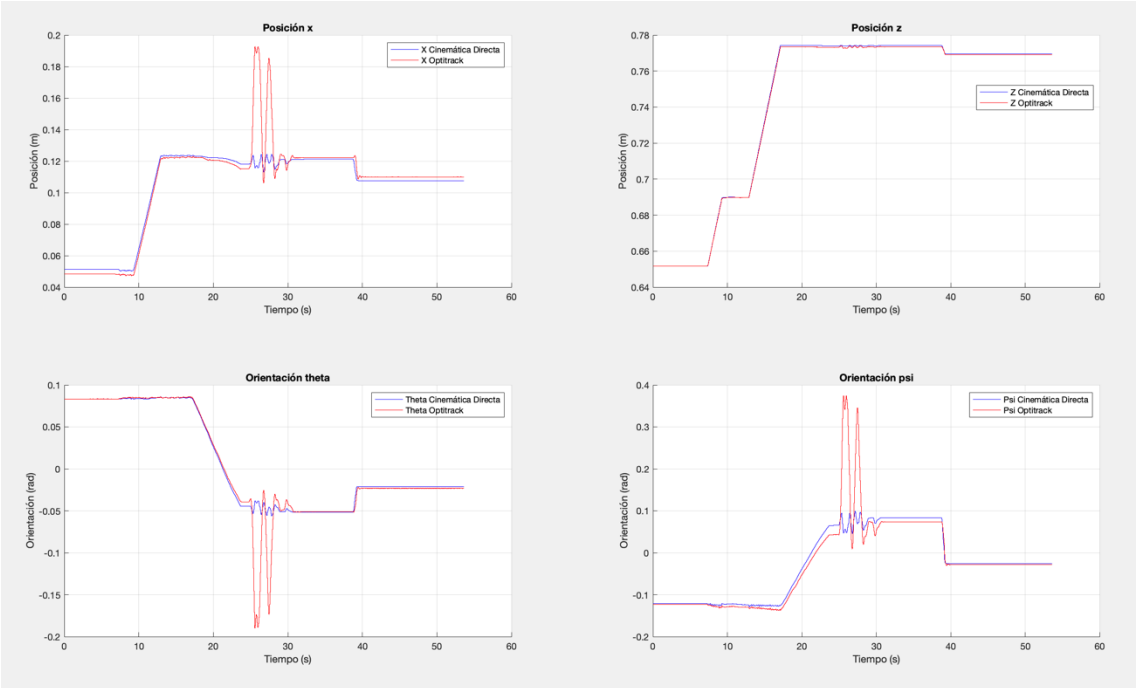


Figura 81: Escape de una singularidad, experimento 1, posición de la plataforma móvil estimada por la cinemática directa y por el sistema de visión

En esta siguiente gráfica se muestra el determinante del jacobiano directo. Se puede apreciar que su valor desciende al comienzo (se está trasladando al robot a una posición singular) y después, cuando se activa el controlador de escape, este aumenta.

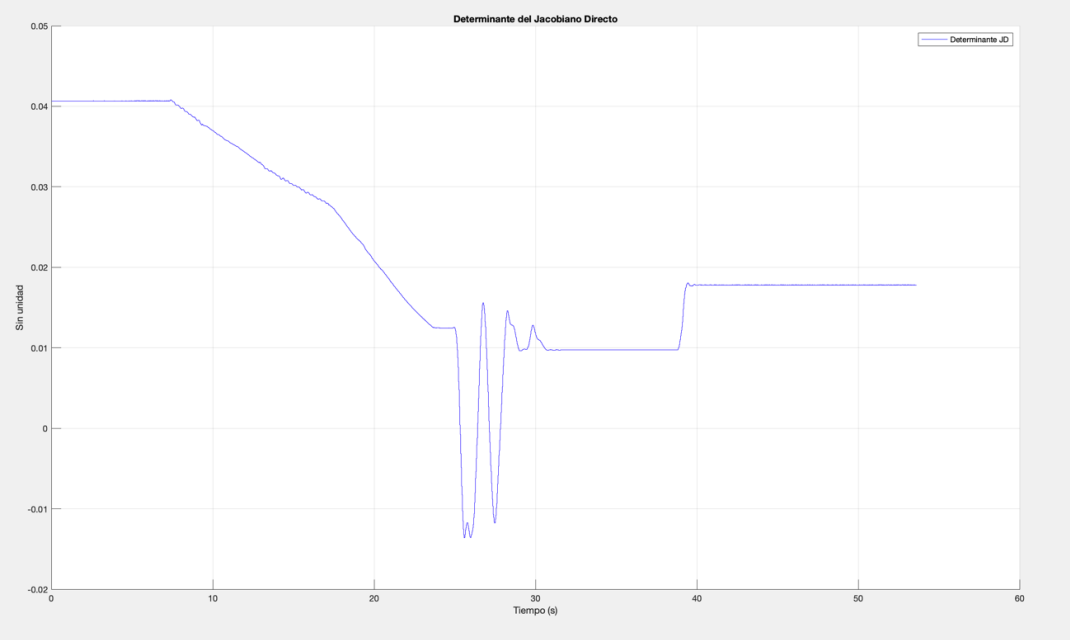


Figura 82: Escape de una singularidad, experimento 1, determinante del jacobiano directo

Esta gráfica muestra los desplazamientos de posición sumados a la referencia original para salir de la singularidad. Justo antes del segundo 40, se ha detectado que las patas 3 y 4 son las causantes de producir la singularidad y se han aplicado los incrementos de posición a al referencia original.

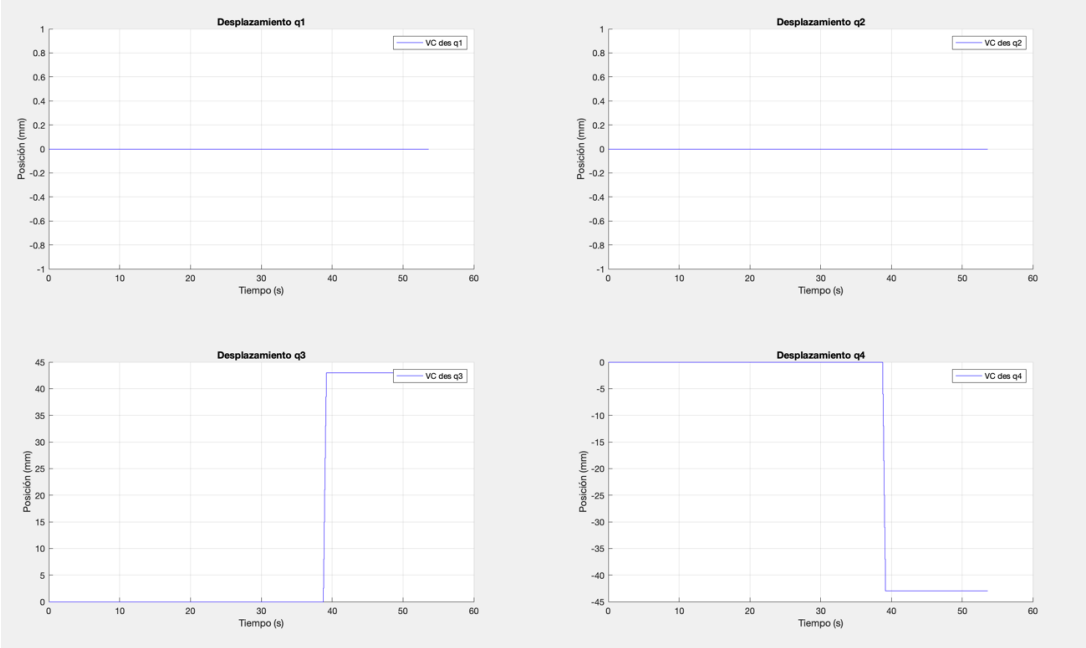


Figura 83: Escape de una singularidad, experimento 1, desplazamientos aplicados para escapar de la singularidad

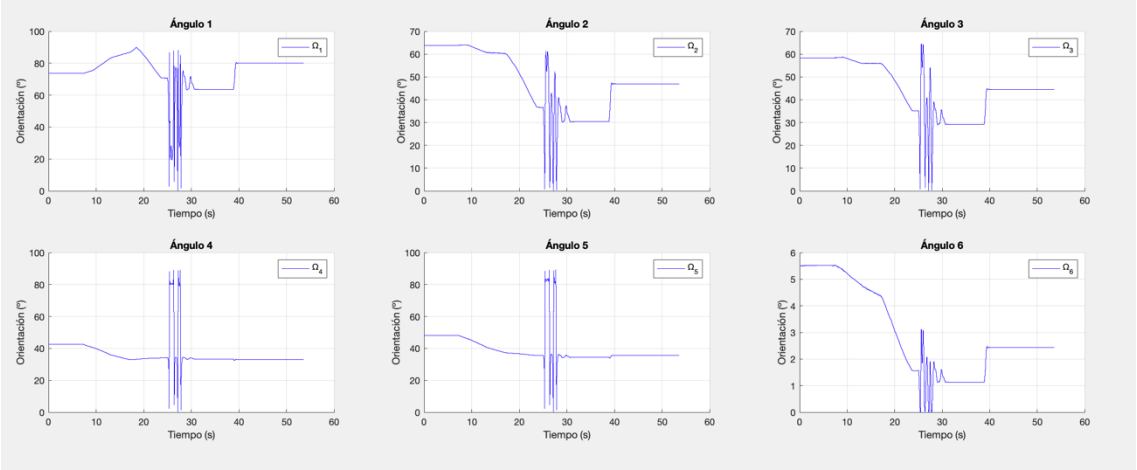


Figura 84: Escape de una singularidad, experimento 1, ángulos OTS

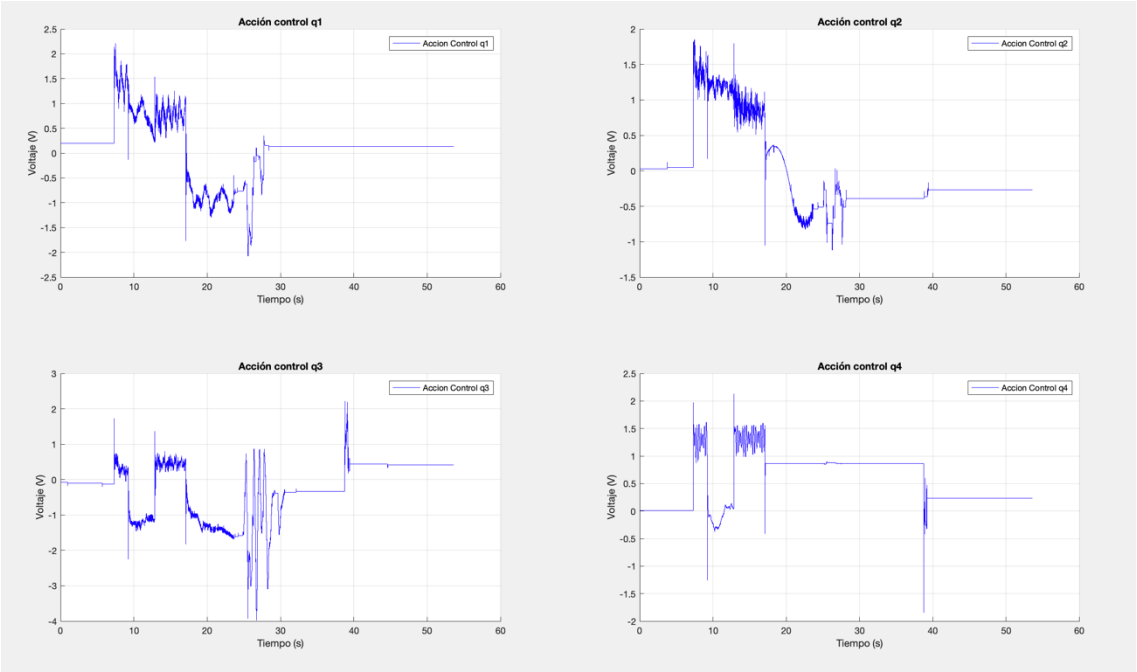


Figura 85: Escape de una singularidad, experimento 1, acción de control

Las siguientes gráficas muestran el segundo experimento realizado (con una trayectoria distinta), la metodología es la misma que la explicada anteriormente.

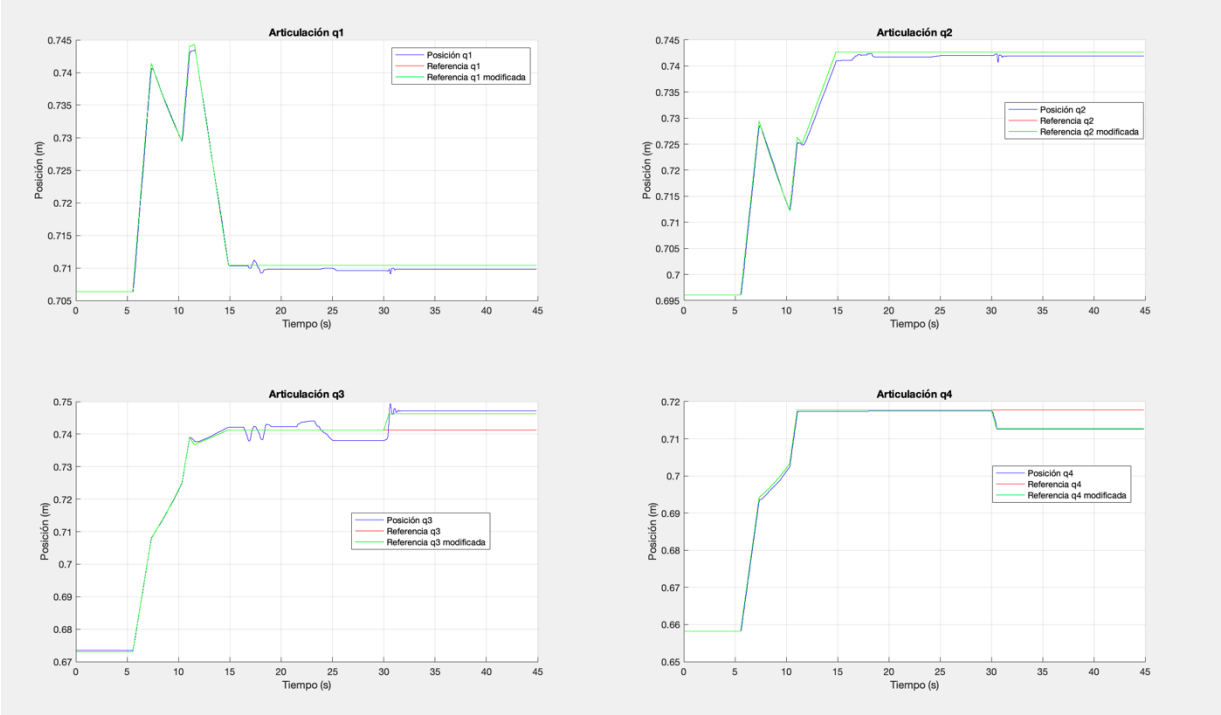


Figura 86: Escape de una singularidad, experimento 2, posición de las articulaciones

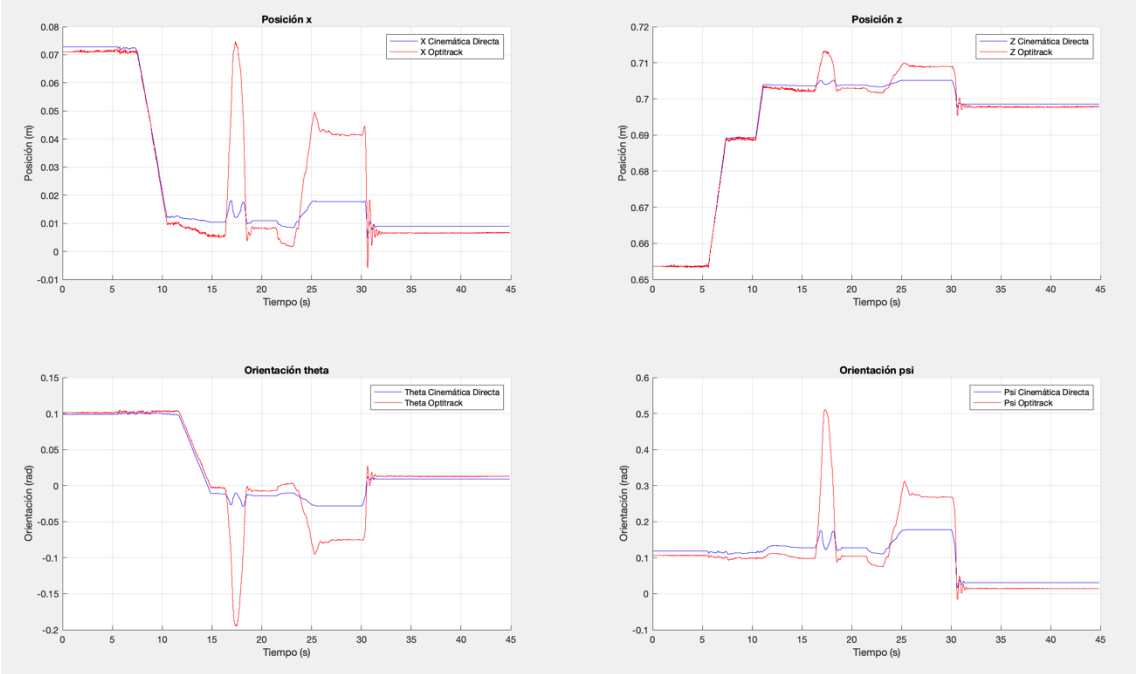


Figura 87: Escape de una singularidad, experimento 2, posición de la plataforma móvil estimada por la cinemática directa y por el sistema de visión

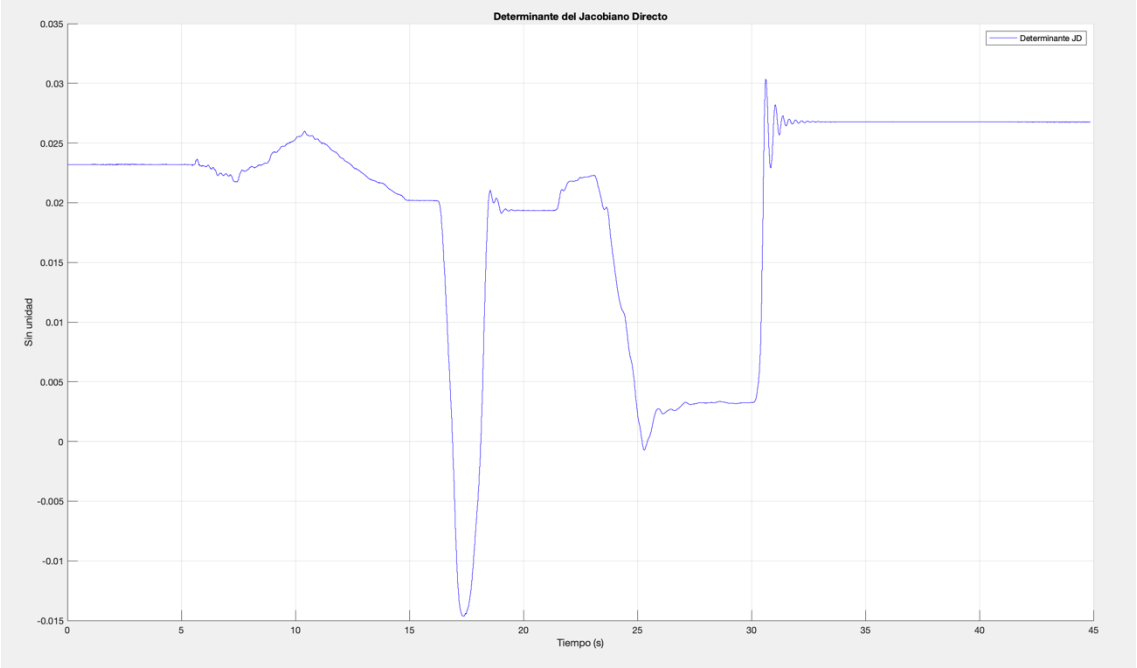


Figura 88: Escape de una singularidad, experimento 2, determinante del jacobiano directo

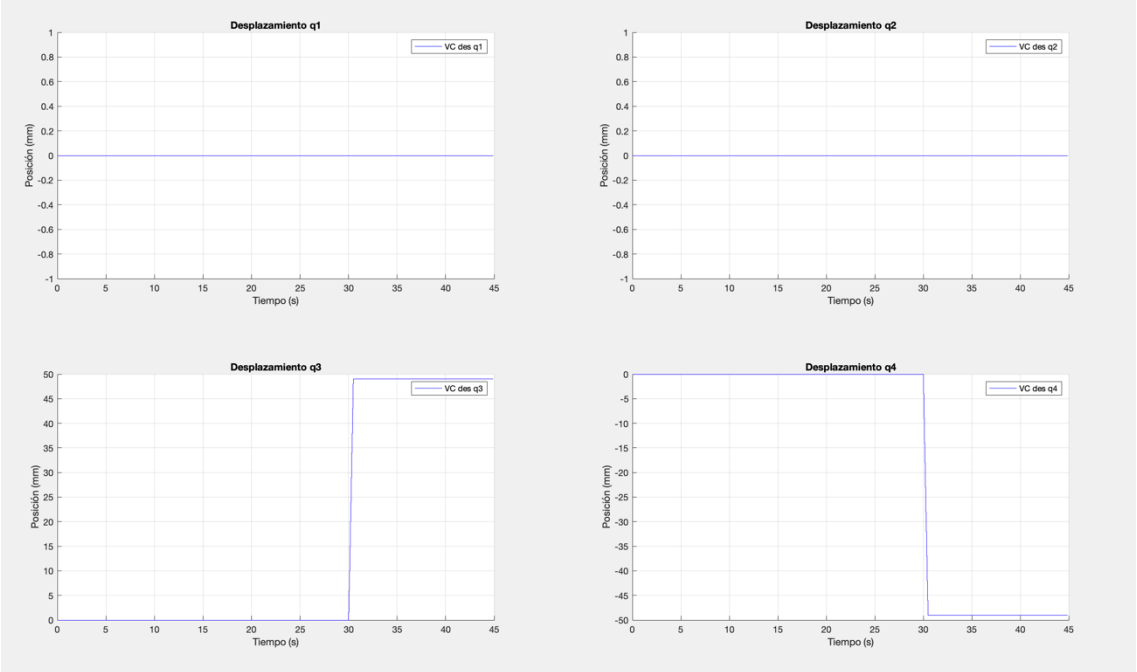


Figura 89: Escape de una singularidad, experimento 2, desplazamientos aplicados para escapar de la singularidad

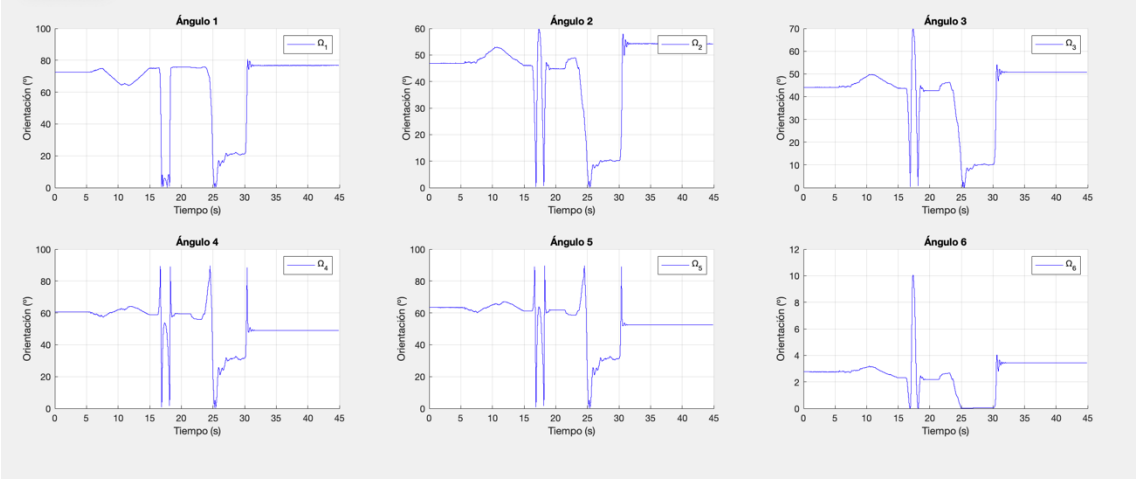


Figura 90: Escape de una singularidad, experimento 2, ángulos OTS

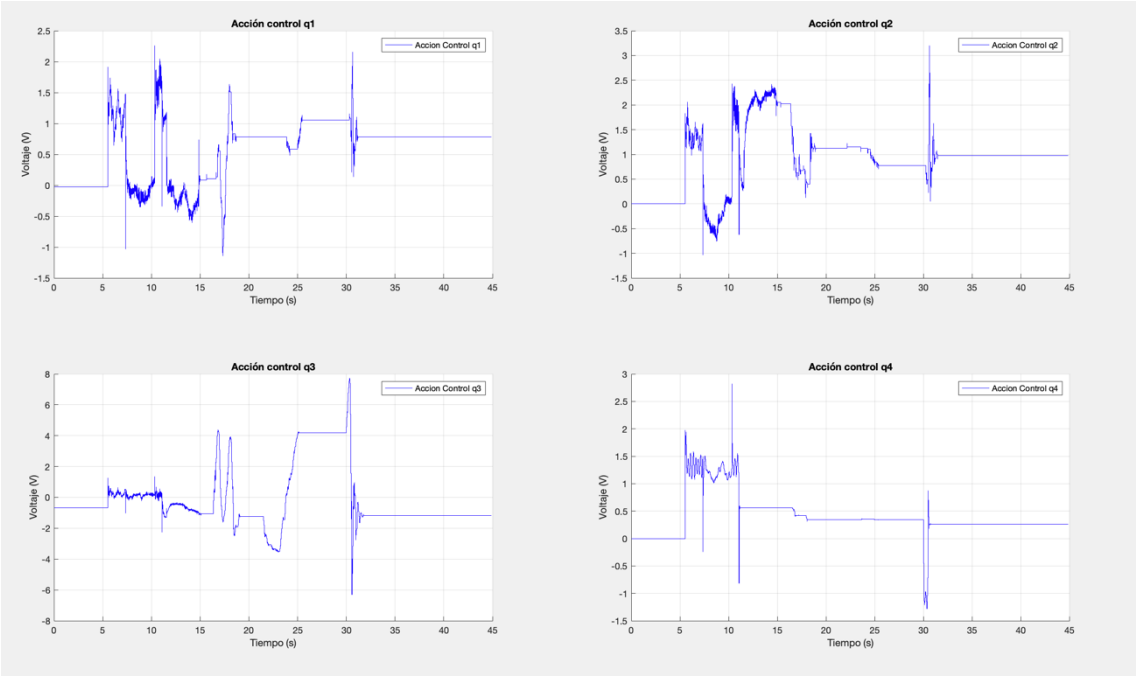


Figura 91: Escape de una singularidad, experimento 2, acción de control

Por último, se muestran los gráficos correspondientes al tercer experimento realizado.

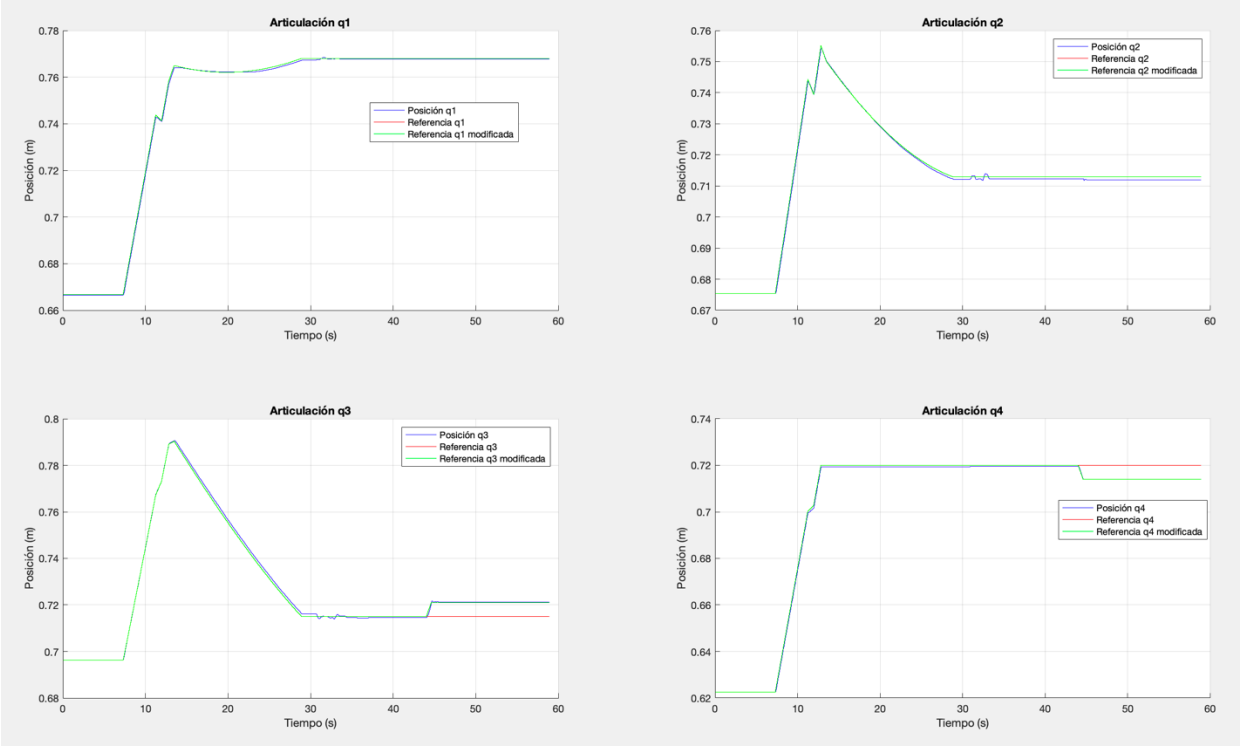


Figura 92: Escape de una singularidad, experimento 3, posición de las articulaciones

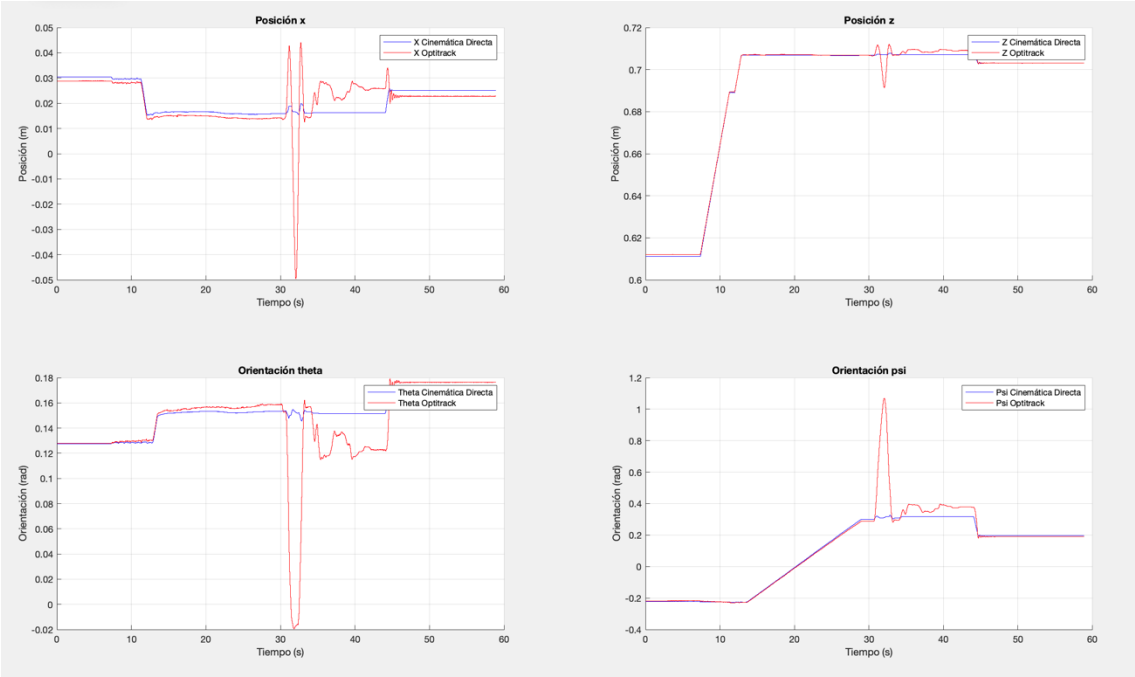


Figura 93: Escape de una singularidad, experimento 3, posición de la plataforma móvil estimada por la cinemática directa y por el sistema de visión

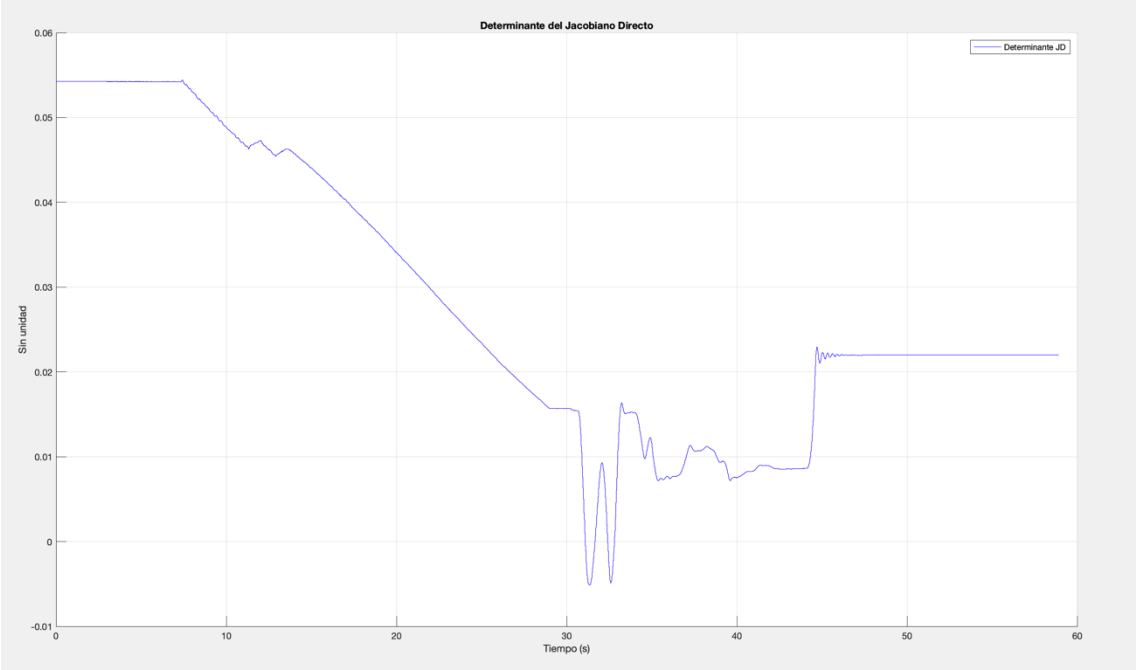


Figura 94: Escape de una singularidad, experimento 3, determinante del jacobiano directo

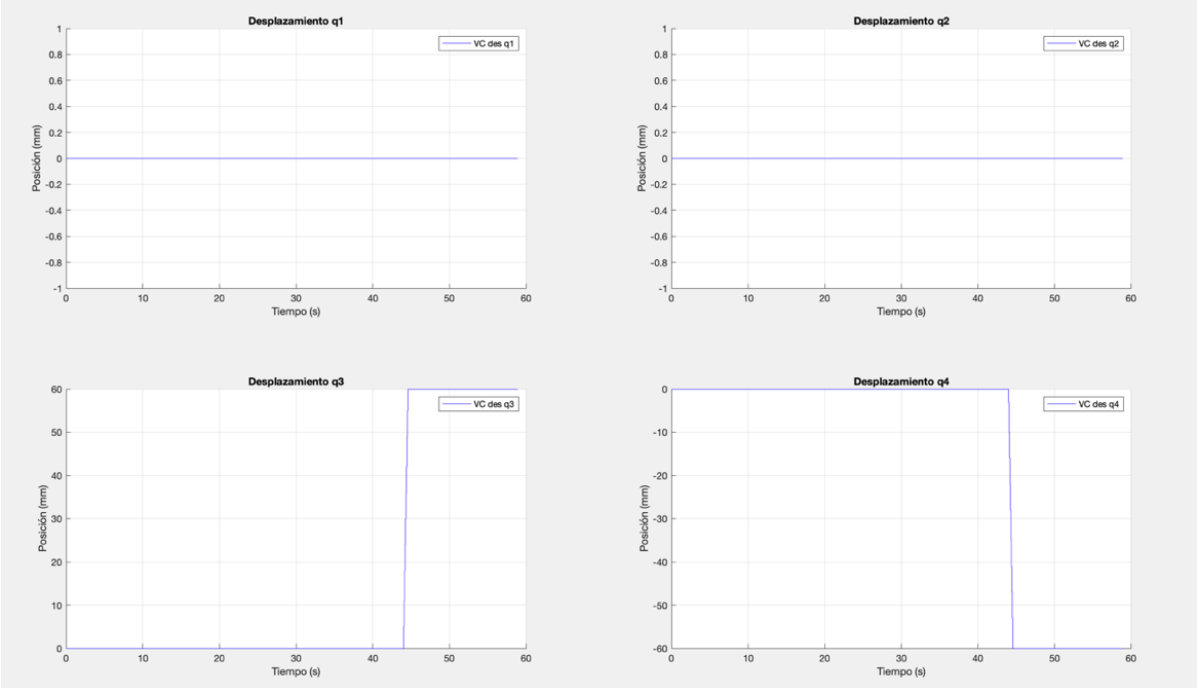


Figura 95: Escape de una singularidad, experimento 3, desplazamientos aplicados para escapar de la singularidad

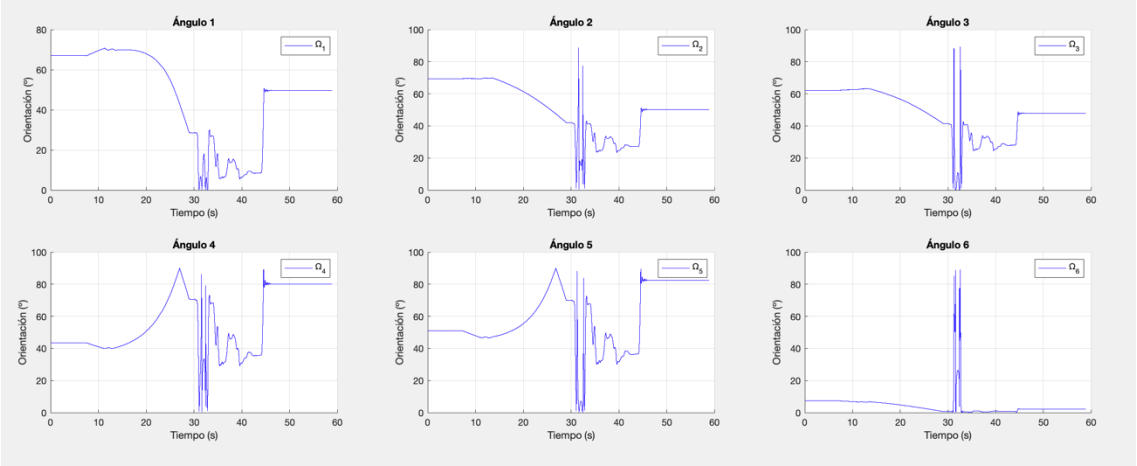


Figura 96: Escape de una singularidad, experimento 3, ángulos OTS

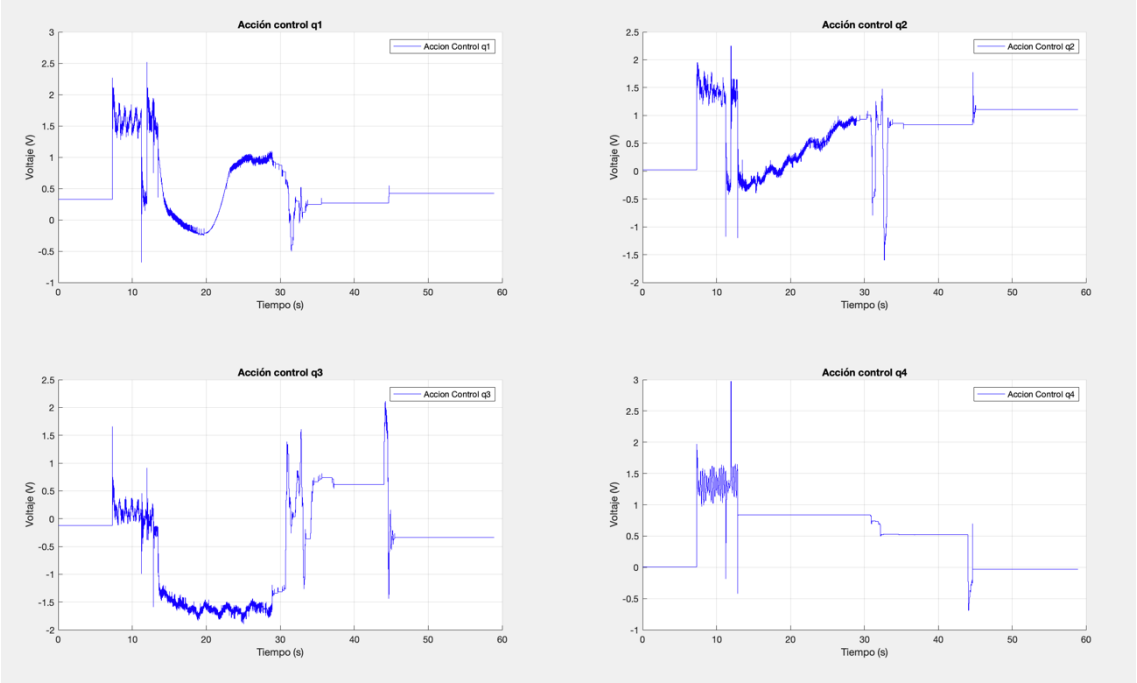


Figura 97: Escape de una singularidad, experimento 3, acción de control

6.2. CONTROL DEL ROBOT DE 3 GRADOS DE LIBERTAD

6.2.1. PD CON COMPENSACIÓN DE LA GRAVEDAD

Antes de realizar el experimento que se muestra a continuación, se han obtenido las constantes del controlador PD+G (proporcional y de velocidad) de manera experimental. Se han tenido en cuenta dos criterios u objetivos, que el error de posición sea el mínimo posible y que también los incrementos de la acción de control sean lo mínimo posible. Finalmente, se ha encontrado una solución (las constantes indicadas en la tabla, la acción de control emitida está en valores de par, después se divide por la

constante de conversión de par a voltaje 382,3318Nm/V) que cumple con ambos criterios y es la que se muestra en las siguientes gráficas:

	Articulación 1	Articulación 2	Articulación 3
Kp	350000	350000	350000
Kv	7000	7000	7000

Tabla 7: Constantes del controlador PD+G para el robot de 3DOF

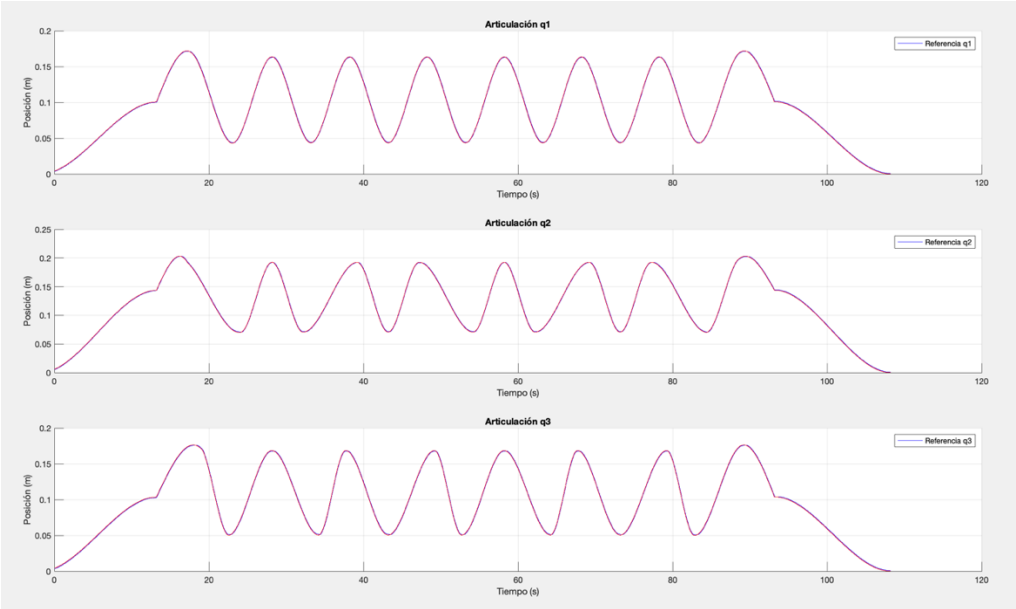


Figura 98: Experimento controlador PD+G para 3DOF, posición

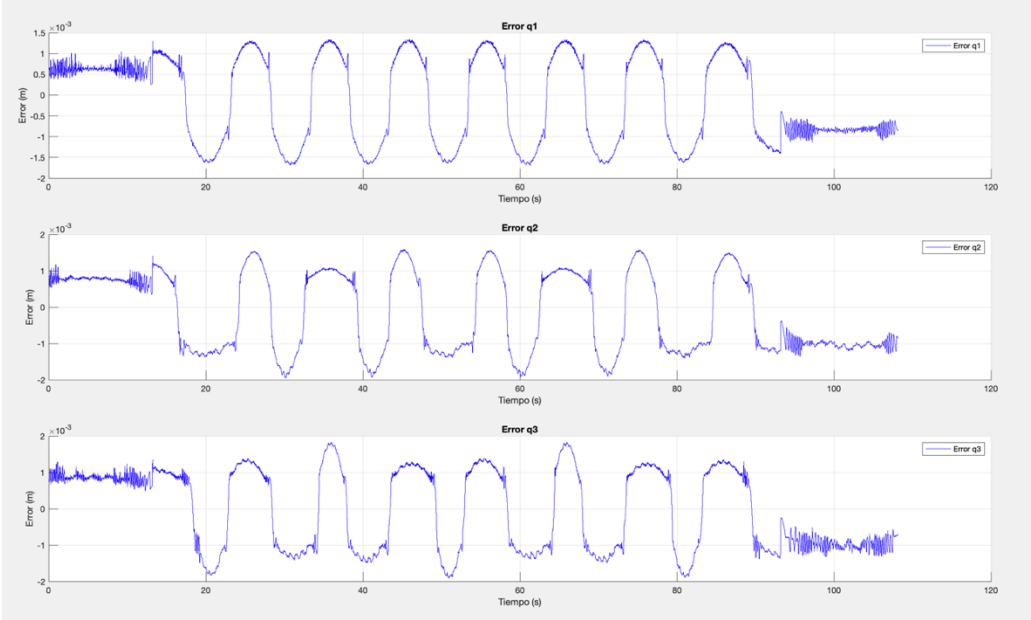


Figura 99: Experimento controlador PD+G para 3DOF, error de posición

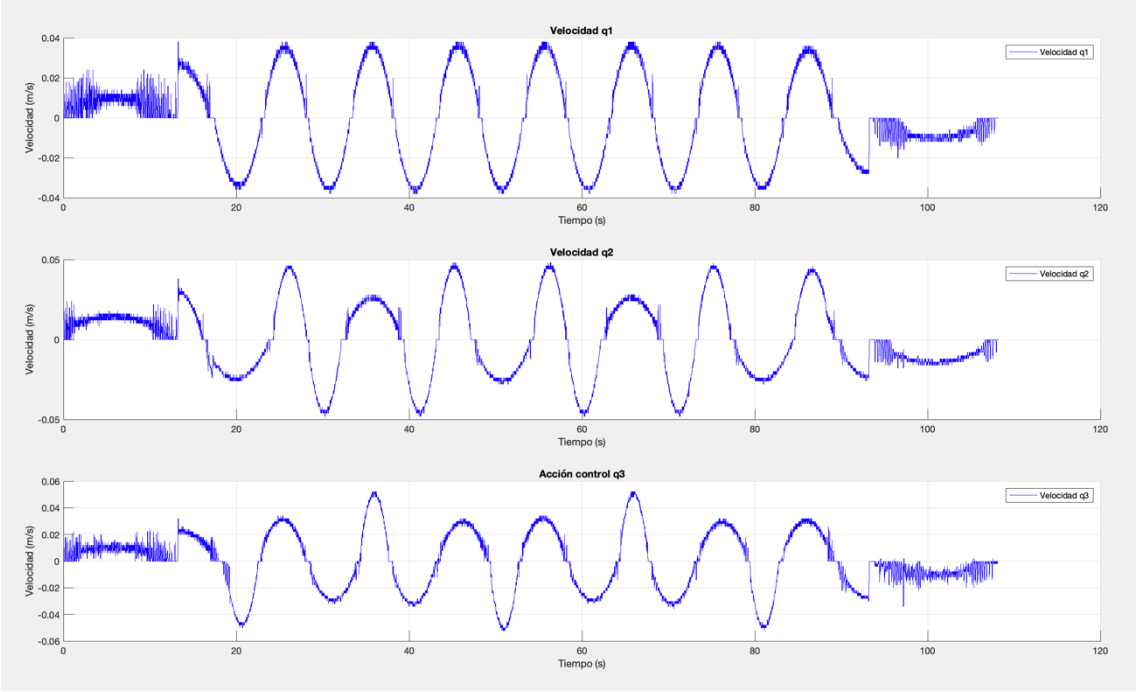


Figura 100: Experimento controlador PD+G para 3DOF, velocidad

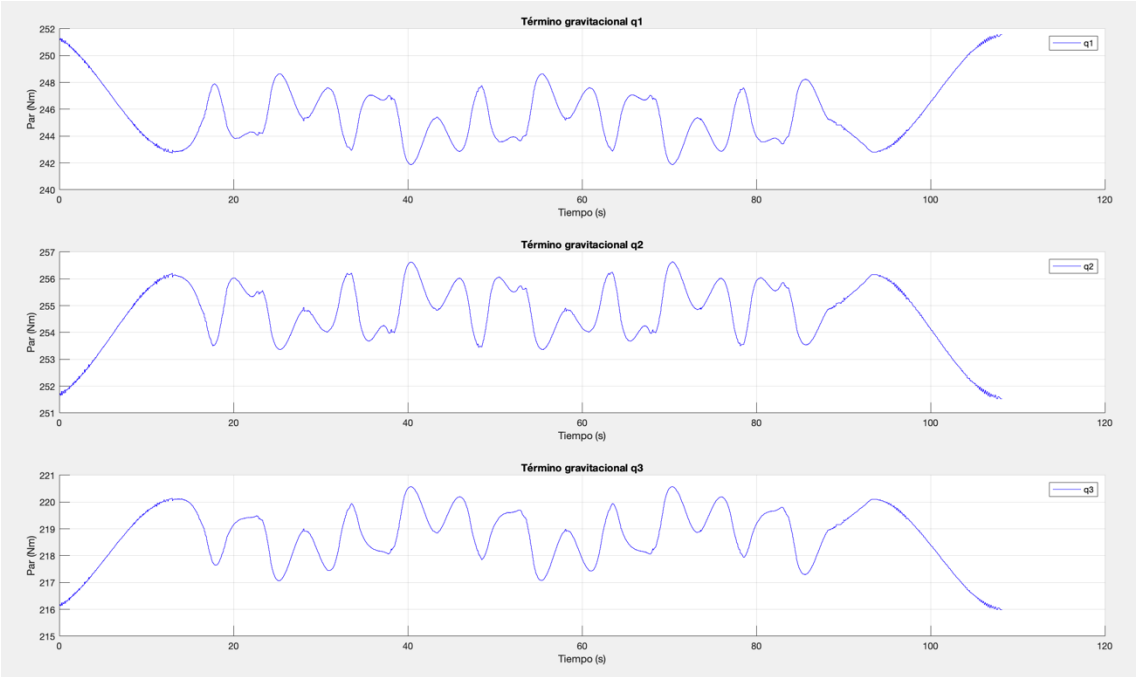


Figura 101: Experimento controlador PD+G para 3DOF, términos gravitacionales

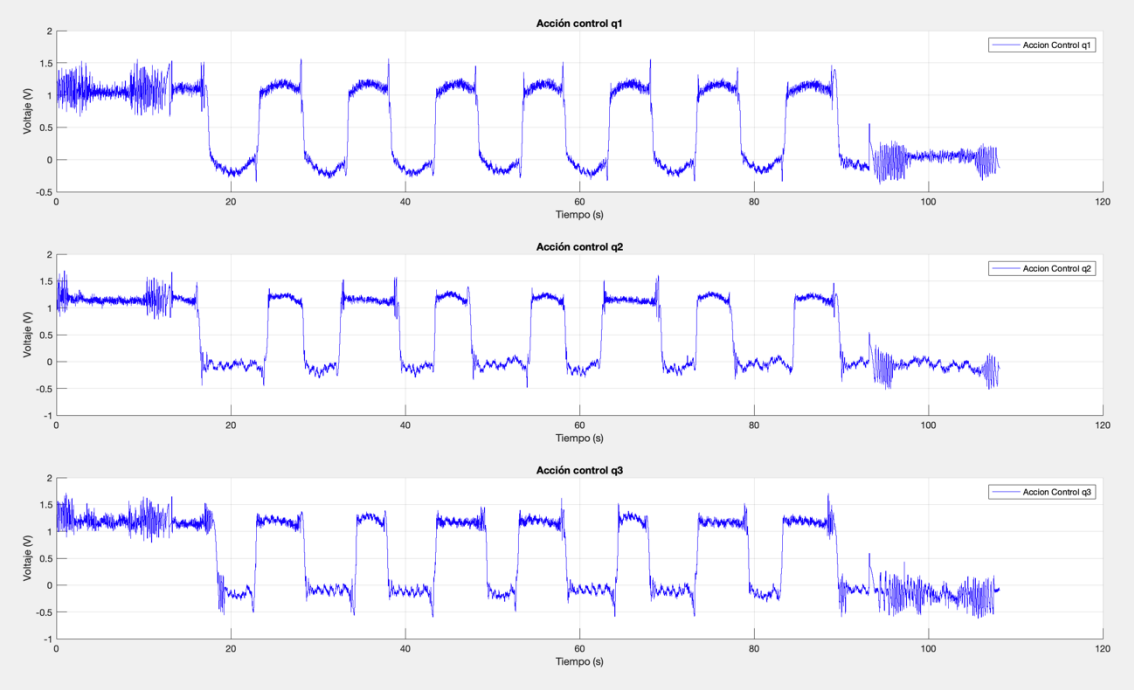


Figura 102: Experimento controlador PD+G para 3DOF, acción de control

6.2.2. CONTROL DE FUERZA

6.2.2.1. CONTROLADOR VERSIÓN 1

Se han desarrollado diversos experimentos con la primera versión del controlador de fuerza. En todos estos, se repite la misma trayectoria de posición. En primer lugar, el robot asciende realizando una curva Spline desde la posición en reposo hasta una altura determinada en la que se comienza a controlar la fuerza. Antes de este momento, el control de fuerza está desactivado. En el momento que se controla la fuerza, ya se puede mover el robot a través de la dinámica indicada por el modelo de admitancia durante 60 segundos. Posteriormente, se vuelve a desactivar el control de fuerza, pero esta vez forzando el error de fuerza a 0. Esto se ha hecho así para que el robot vuelva de una manera suave y controlada a la trayectoria original. En otras palabras, si el robot en ese momento se encuentra en una posición diferente a la referencia original, la respuesta del modelo de admitancia frente a un error de fuerza 0 será la encargada de que el robot tienda a desplazarse hacia la posición original de referencia.

Todos los experimentos que se van a mostrar a continuación se han realizado con una persona que ha colocado su pie en el arnés.

Figura 103: Desarrollo del experimento de fuerza

De esta manera, si se mueve el pie, esa fuerza es detectada por el sensor y el robot la compensa. En el siguiente experimento, se ha introducido una referencia de fuerza 0. Lo que se espera con esto es que el paciente pueda mover libremente la plataforma.

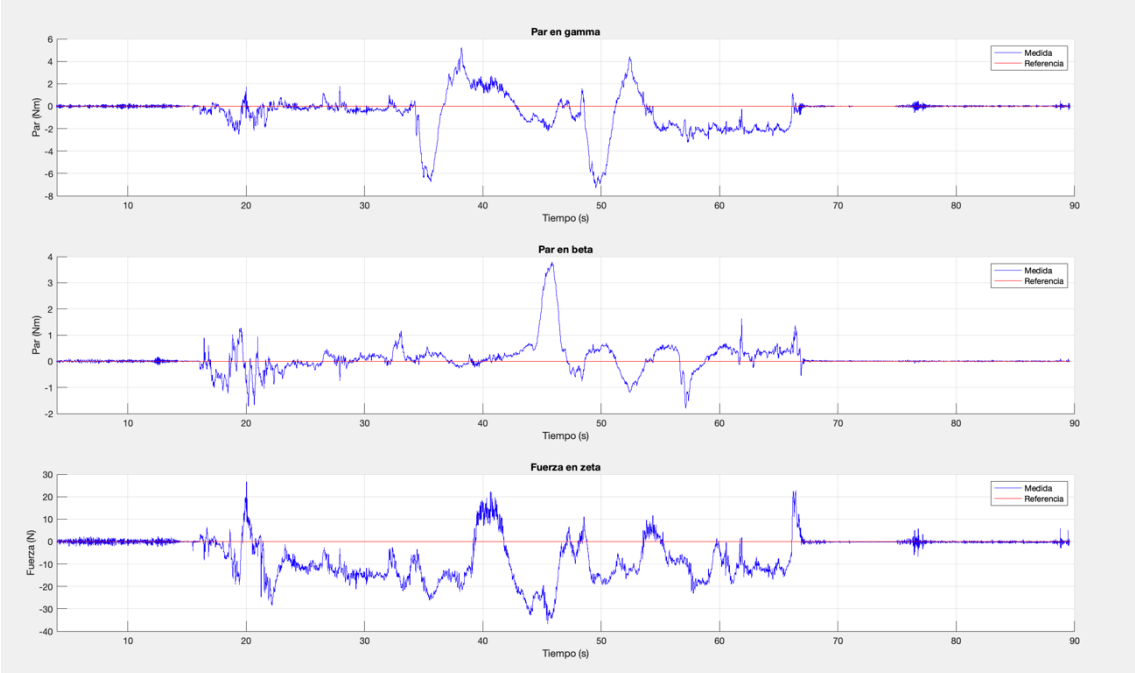


Figura 104: Experimento controlador de fuerza versión 1, referencia de fuerza 0, fuerza medida y fuerza de referencia

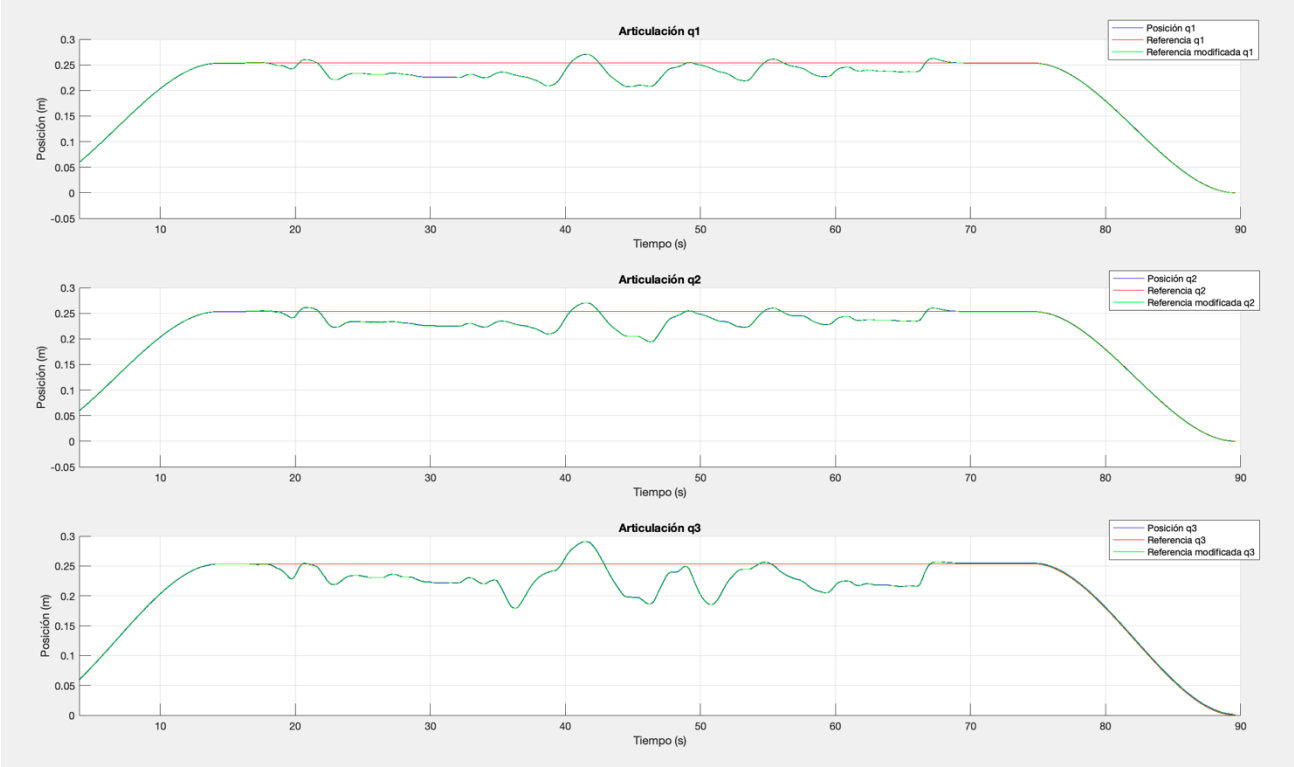


Figura 105: Experimento controlador de fuerza versión 1, referencia de fuerza 0, posición

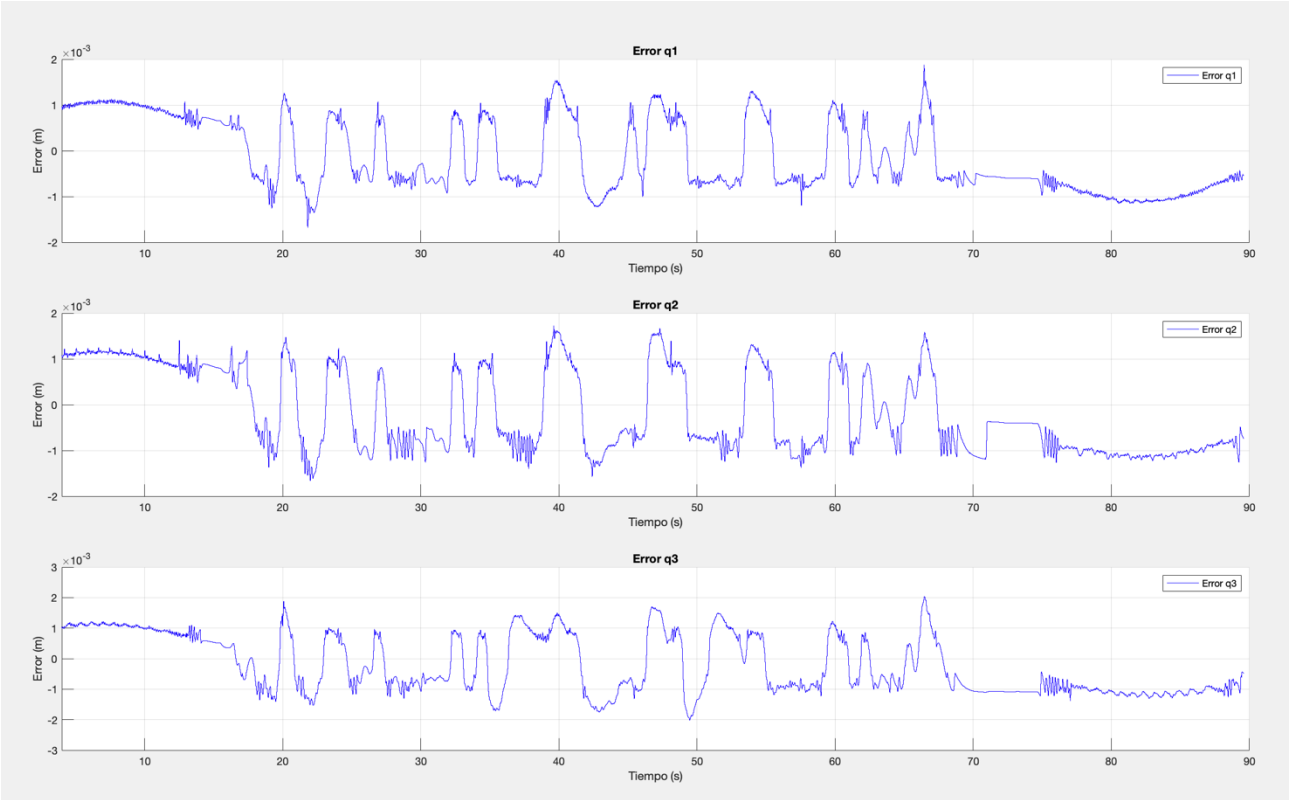


Figura 106: Experimento controlador de fuerza versión 1, referencia de fuerza 0, error de posición

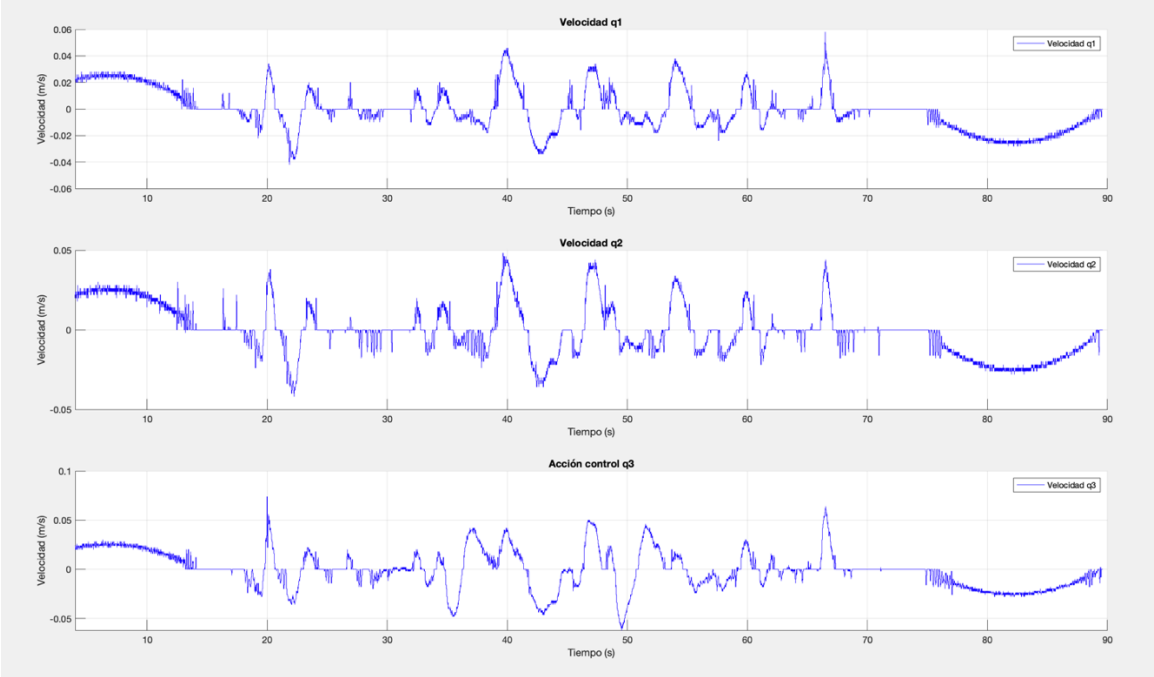


Figura 107: Experimento controlador de fuerza versión 1, referencia de fuerza 0, velocidad

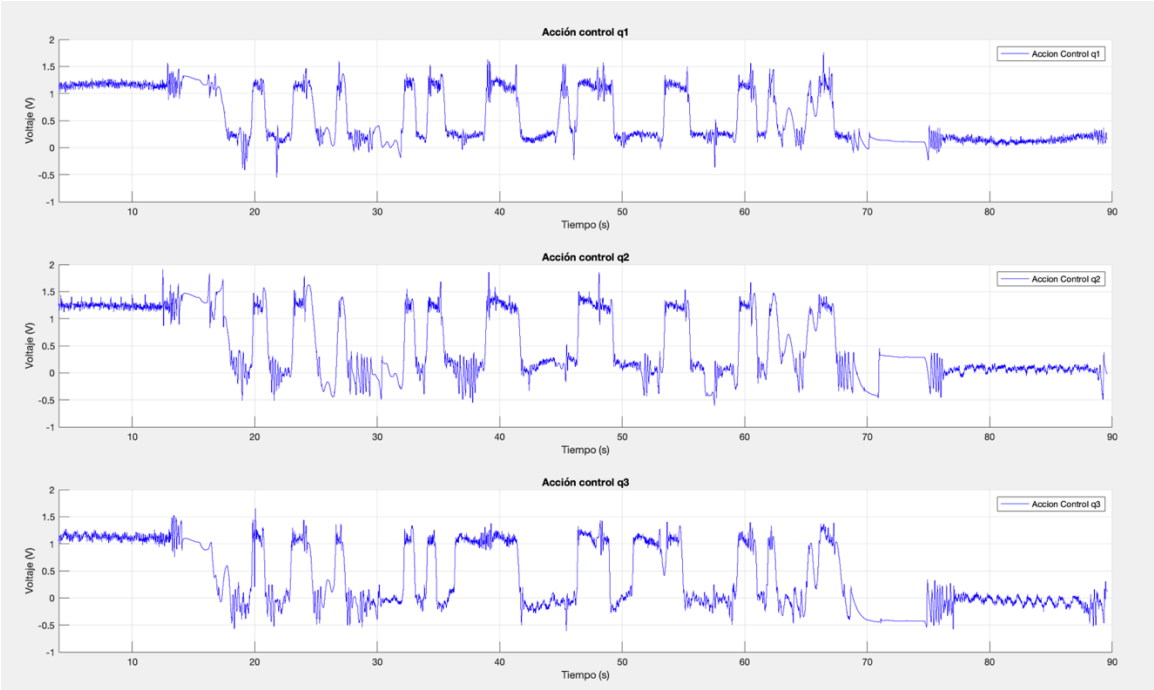


Figura 108: Experimento controlador de fuerza versión 1, referencia de fuerza 0, acción de control

En el segundo experimento, se ha realizado la misma trayectoria de posición, pero aplicando una referencia de senoidal de fuerza en el eje z. Esto es útil para practicar ejercicios de rehabilitación resistivos en los que el paciente ha de ejercer una fuerza determinada. El resultado es el siguiente:

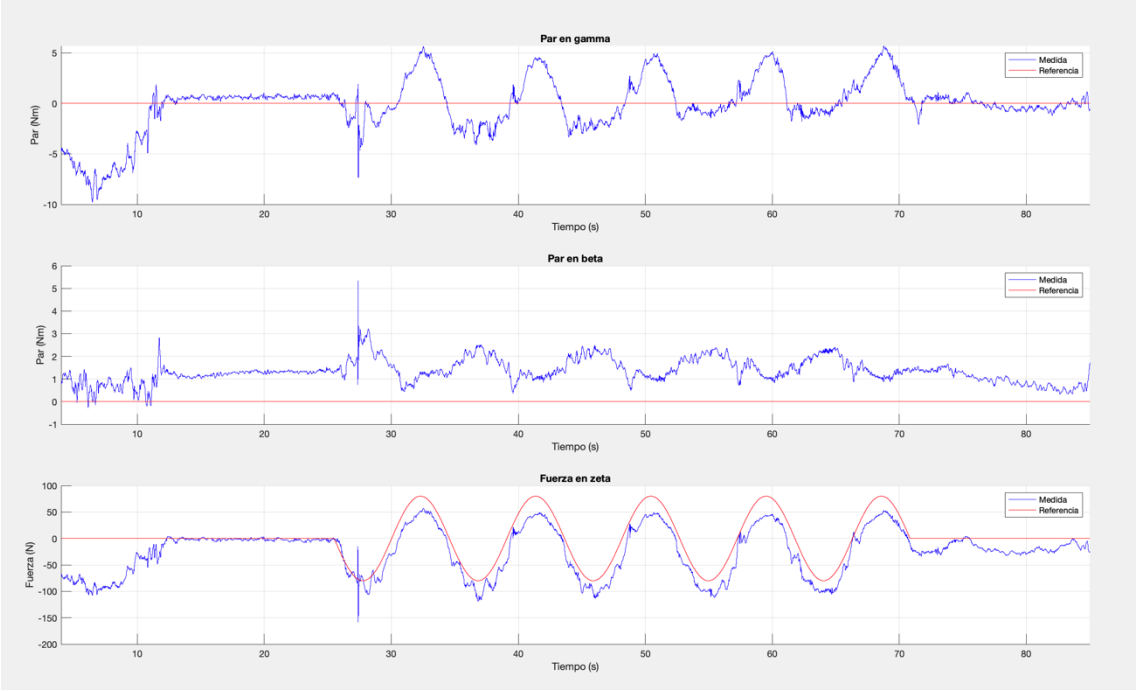


Figura 109: Experimento controlador de fuerza versión 1, referencia de fuerza senoidal en el eje z, fuerza medida y fuerza de referencia

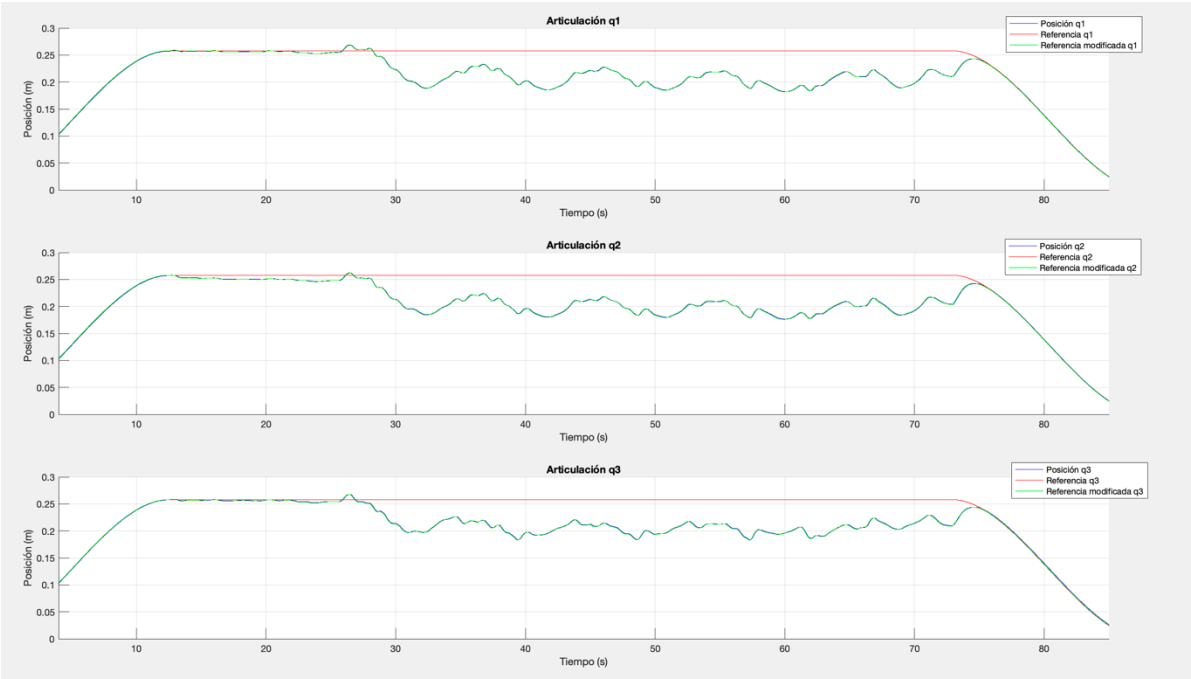


Figura 110: Experimento controlador de fuerza versión 1, referencia de fuerza senoidal en el eje z, posición

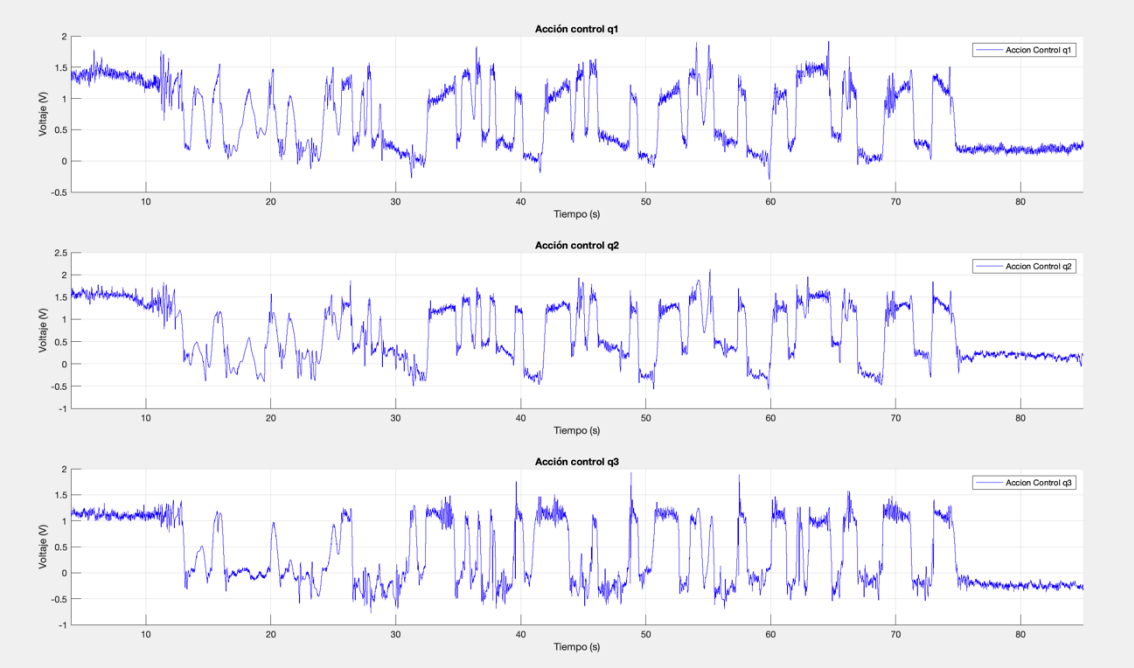


Figura 111: Experimento controlador de fuerza versión 1, referencia de fuerza senoidal en el eje z, acción de control

En el tercer experimento, se ha aplicado una referencia de par senoidal al eje gamma:

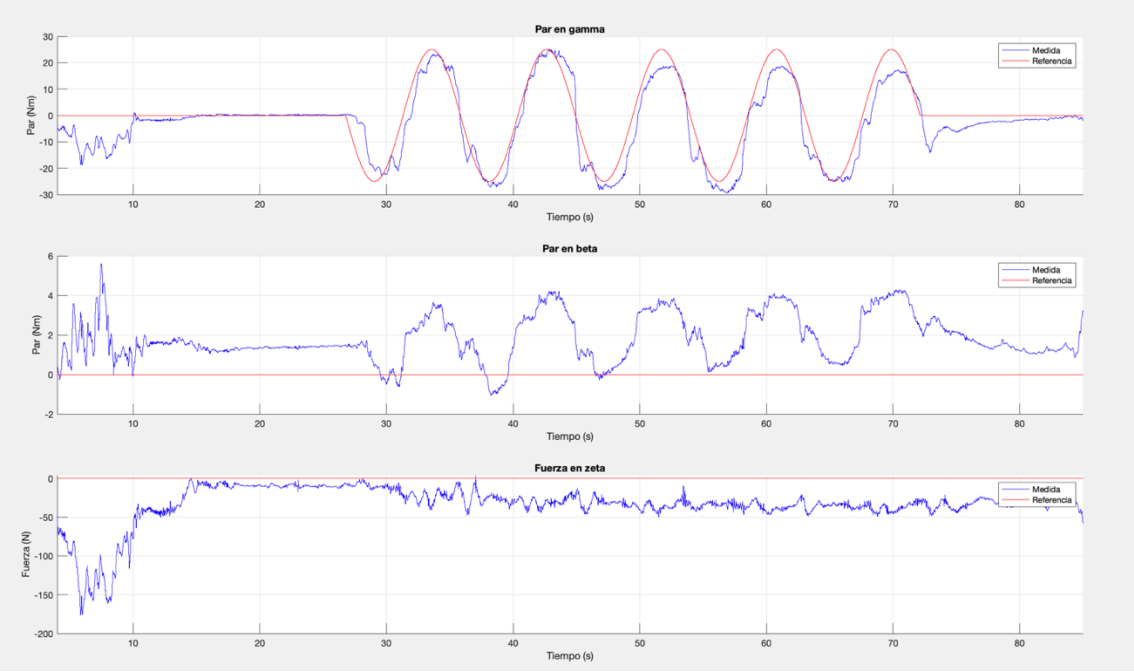


Figura 112: Experimento controlador de fuerza versión 1, referencia de par senoidal en el eje gamma, fuerza medida y fuerza de referencia

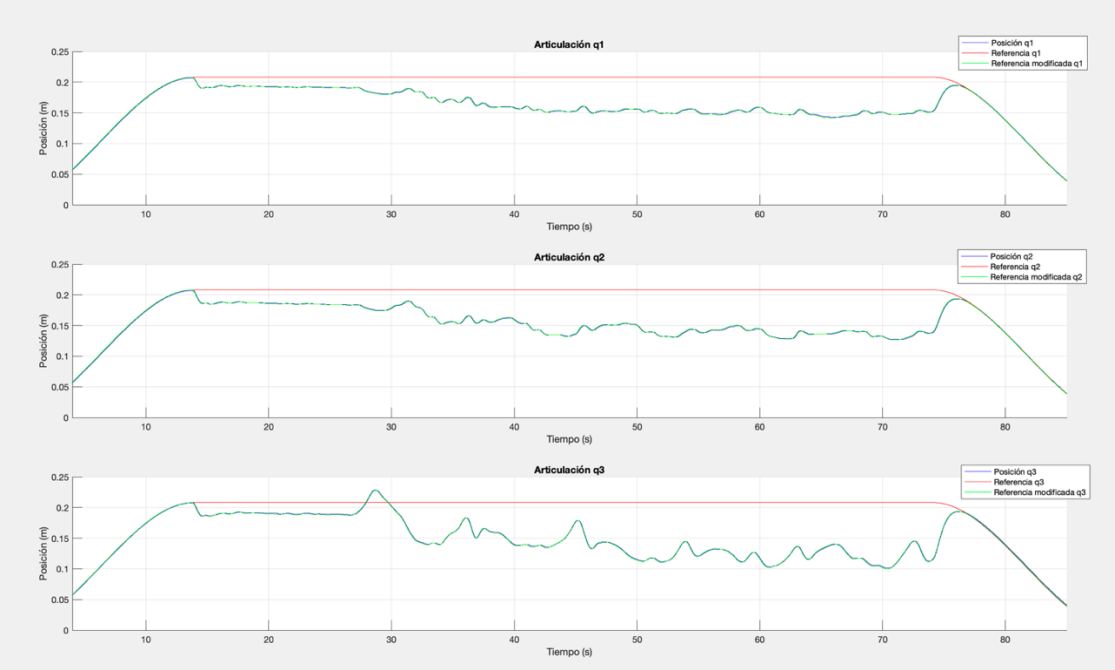


Figura 113: Experimento controlador de fuerza versión 1, referencia de par senoidal en el eje gamma, posición

En la gráfica anterior, en el segundo 75 es el momento en el que se deja de controlar la fuerza, pero el modelo de admitancia sigue emitiendo velocidades con una respuesta frente a un error forzado de fuerza 0. Se puede observar que el robot se encuentra en una posición distinta a la referencia original, pero se traslada a ella de una manera suave.

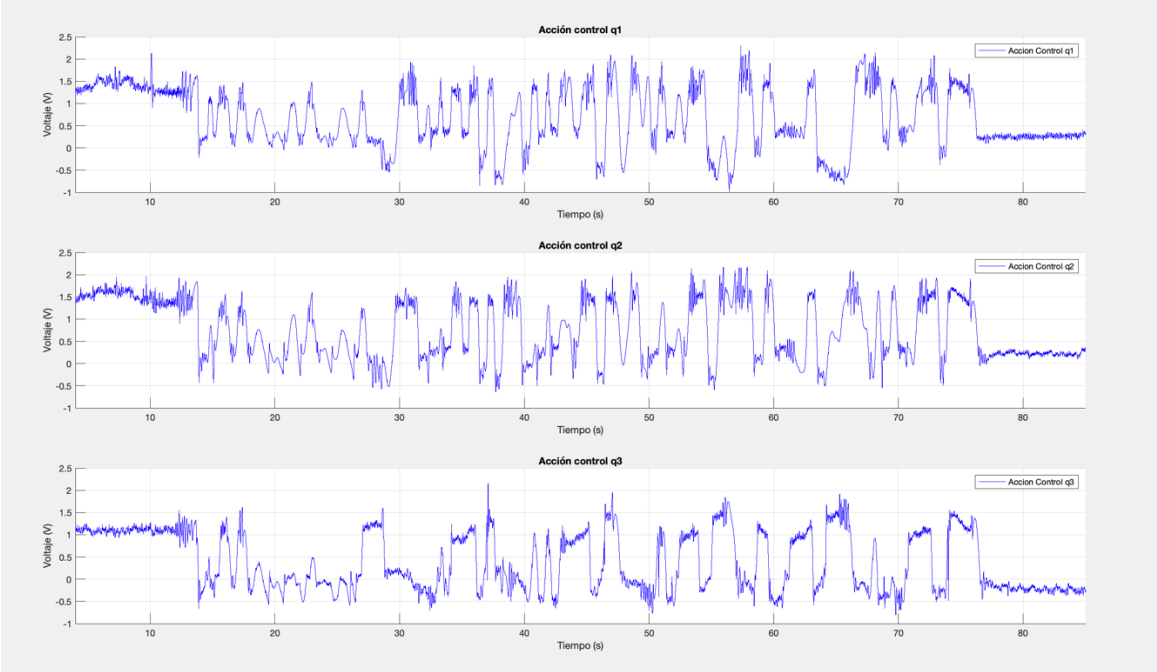


Figura 114: Experimento controlador de fuerza versión 1, referencia de par senoidal en el eje gamma, acción de control

Por último, también se ha aplicado una referencia de par senoidal al eje beta:

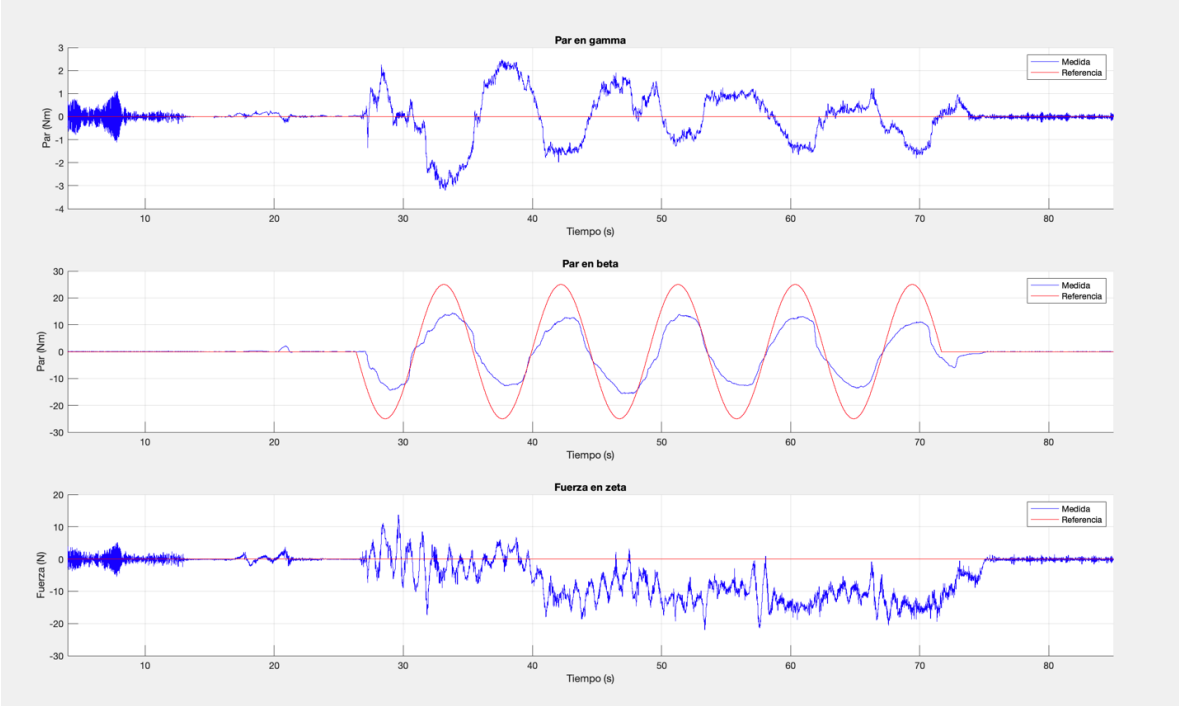


Figura 115: Experimento controlador de fuerza versión 1, referencia de par senoidal en el eje beta, fuerza medida y fuerza de referencia

El seguimiento de la fuerza en el eje beta es más difícil puesto que es más complicado el movimiento del tobillo en este eje y también porque el arnés tiene una cierta holgura y no está completamente sujeto el pie.

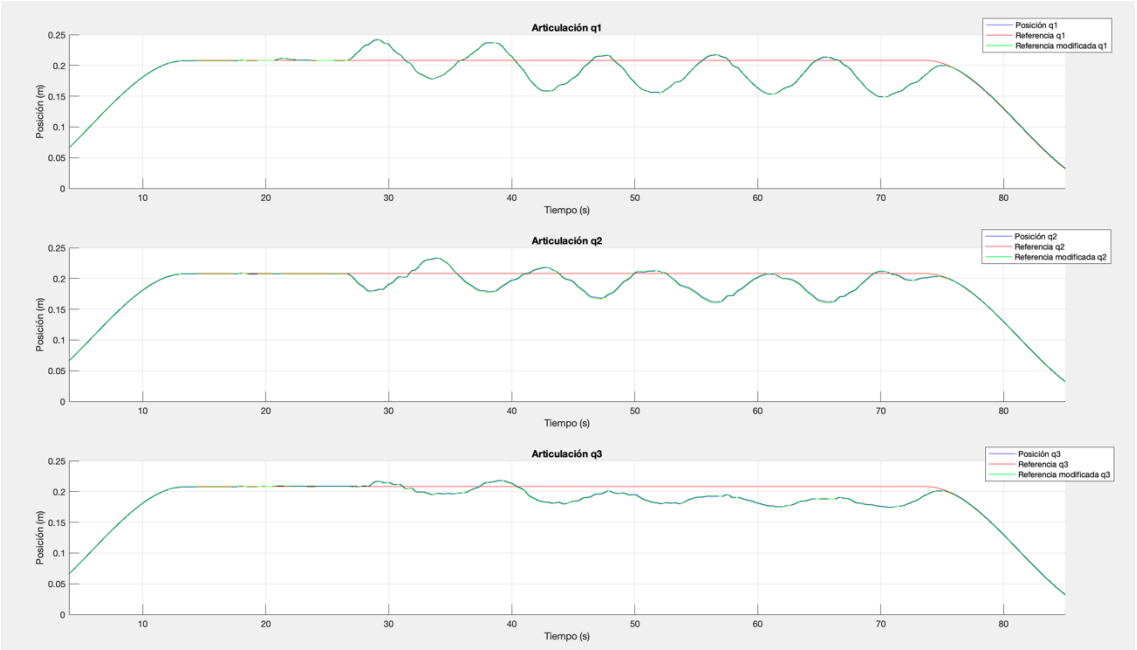


Figura 116: Experimento controlador de fuerza versión 1, referencia de par senoidal en el eje beta, posición

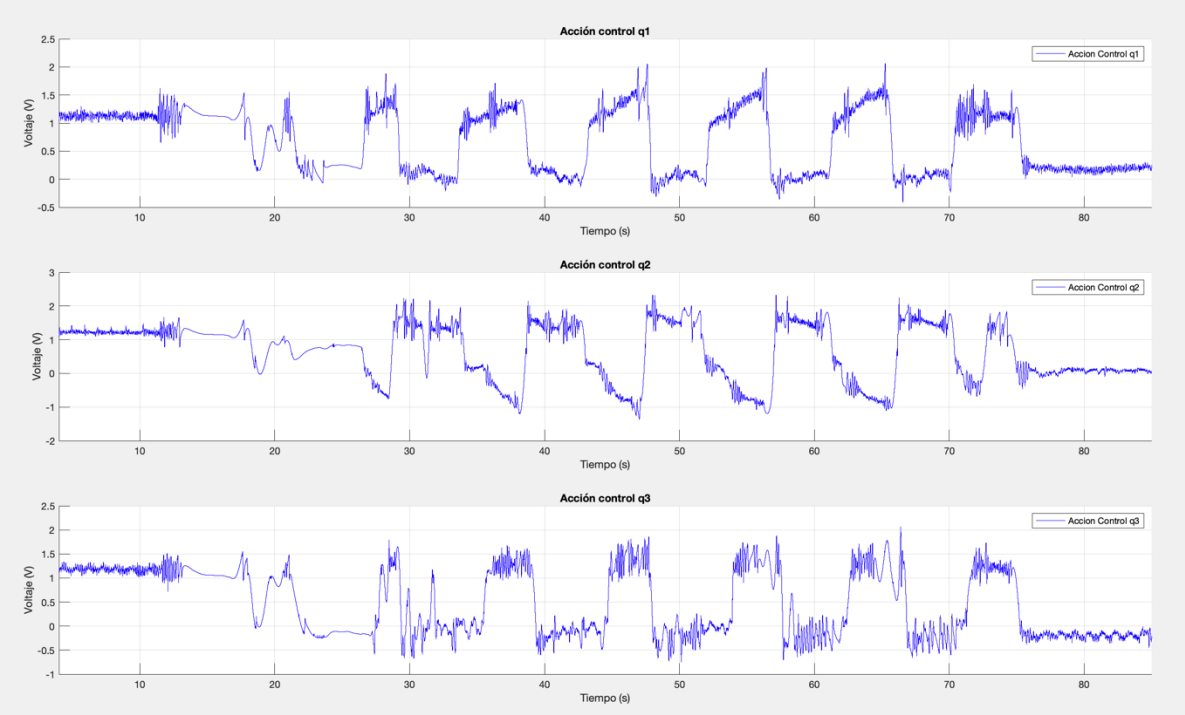


Figura 117: Experimento controlador de fuerza versión 1, referencia de par senoidal en el eje beta, acción de control

6.2.2.2. CONTROLADOR VERSIÓN 2

Se ha validado la versión 2 del controlador de fuerza basado en el modelo de admitancia experimentalmente con el robot real de 3 grados de libertad. El experimento desarrollado ha sido el mismo que el explicado en el apartado anterior. Se ha elevado el robot una altura y después se ha activado el control de fuerza. La referencia de fuerza esta vez ha sido de 0 en los tres ejes y se ha comprobado que al presionar la bota el robot responde y compensa esta fuerza. Después de un cierto tiempo, como sucedía anteriormente, se desactiva el control de fuerza, forzando a que el error de fuerza sea 0 para que el robot vuelva a la posición de la referencia de posición original siguiendo la dinámica indicada por los parámetros del modelo de la admitancia y que desciendia siguiendo una curva Spline hasta la posición de reposo. El resultado se puede ver a continuación:

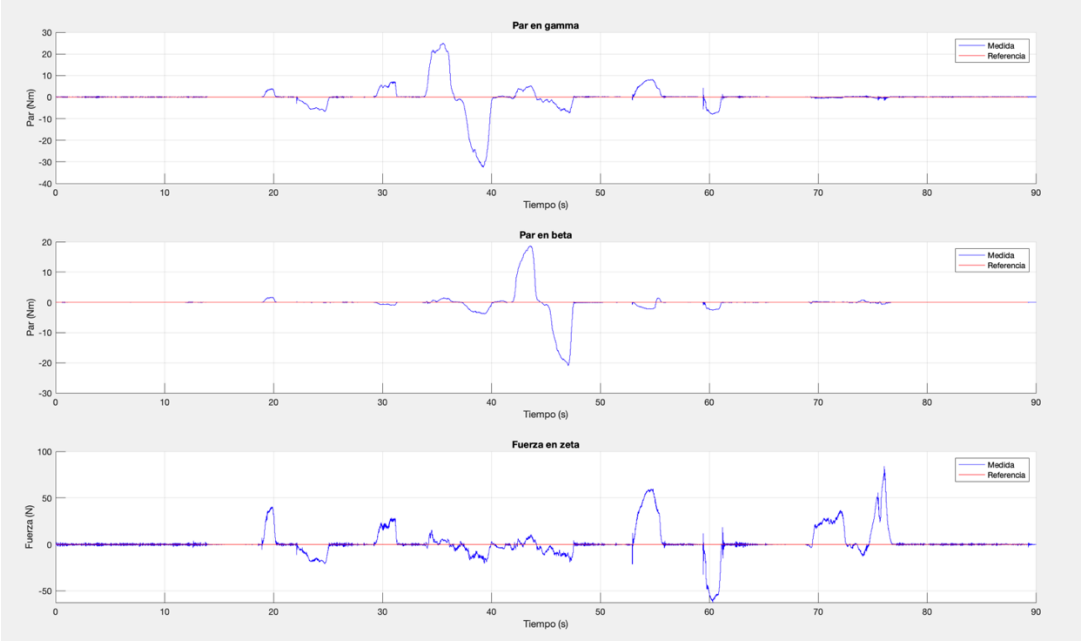


Figura 118: Experimento controlador de fuerza versión 2, referencia de fuerza 0, fuerza medida y fuerza de referencia

Se puede observar que, al detectarse una fuerza, esta después tiende a 0 gracias al movimiento de la plataforma del robot que compensa la fuerza o par detectado en el sensor. En la siguiente figura se muestra la referencia de posición en el espacio articular modificada y la posición real de las articulaciones:

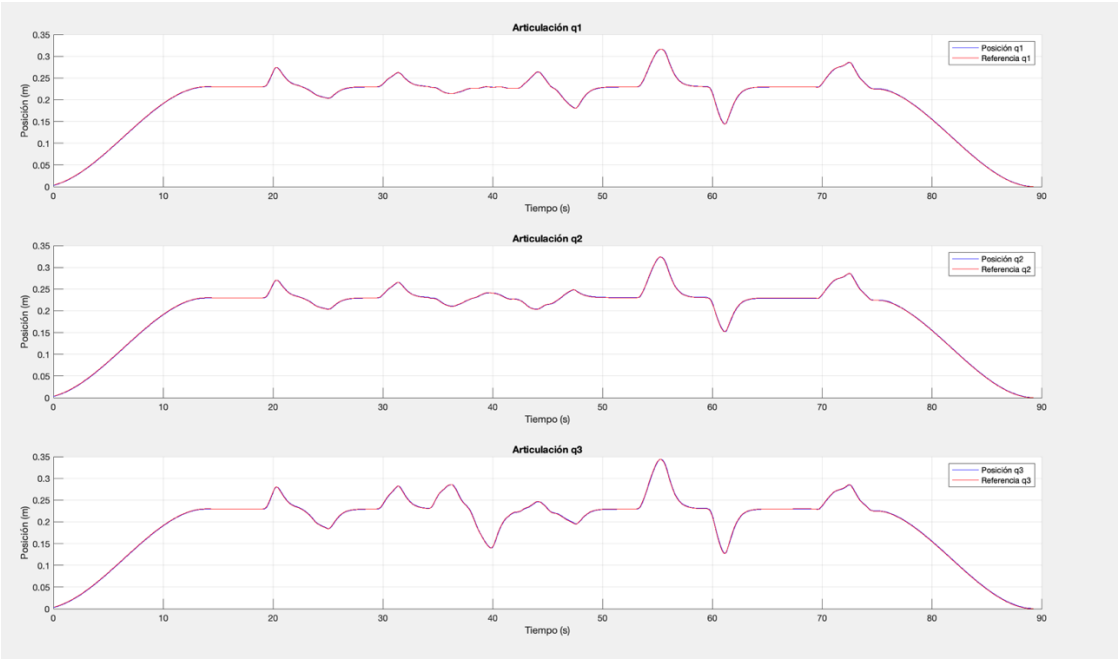


Figura 119: Experimento controlador de fuerza versión 2, referencia de fuerza 0, posición

6.2.3. APRENDIZAJE POR REFUERZO

En primer lugar, con el fin de comprobar si los algoritmos entrenados han aprendido a disminuir el error de posición, en las siguientes imágenes se muestra la respuesta del controlador PD que se ha utilizado junto con los algoritmos de aprendizaje por refuerzo. En estos experimentos entonces, la acción de control de los agentes de aprendizaje es nula.

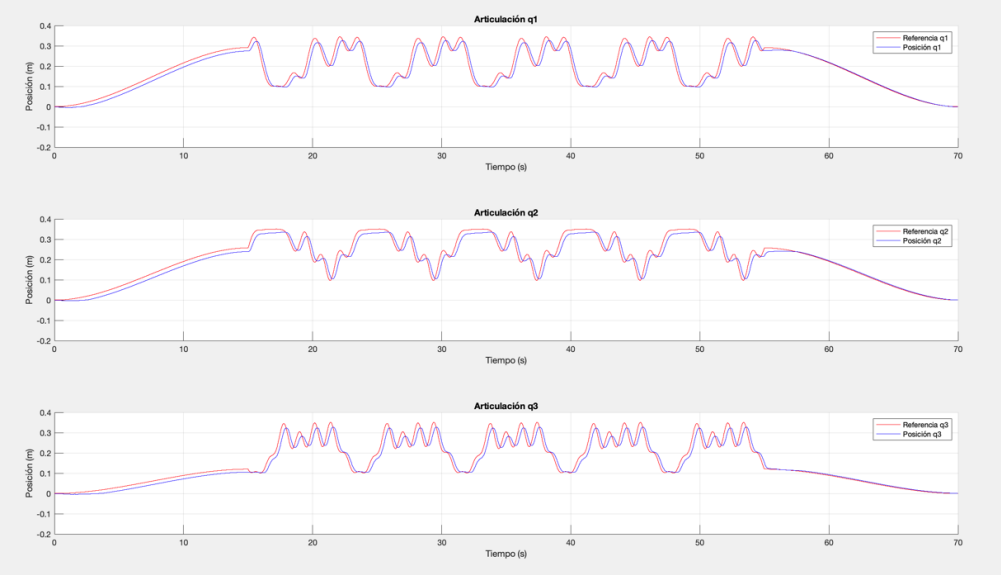


Figura 120: Experimento del controlador PD que se ha utilizado junto con los algoritmos de aprendizaje por refuerzo. Posición y referencia.

Se puede apreciar que el controlador tiene un error de posición considerable, que alcanza los 5 centímetros. Para apreciarlo mejor, se ha mostrado en una gráfica también:

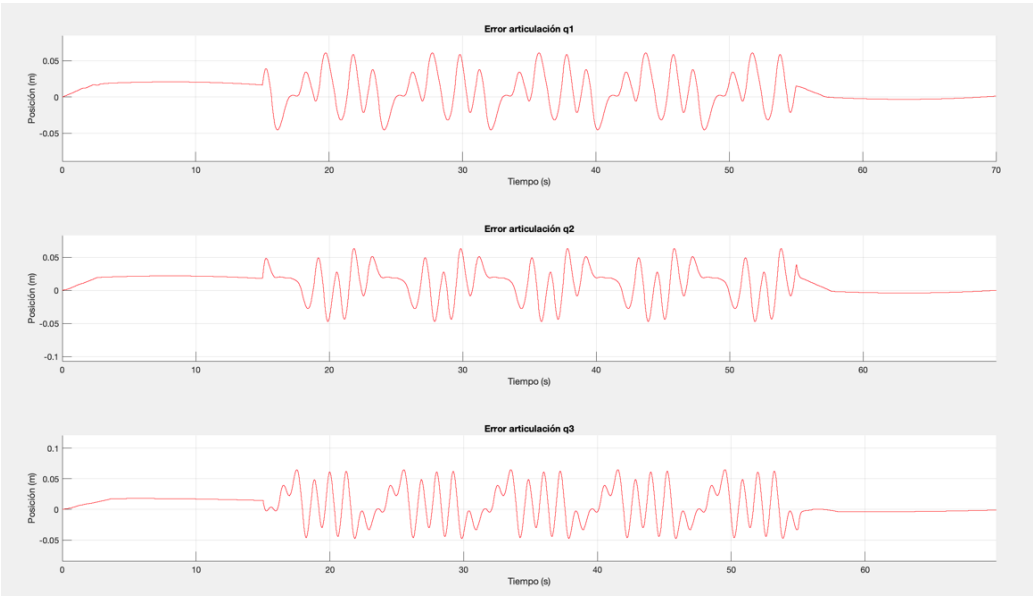


Figura 121: Experimento del controlador PD que se ha utilizado junto con los algoritmos de aprendizaje por refuerzo. Error de posición.

Para evaluar estos algoritmos, no solamente es importante el error de posición, si no también las acciones de control que emiten. Una ventaja del PD con respecto a los algoritmos de aprendizaje por refuerzo es que es más sencillo que el PD proporcione acciones de control más suaves:

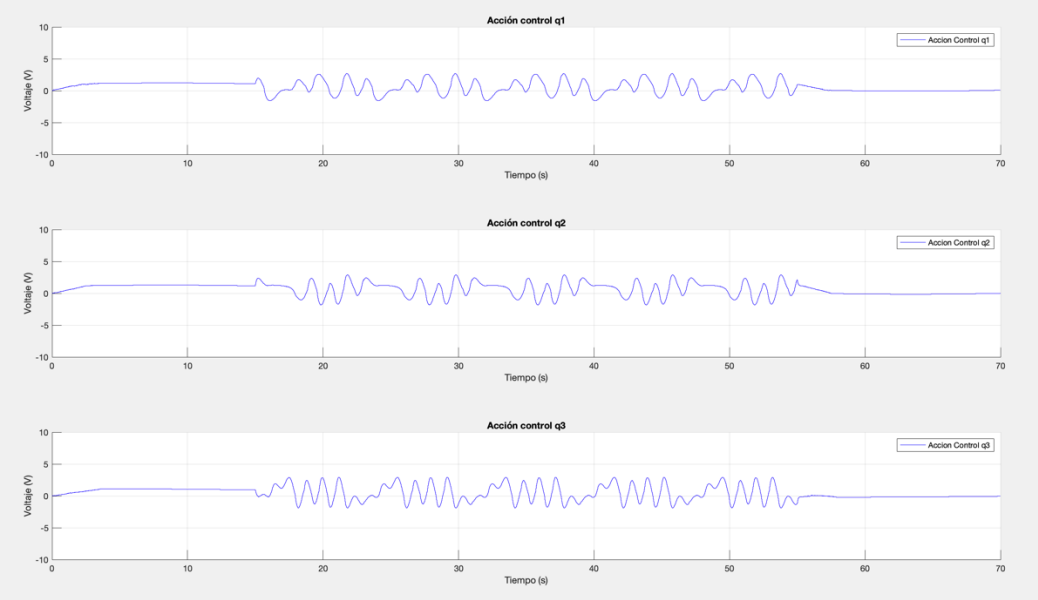


Figura 122: Experimento del controlador PD que se ha utilizado junto con los algoritmos de aprendizaje por refuerzo. Acción de control.

A continuación, se van a mostrar experimentos realizados con el algoritmo TRPO entrenado con la misma trayectoria. En la siguiente simulación se muestra una trayectoria en la que durante el entrenamiento solamente se ha penalizado el error de posición:

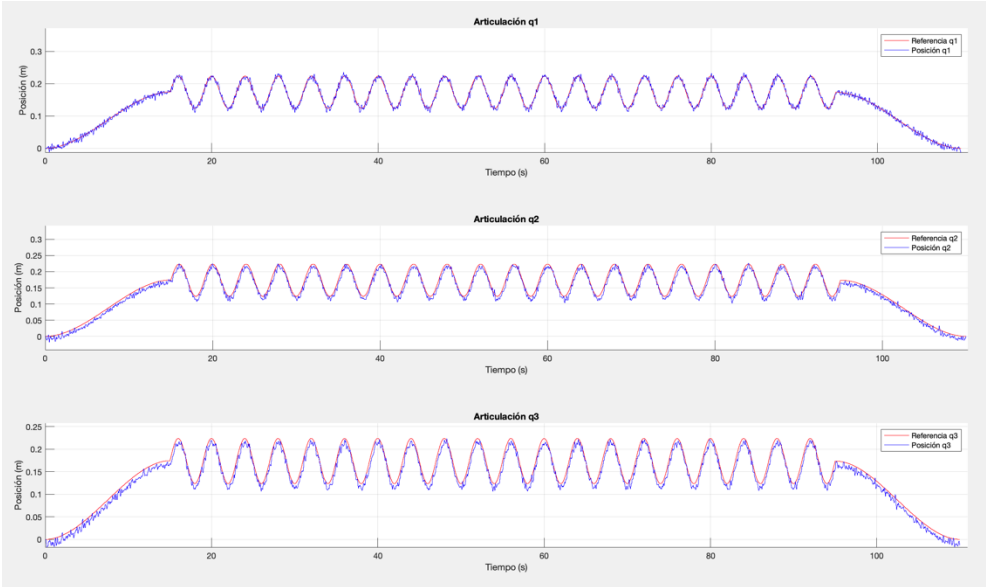


Figura 123: Experimento del agente TRPO+PD, penalizando solamente el error de posición. Posición de las articulacionesy referencia.

Se puede observar que la respuesta es muy oscilatoria. La causa de esto son las altas acciones de control y altos incrementos que proporciona el algoritmo:

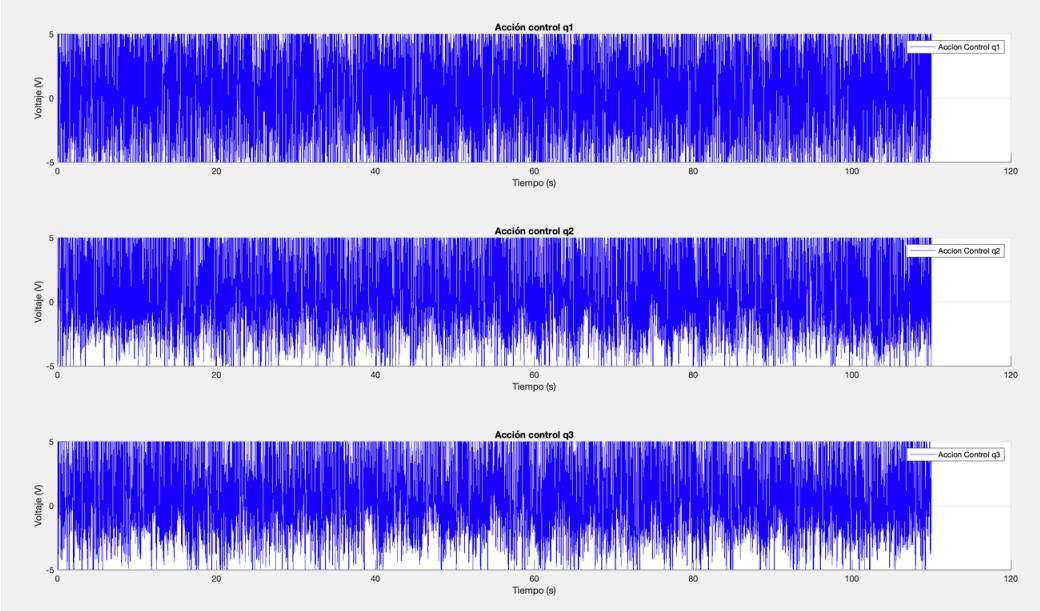


Figura 124: Experimento del agente TRPO+PD, penalizando solamente el error de posición. Posición y referencia.

Estas acciones de control no son físicamente realizables por actuadores reales, por este motivo, es necesario también penalizar o bien los incrementos de la acción de control o el error de velocidad. En las siguientes imágenes, se muestra la respuesta del algoritmo TRPO+PD entrenado durante 900e3 iteraciones, pero esta vez penalizando también el incremento de la acción de control en la señal de recompensa:

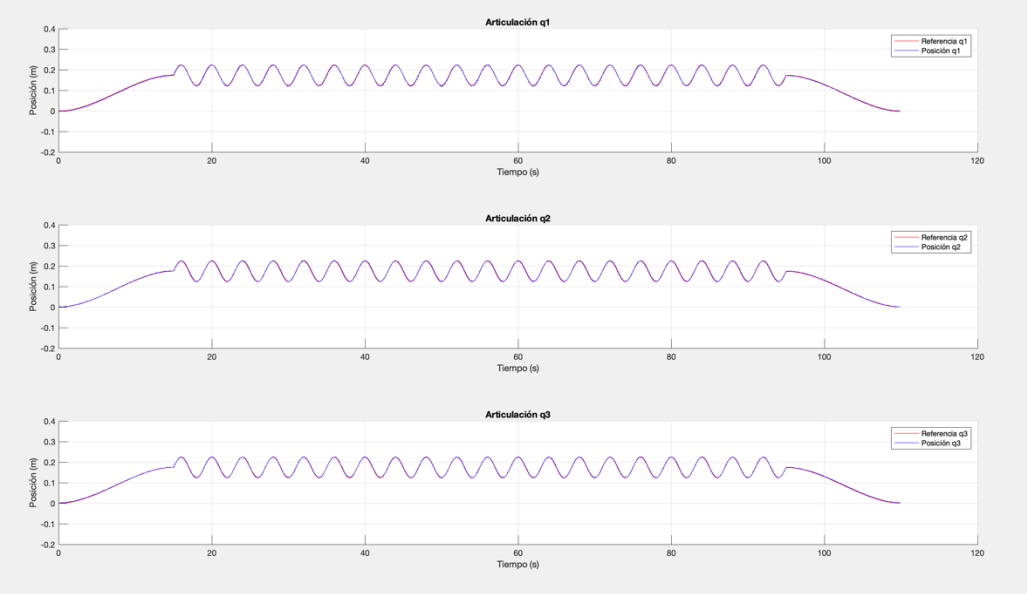


Figura 125: Experimento del agente TRPO+PD, penalizando solamente el error de posición y el incremento de la acción de control. Posición de las articulaciones y referencia.

Ahora la posición no oscila tanto y la acción de control no satura tanto como en el caso anterior:

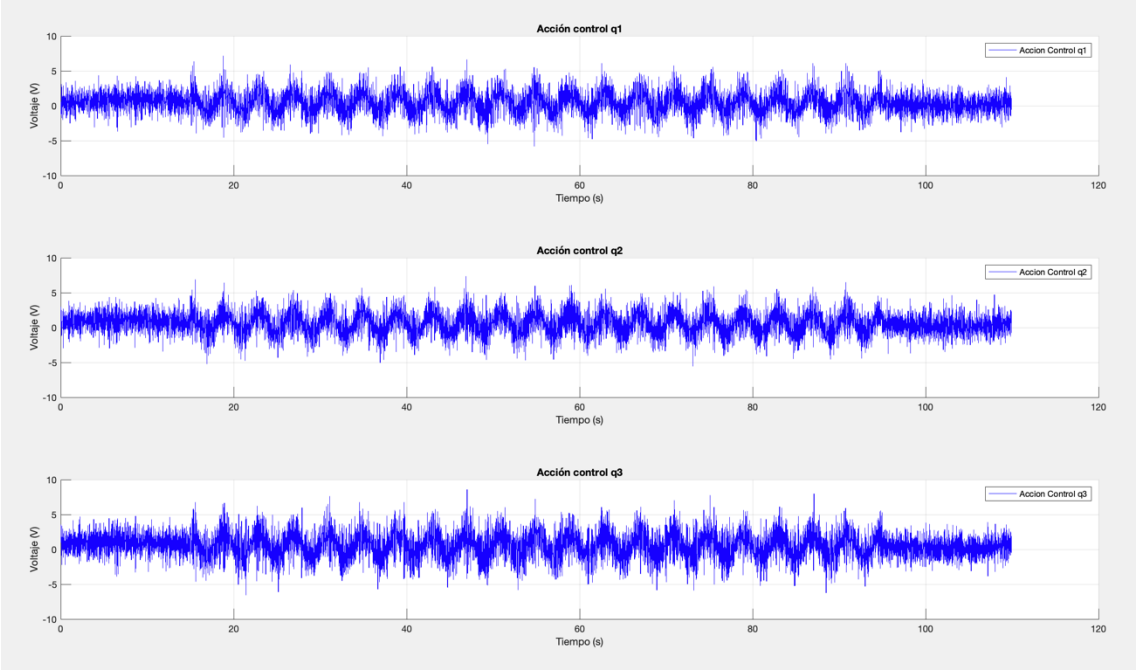


Figura 126: Experimento del agente TRPO+PD, penalizando solamente el error de posición y el incremento de la acción de control. Acción de control.

En cuanto al error de posición, en este caso no supera los 5 milímetros.

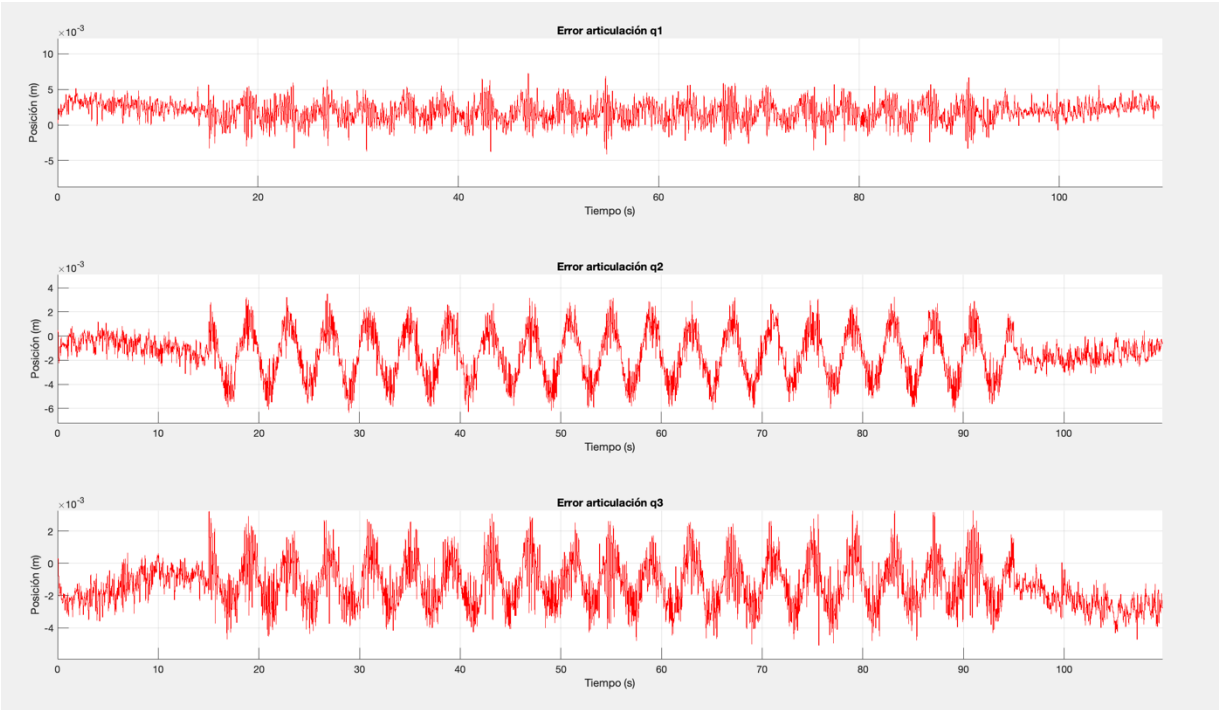


Figura 127: Experimento del agente TRPO+PD, penalizando solamente el error de posición y el incremento de la acción de control. Error de posición.

Las simulaciones que se van a mostrar a continuación corresponden con el algoritmo PPO junto con el controlador PD. Con el controlador PPO se han obtenido resultados mejores que con TRPO, su aprendizaje es más estable y ofrece acciones de más suaves. En el siguiente experimento, se ha entrenado al controlador PPO+PD durante 3e6 iteraciones (con 39 trayectorias diferentes) penalizando el error de posición y los incrementos de la acción de control. El resultado es el siguiente:

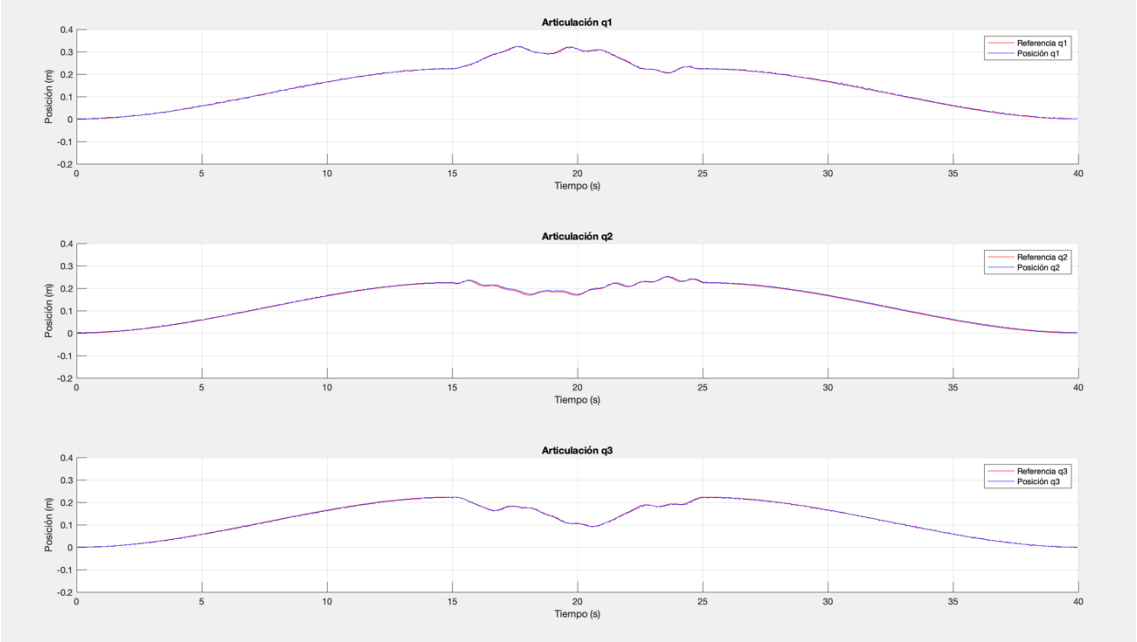


Figura 128: Experimento del agente PPO+PD. Posición de las articulaciones y de referencia.

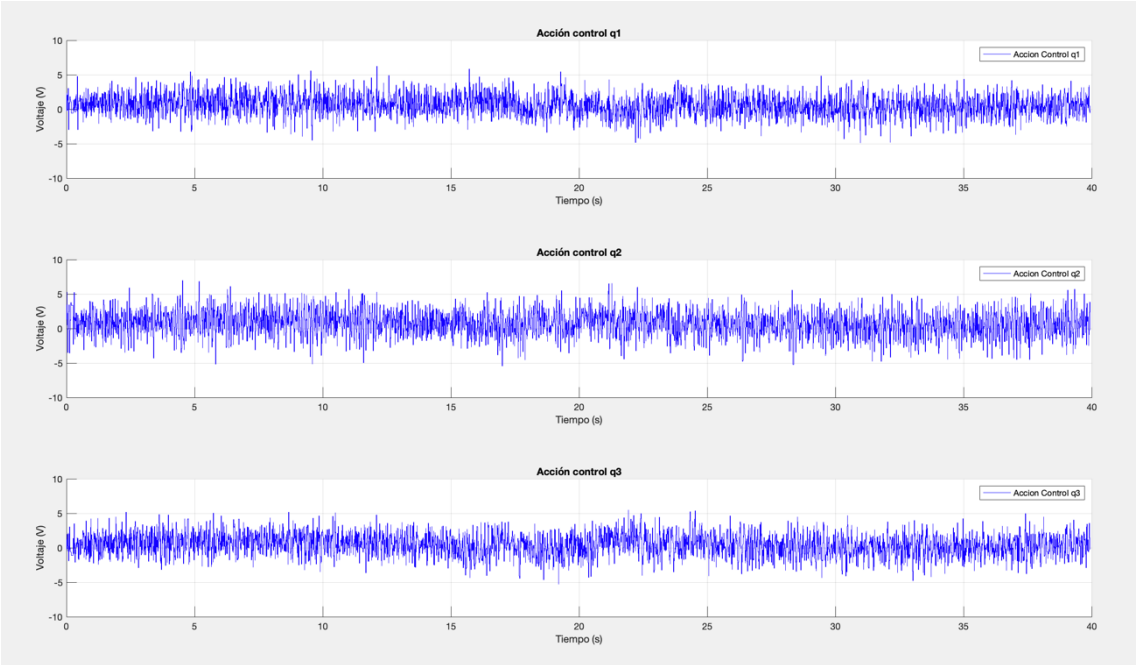


Figura 129: Experimento del agente PPO+PD. Acción de control de PPO.

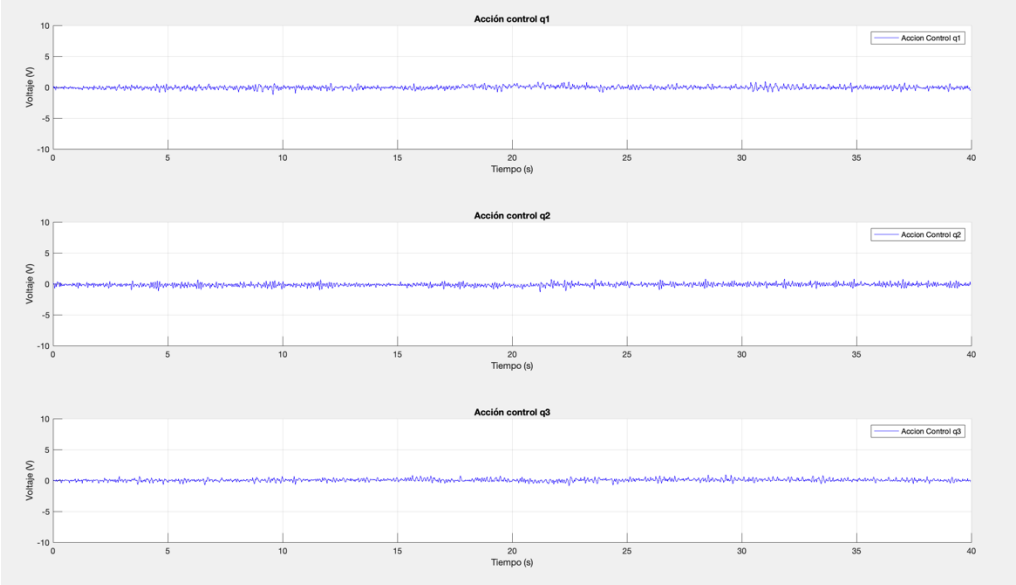


Figura 130: Experimento del agente PPO+PD. Acción de control de PD.

Se puede observar que el algoritmo PPO emite acciones muchos más elevadas que el PD. Al principio del entrenamiento, esto no era así. El algoritmo PPO emitía acciones equivocadas y el PD compensaba el error con acciones más grandes. Después de 3e6 iteraciones, el error de posición es muy pequeño y por lo tanto las acciones del controlador PD también lo son.

A continuación, se muestra la respuesta del controlador PPO+PD frente a una trayectoria distinta.

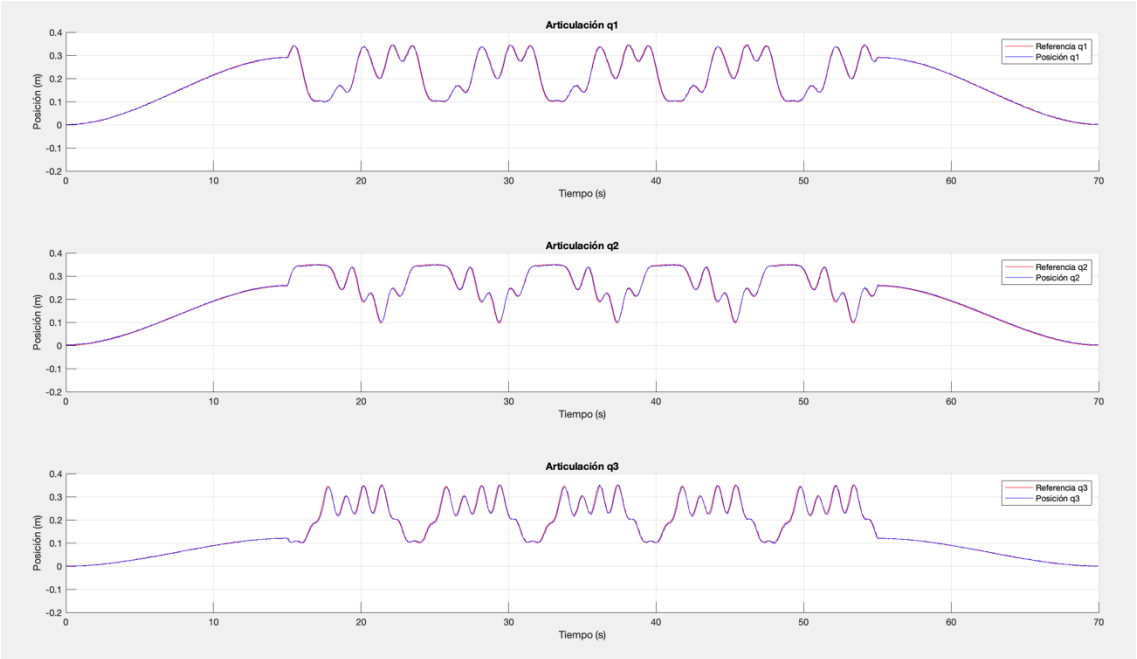


Figura 131: Experimento 2 del agente PPO+PD. Posición de las articulaciones y de referencia.

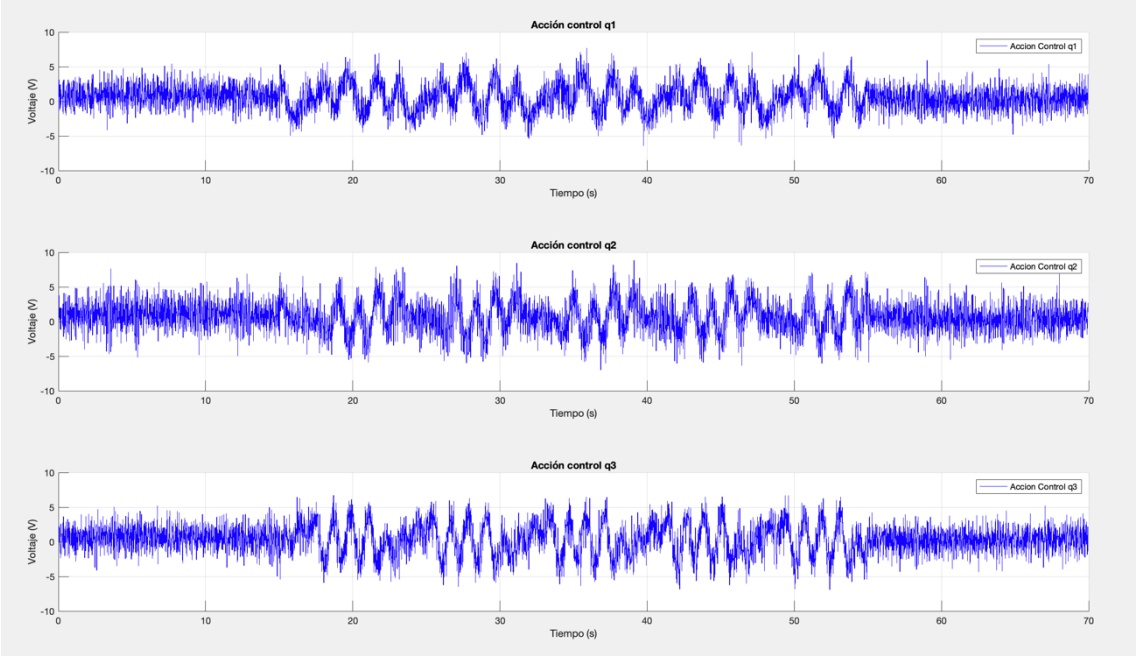


Figura 132: Experimento 2 del agente PPO+PD. Acción de control de PPO.

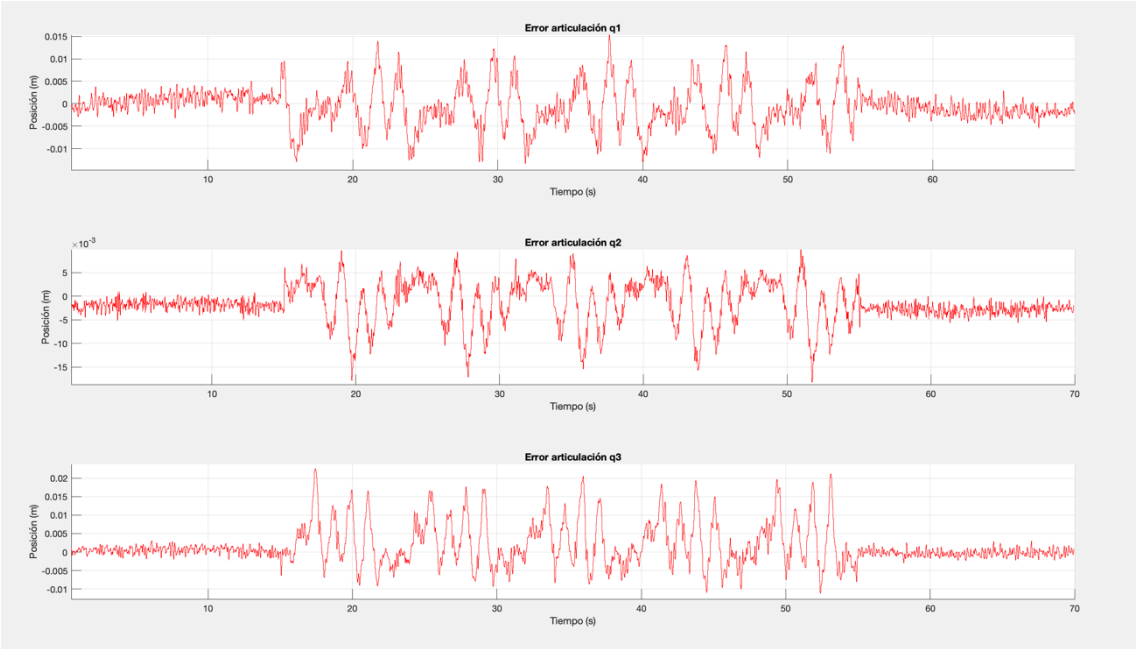


Figura 133: Experimento 2 del agente PPO+PD. Error de posición.

Esta trayectoria es precisamente la misma que se ha mostrado en el primer experimento con el controlador PD sin la acción de control un algoritmo de aprendizaje (Figs. 120, 121, 122). El error de dicho experimento alcanzaba los 5 centímetros en las tres articulaciones. Ahora este error ha disminuido y se encuentra en el orden de los milímetros.

Como se ha mencionado antes, también se ha probado a penalizar el error de velocidad en la señal de recompensa en vez de los incrementos de la acción de control.

La penalización de la velocidad ha ayudado a que las acciones de control sean más suaves. La respuesta de este experimento se trata de un controlador PPO+PD entrenado durante 400e3 iteraciones.

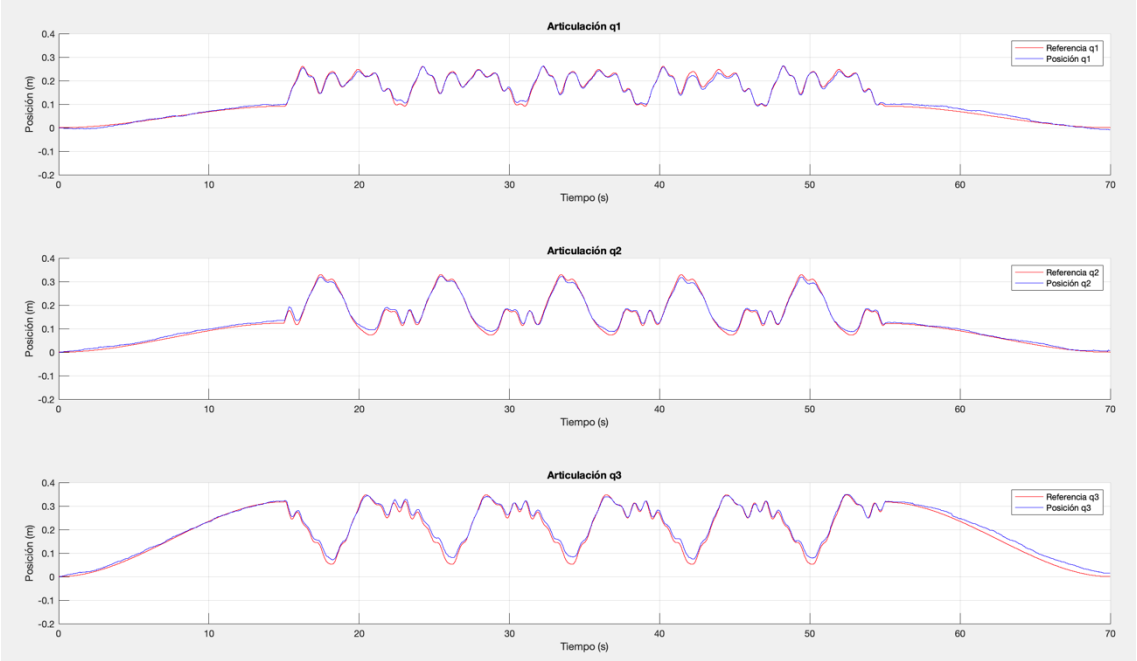


Figura 134: Experimento del agente PPO+PD penalizando la posición y el error de velocidad . Posición de las articulaciones y de referencia.

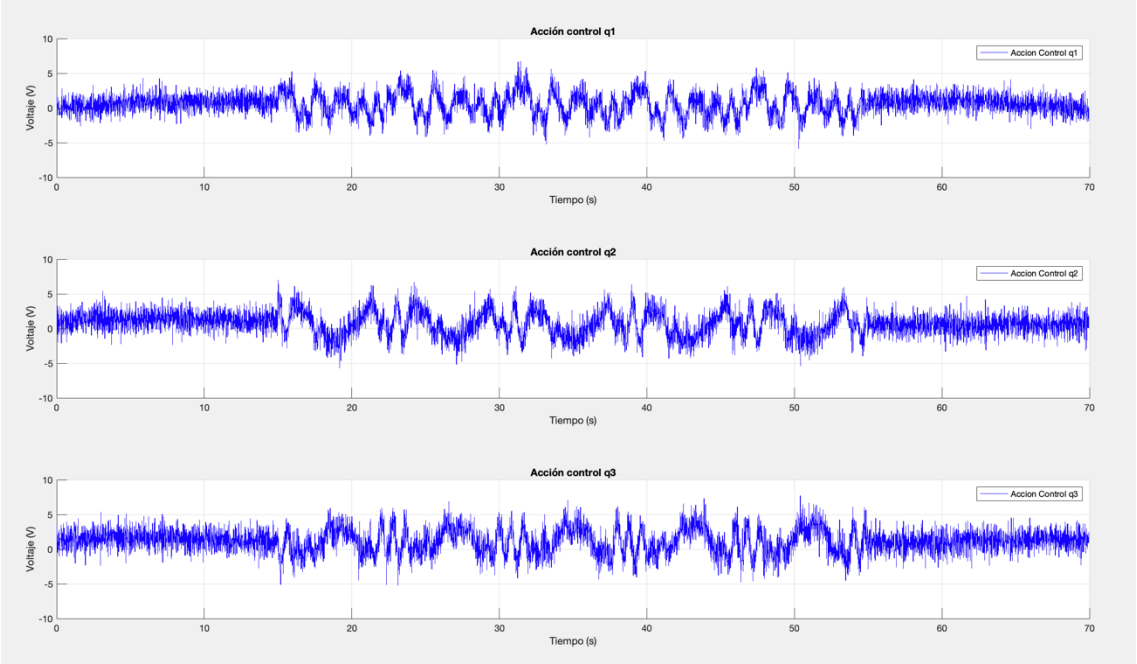


Figura 135: Experimento del agente PPO+PD penalizando la posición y el error de velocidad. Acción de control de PPO.

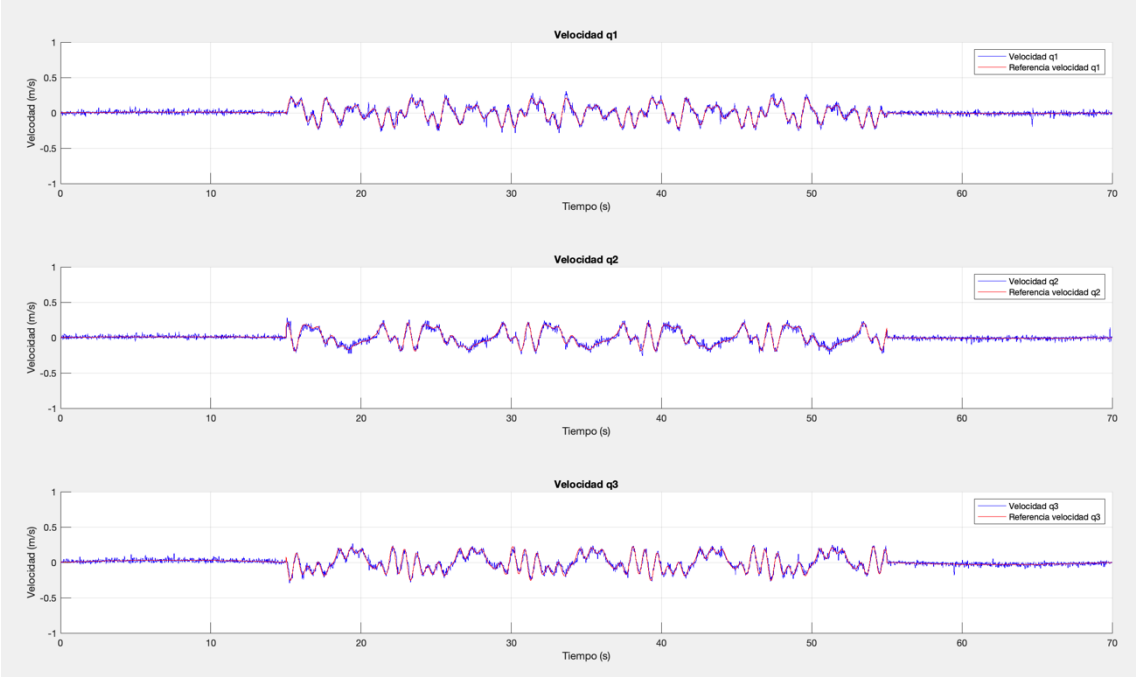


Figura 136: Experimento del agente PPO+PD penalizando la posición y el error de velocidad. Velocidad de las articulaciones y de referencia.

Se ha conseguido que los incrementos de la acción de control no sean muy elevados, sin embargo, le es más difícil a este controlador seguir la posición. En el siguiente experimento se ha penalizado más el error de posición con respecto al error de velocidad, esto ha dado lugar a un mejor seguimiento de la referencia de posición, pero a unas acciones de control más agresivas. El controlador se ha entrenado durante 1e6 iteraciones:

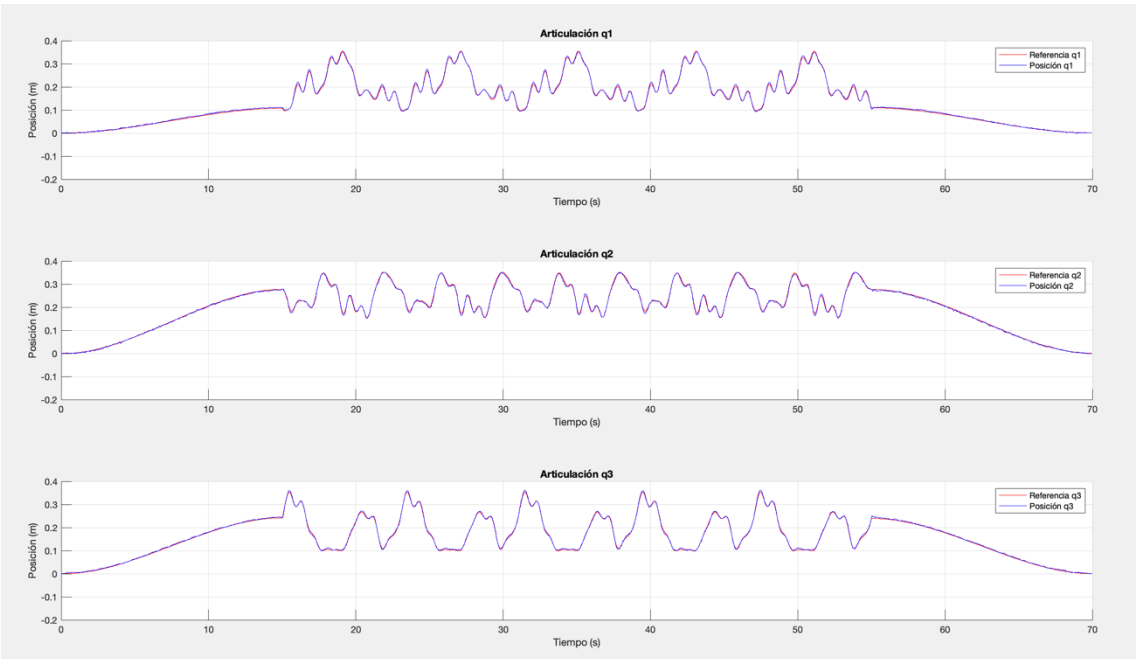


Figura 137: Experimento 2 del agente PPO+PD penalizando la posición y el error de velocidad. Posición de las articulaciones y de referencia.

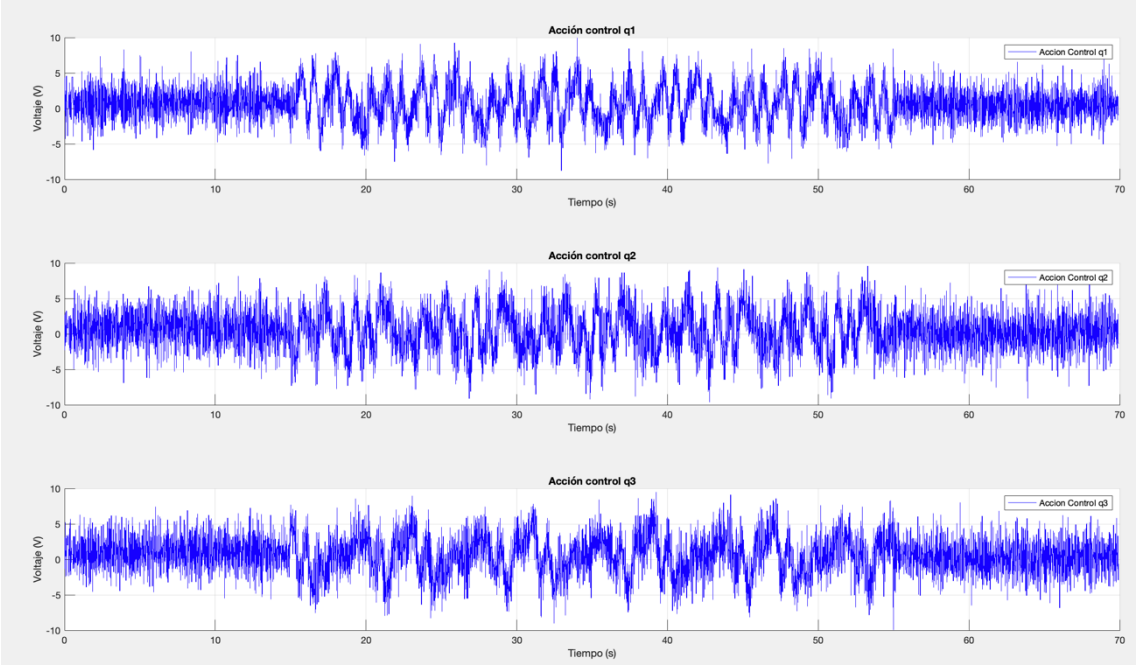


Figura 138: Experimento 2 del agente PPO+PD penalizando la posición y el error de velocidad. Acción de control de PPO.

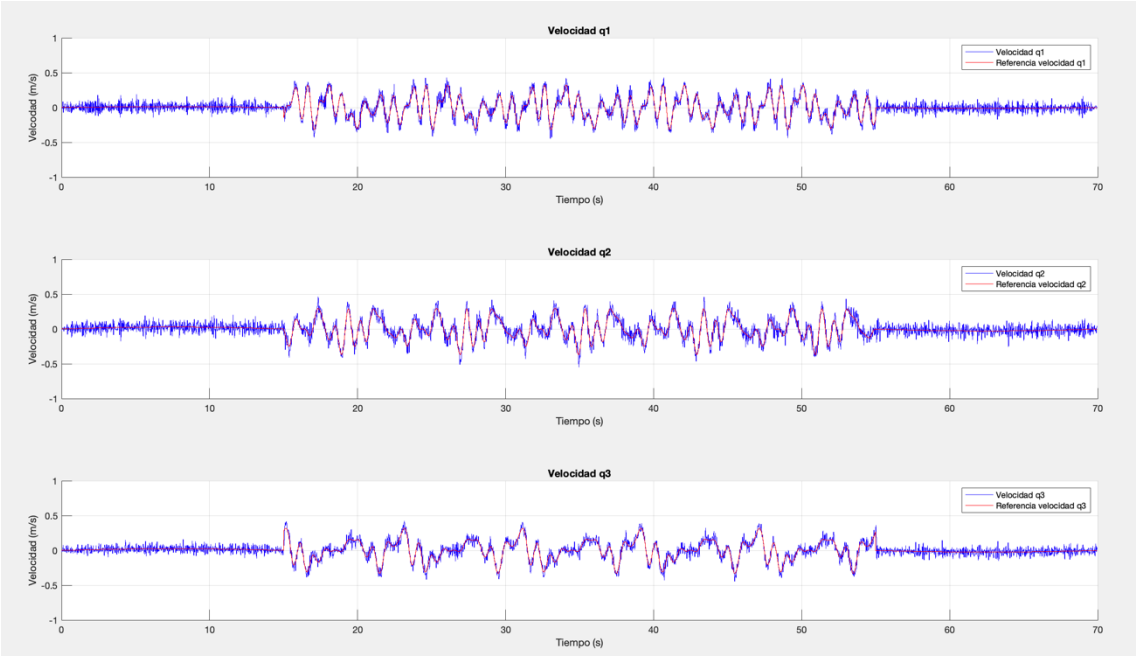


Figura 139: Experimento 2 del agente PPO+PD penalizando la posición y el error de velocidad. Velocidad de las articulaciones y de referencia.

Penalizando el error de velocidad ha sido más difícil encontrar un equilibrio entre ponderar este error y el de posición. Esto se podría resolver utilizando una función no lineal en la señal de recompensa que haga que facilite el aprendizaje al algoritmo.

Por otro lado, en todas las simulaciones realizadas, después de entrenar los algoritmos, se ha observado que la acción de control de estos es mucho mayor que los

controladores PD (debido a que el error de posición es muy pequeño). Entonces, se ha comprobado si después de un entrenamiento (2e6 iteraciones), el algoritmo es capaz de controlar sin el PD. El resultado se muestra a continuación:

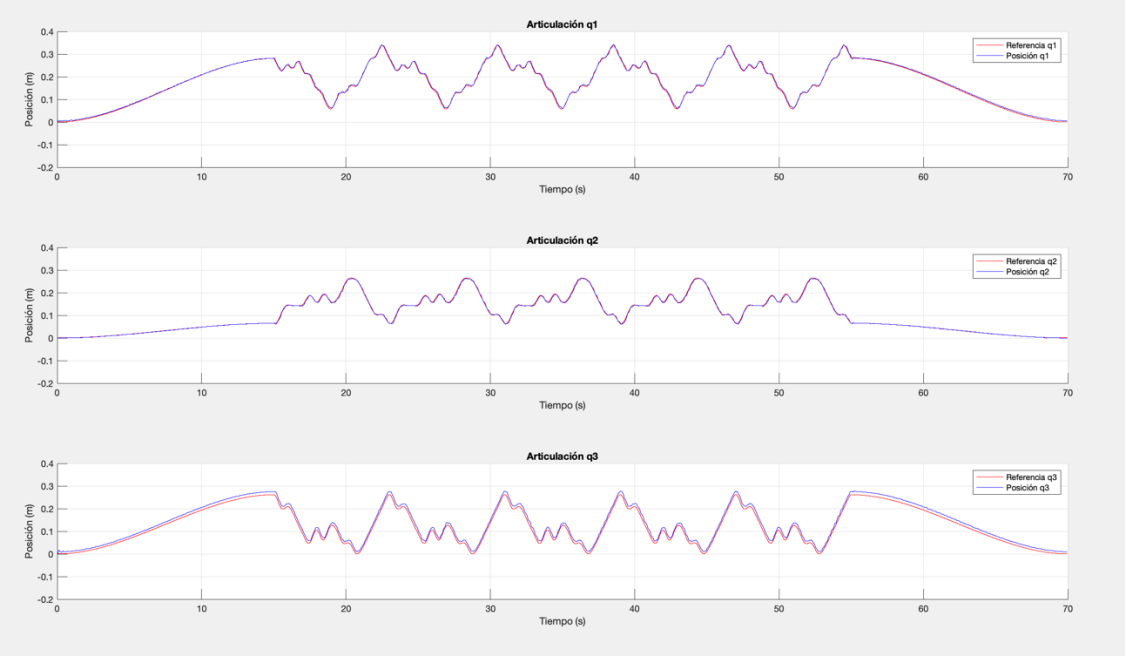


Figura 140: Experimento algoritmo PPO sin PD. Posición de las articulaciones y de referencia.

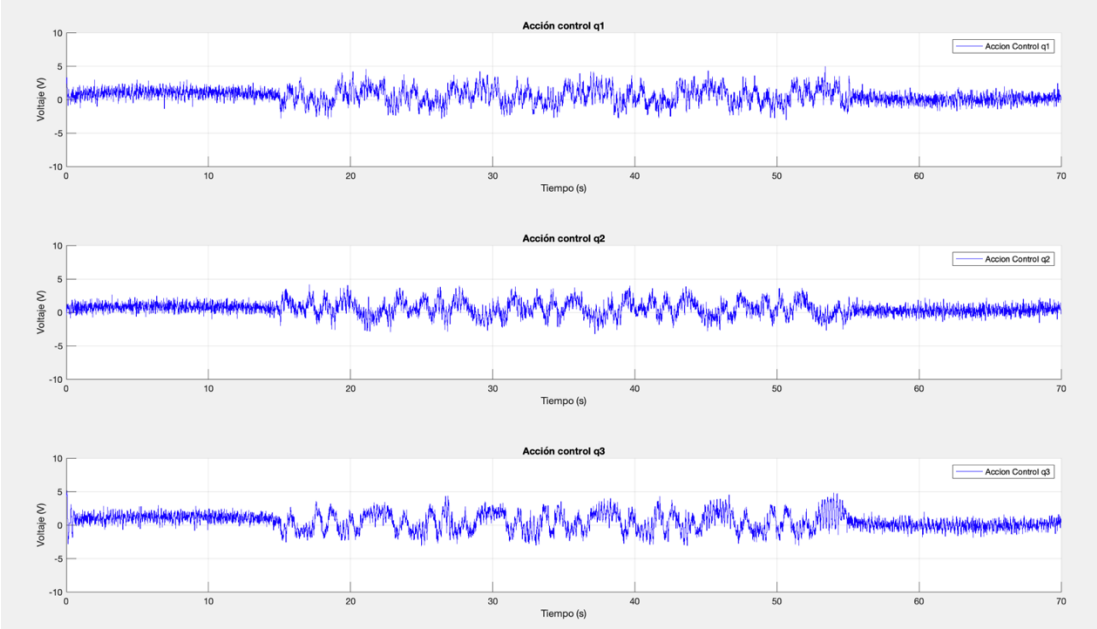


Figura 141: Experimento del algoritmo PPO sin PD. Acción de control.

Efectivamente, el algoritmo ha aprendido a controlar gran parte de la dinámica del robot y por este motivo, puede funcionar después del entrenamiento sin el controlador PD.

Por último, para cuantificar cuánto mejora el algoritmo de aprendizaje por refuerzo al PD utilizado, en la siguiente tabla se muestra una comparación entre la recompensa acumulada al final del episodio obtenida por el algoritmo PPO+PD entrenado y la recompensa acumulada que obtiene solamente la acción PD (la acción del algoritmo PPO se ha anulado). Se han utilizado 4 trayectorias diferentes de 4000 iteraciones cada una.

Recompensa acumulada (función del error de posición e incremento de la acción de control)	**PPO + PD**	**PD**	**Trayectoria**
	-1,468375	-4,097255	1
	-1,491151	-3,791122	2
	-1,469952	-3,133456	3
	-1,459443	-3,705849	4

Tabla 8: Comparación de la recompensa acumulada obtenida por PPO+PD y PD para diferentes trayectorias

7. CONCLUSIONES

En este trabajo de fin de máster se ha desarrollado una arquitectura de control para robots paralelos de rehabilitación. Se ha seleccionado tanto el hardware como el software y sobre esto se han implementado controladores de manera modular de posición y de fuerza para practicar ejercicios de rehabilitación.

El primer problema resuelto ha sido el uso de ROS2. ROS2 es un software relativamente nuevo de código abierto, que, a pesar de tener una amplia documentación, no puede abarcar todos los aspectos técnicos o problemas que pueden surgir. Pero, por otro lado, el uso de este software ha permitido poder combinar el control de bajo nivel implementado a través de programas en lenguaje C++ con ROS2, con aplicaciones y controladores más complejos que actúan en un nivel más alto.

Gracias a la implementación modular, ha sido posible validar la implementación de los controladores a través de simulaciones, tan solo ha sido necesario sustituir los programas que acceden a los interfaces físicos reales por las interfaces entre el controlador y el modelo de Simulink. También se han utilizado otras estrategias para la validación, como la publicación de los valores de posición medidos en otros experimentos para comprobar que la salida del controlador es la misma. Así pues, una implementación en ROS2 con C++ ha requerido una gran cantidad de tiempo inicial para familiarizarse y comenzar, pero la modularidad permite reutilizar el código desarrollado muy fácilmente de manera que finalmente acaba reduciendo la cantidad de trabajo de desarrollo.

ROS2 también es un software cuyo propósito es cumplir con los requerimientos de tiempo real. En este sentido, también se ha desarrollado una programación en C++ evitando realizar operaciones no deterministas como la reserva de memoria dentro del bucle de control y el acceso a disco. Para resolver esto, la reserva de memoria se realiza en el constructor de cada nodo a modo de configuración inicial. En cuanto al acceso a disco duro, se ha utilizado un segundo ordenador para liberar al ordenador de control de esta carga y que las operaciones no deterministas no afecten al procesador que está trabajando con el robot real.

Sobre la arquitectura de control se han implementado controladores de posición para los robots paralelos entre los que se incluyen un controlador algebraico y controladores proporcional-derivativos con una compensación de la gravedad. Dado que los movimientos de estos robots no son muy rápidos ni con una gran aceleración, añadir los términos de coriolis, centrífugas e inerciales, no suponen una gran ventaja y cargaría más al procesador. Sin embargo, al tratarse de una aplicación en la que hay una interacción directa entre el robot y un paciente, ha sido necesario implementar un controlador de fuerza que trabaja por encima del controlador de posición formando dos bucles de control. Para el control de la fuerza se ha seleccionado un modelo de

admitancia mediante el cual es necesario medir la fuerza y se puede controlar la respuesta del robot en cada eje por medio de tres parámetros intuitivos. En este momento sí que ha sido posible realizar ejercicios de rehabilitación con personas.

A lo largo del trabajo, se ha mencionado también que los robots paralelos tienen ventajas como mayor precisión y fuerza, pero la cinemática directa es más complicada y presentan más singularidades. Las singularidades son un problema real en ejercicios de rehabilitación, puesto que se puede perder el control del robot. Por este motivo, también se ha implementado un controlador sobre los controles de posición ya implementados, que, en primer lugar, detecta cuando el robot se encuentra en una singularidad, y escapa de esta. Esto ha sido posible gracias al uso de un sistema de captura de movimiento, OptiTrack. Este sistema de visión ha sido necesario para obtener la posición de la plataforma móvil con una gran precisión. Cuando el robot se encuentra en una singularidad, no se puede utilizar el modelo para estimar esta posición debido a que el bajo valor del determinante del Jacobiano Directo puede dar lugar a valores muy grandes e inestabilidades. Este controlador ha sido objeto de una publicación [17] en la revista Sensors, indexada Q1.

Posteriormente, se ha implementado una interfaz de usuario para el control de los robots paralelos. Siguiendo con el criterio de hacer la implementación lo más modular posible, esta ha sido basada en web para que se pueda acceder desde cualquier dispositivo. Se ha trabajado con una programación muy diferente a la de C++ anterior. Esta vez con los lenguajes JavaScript, XML, CSS y Node. El uso de la aplicación ROS2 Web Bridge ha permitido unir dos paradigmas de programación muy distintos.

En cuanto al aprendizaje por refuerzo, se ha conseguido entrenar un algoritmo que puede controlar el robot hasta sin ayuda del controlador PD y que consigue generalizar para diferentes trayectorias. Tiene el problema de que emite acciones con un alto incremento y por este motivo, es posible que estos algoritmos funcionen mejor en bucles de control en niveles más altos, dejando el nivel más bajo para los controladores basados en teoría de control convencional.

Como trabajo futuro se propone implementar una interfaz de usuario más enfocada al doctor y al paciente. En el desarrollo actual se han resuelto todas las cuestiones técnicas necesarias, pero los datos mostrados son más útiles para un ingeniero. También se propone implementar el controlador de fuerza esta vez en el robot de 4 grados de libertad, para así, poder combinar este control con otro que tiene en cuenta las singularidades. De esta manera ya se podría hacer ejercicios de rehabilitación de una manera totalmente segura. Y, por último, el trabajo realizado con el aprendizaje por refuerzo permite de una manera sencilla implementar nuevos controladores basados en este principio de manera que se podría utilizar para entrenar un modelo de fuerza que se adapte a las necesidades de cada paciente.

8. BIBLIOGRAFÍA

[1] Service Robotics for the Home: A State of the Art Review. K. Doelling; J. Sin; D. Popa. Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments, 2014.

[2] Parallel Robots. J.-P. Merlet. Springer 2006.

[3] Dynamics of Parallel Robots. Stefan Staicu. Springer, 2019.

[4] Sistemas de Tiempo Real y Lenguajes de Programación. Alan Burns; Andy Wellings. Addison Wesley, 2002.

[5] Response-Time Analysis of ROS2 Processing Chains Under Reservation-Based Scheduling. D. Casini; T. Blaβ; I. Lütkebohle; B. Brandenburg. Proceedings of the 31th Euromicro Conference on Real-Time Systems, ECRTS, 2019.

[6] https://docs.ros.org/en/eloquent/Tutorials/Composition.html

[7] Solving Rubik's Cube with a Robot Hand. Ilge Akkaya; Marcin Andrychowicz; Maciek Chociej; Mateusz Litwin; Bob McGrew; Arthur Petron; Alex Paino; Matthias Plappert; Glenn Powell; Raphael Ribas; Jonas Schneider; Nikolas Tezak; Jerry Tworek; Peter Welinder; Lilian Weng; Qiming Yuan; Wojciech Zaremba; Lei Zhang. OpenAI, 2019.

[8] Reinforcement Learning, An Introduction. Richard S. Sutton; Andrew G. Barto. The MIT Press, 2020.

[9] Deterministic Policy Gradient Algorithms. David Silver; Guy Lever; DeepMind. Proceedings of the 31st International Conference on Machine Learning, 2014.

[10] Continuous Control with Deep Reinforcement Learning. Timothy P. Lillicrap: Jonathan J. Hunt; Alexander Pritzel; Nicolas Heess; Tom Erez; Yuval Tassa; David Silver; Daan Wierstra. ICLR, 2016.

[11] Implementación de controladores dinámicos para un robot paralelo de 4 grados de libertad sobre un controlador empotrado industrial de tiempo real y FPGA. Jesús Ferrándiz, Marina Vallés, José Pulloquinga. TFG Universitat Politècnica de València, 2020.

[12] ROS2Learn: a reinforcement learning framework for ROS 2. Yue Leire Erro Nuin, Nestor Gonzalez Lopez, Elias Barba Moral, Lander Usategui San Juan, Alejandro Solano Rueda, Víctor Mayoral Vilches, Risto Kojcev. Acutronic Robotics, 2019.

[13] https://design.ros2.org/articles/realtime_background.html

[14] https://docs.ros.org/en/eloquent/Installation/Linux-Install-Debians.html

[15] https://eigen.tuxfamily.org/index.php?title=Main_Page

[16] https://optitrack.com/software/natnet-sdk/

[17] Vision-Based Hybrid Controller to Release a 4-DOF Parallel Robot from a Type II Singularity. Pulloquinga, J.L.; Escarabajal, R.J.; Ferrándiz, J.; Vallés, M.; Mata, V.; Urízar, M. Sensors, 2021.

[18] Springer HandBook of Robotics. Bruno Siciliano, Oussama Khatib. Springer, 2016.

[19] https://gym.openai.com/

[20] Reinforcement Learning Based Compensation Methods for Robot Manipulators. Yudha P. Pane; Subramanya P. Nageshrao; Jens Kober; Robert Babuˇska. Engineering Applications of Artificial Intelligence, 2018.

[21] Residual Reinforcement Learning for Robot Control. Tobias Johannink∗1,3, Shikhar Bahl; Ashvin Nair; Jianlan Luo; Avinash Kumar; Matthias Loskyll; Juan Aparicio Ojea; Eugen Solowjow; Sergey Levine. 2018.

[22] Trust Region Policy Optimization. John Schulman; Sergey Levine; Philipp Moritz; Michael I. Jordan; Pieter Abbeel. Proceedings of the 31 st International Conference on Machine Learning. JMLR 2015.

[23] Proximal Policy Optimization Algorithms. John Schulman; Filip Wolski; Prafulla Dhariwal; Alec Radford; Oleg Klimov. OpenAI, 2017.

[24] Deep Reinforcement Learning Hands-On. Maxim Lapan. Packt.

[25] https://github.com/RobotWebTools/ros2-web-bridge

[26] https://es.reactjs.org/

[27] https://code.visualstudio.com/

[28] https://git-scm.com/

2021

Pliego de condiciones

DOCUMENTO 2

AUTOR: JESÚS FERRÁNDIZ ALARCÓN

TUTORA: MARINA VALLÉS MIQUEL

TABLA DE CONTENIDO

1. OBJETO

La presente especificación técnica tiene como objeto la preparación y el control de los robots paralelos de rehabilitación del Instituto de Automática e Informática Industrial ubicados en el Laboratorio de Robótica de la Universitat Politècnica de València.

Los trabajos relacionados con el propio hardware de los tres robots paralelos incluyendo los sensores, actuadores, el cableado y los cuadros de control quedan excluidos de esta especificación técnica.

2. CONDICIONES DE LOS MATERIALES

2.1. ORDENADOR DE CONTROL

2.1.1. ESPECIFICACIONES MÍNIMAS

Debe emplearse un ordenador cuyo hardware debe cumplir con las siguientes especificaciones mínimas:

- Memoria RAM: 16 GB
- Procesador: Intel Core i7-6700 3,40 GHz, 4 núcleos
- Disco duro: 1 TB

En cuanto al procesador, puede emplearse el modelo mencionado o uno diferente que tenga las mismas prestaciones.

2.1.2. INTERFACES FÍSICOS

Para el control de las salidas y entradas de los robots deben utilizarse las siguientes tarjetas de adquisición de datos:

- PCI-1784U de Advantech. Tarjeta de lectura de encoders.
- PCI-1720U de Advantech. Tarjeta de salidas analógicas.

2.1.3. SOFTWARE

Se debe instalar Ubuntu Desktop 18.04 LTS en el ordenador de control. Para el acceso a las tarjetas de adquisición de datos debe instalarse la librería BIODAQ de Advantech. Se debe instalar ROS2, concretamente la versión Eloquent, la instalación del paquete Debian es suficiente, no es necesario instalar a partir de compilar el código fuente. Se recomienda instalar la versión desktop para disponer de herramientas que ayudan a visualizar la red de nodos y datos relevantes.

Para el acceso a los datos de la captura de OptiTrack, debe instalarse la SDK NatNet. Por último, también es necesario instalar la librería Eigen de C++ para el cálculo de operaciones matemáticas.

Para la ejecución de los controladores basados en aprendizaje por refuerzo, es necesario instalar el gestor de paquetes de Python pip, después se debe instalar virtualenv y crear un entorno virtual con esta herramienta. Por último, se debe activar el entorno ejecutando el fichero bin/activate del entorno virtual e instalar la librería Stable Baselines con todas las dependencias que requiere. Se debe instalar la versión 1.14.0 de TensorFlow.

Por último, se debe instalar como mínimo la versión 2020b de MATLAB.

2.2. ORDENADOR AUXILIAR

2.2.1. ESPECIFICACIONES MÍNIMAS

Se debe disponer de un ordenador con las siguientes especificaciones mínimas de hardware:

- Memoria RAM: 8 GB
- Procesador: Intel Core i3-10100 3,6 GHz, 4 núcleos
- Disco duro: 1 TB

En cuanto al procesador es posible utilizar otro modelo que cumpla con las especificaciones técnicas del propuesto.

2.2.2. SOFTWARE

Para el ordenador auxiliar es necesario instalar el sistema operativo Ubuntu Desktop 18.04 LTS. Posteriormente, se debe instalar la versión Eloquent de ROS2, los paquetes Debian también son suficientes para este ordenador y se recomienda instalar la versión desktop.

Se debe instalar el gestor de versiones de Node n, e instalar como mínimo la versión 12.0.0 de Node. Después, debe instalarse también el gestor de dependencias de JavaScript npm e instalar también el programa npx para crear la aplicación de React. Posteriormente, se debe descargar e instalar el programa ros2-web-bridge desde el repositorio de GitHub.

2.3. SISTEMA OPTITRACK

Deben adquirirse los siguientes elementos de la marca OptiTrack:

- 10 cámaras Flex13
- 2 dispositivos OptiHub2
- Marcadores de material reflectante esféricos de 14mm de diámetro

3. CONDICIONES DE LA EJECUCIÓN

3.1. CABLEADO Y COMUNICACIONES EN RED

Para conectar el ordenador de control con uno de los robots basta con conectar mediante un cable DB37 el propio ordenador de control con el cuadro de control del robot que se quiera utilizar. Cada robot tiene su propio cuadro de control.

Con el fin de que el ordenador de control y el ordenador auxiliar puedan intercambiar información a través de ROS2 debe comprobarse que ambos se encuentran en la misma red y de que la variable de entorno ROS_LOCALHOST_ONLY es igual a 0.

3.2. INSTALACIÓN DE LOS PROGRAMAS

Para instalar los programas de control de los robots es necesario acceder a los repositorios privados del Laboratorio de Robótica de la UPV en GitHub. Se deben clonar los repositorios pr_4dof, pr_3dof y gym-parallel_robot en el ordenador de control. Se deben compilar los paquetes del repositorio pr_4dof y pr_3dof con colcon.

Para poder utilizar los controladores de aprendizaje por refuerzo, se debe activar el entorno virtual en el que se haya instalado la librería Stable Baselines.

En el ordenador auxiliar es necesario clonar los repositorios pr_pc, pr_web y pr_db_server. Se deben compilar los paquetes de pr_pc con colcon. También se debe ejecutar *npm install* en el código fuente de pr_web y pr_db_server para instalar todas las dependencias que necesiten.

3.3. EJECUCIÓN DE LOS PROGRAMAS

En primer lugar, es necesario ejecutar el script de lenguaje bash que se crea en la carpeta install cuando se compila con colcon cada entorno de trabajo, de la siguiente manera para el caso de pr_4dof: *./pr_4dof/install/setup.bash*. Después, ya se pueden utilizar los archivos que lanzan los controladores, estos se encuentran en el paquete pr_bringup. Para lanzar una trayectoria con el controlador PDG se debe ejecutar: *ros2 launch pr_bringup pr_pdg.launch.py*.

Para guardar los datos de un experimento en el ordenador auxiliar hay que ejecutar primero *./pr_pc/install/setup.bash* y después ejecutar en el caso del robot de 4 grados de libertad: *ros2 launch pr_storage recorder.launch.py*.

Con relación al servidor web, para ejecutarlo, es necesario primero lanzar el servidor de trayectorias mediante la sentencia *node pr_db_server/bin/pr_db.js*. Después se debe ejecutar *npm start* dentro de la carpeta pr_web. De esta manera ya se puede acceder a la interfaz gráfica por medio de un navegador.

Para lanzar un control con aprendizaje por refuerzo, es necesario activar el entorno virtual en el que se haya instalado Stable Baselines. Después exportar la variable de entorno ParallelRobot3DOF para utilizar los entornos del robot de 3 grados de libertad y ejecutar *ros2 run pr_3dof_rl pr_rl_ppo* para entrenar un agente PD+PPO. En el caso de utilizar un modelo ya entrenado solamente para realizar predicciones se debe ejecutar *ros2 run pr_3dof_rl pr_rl_inference*. Después, en una sesión distinta de terminal, se debe ejecutar *ros2 launch pr_3dof_bringup pr_rl_pd.launch.py*. Por último, se debe ejecutar el script de MATLAB macro_ros2_rl.m.

3.4. CALIBRACIÓN DE OPTITRACK

Antes de realizar experimentos en los que se requiera obtener la información del sistema de captura de movimiento es necesario calibrar las cámaras, al menos una vez al día. Para la calibración hay que seguir los pasos indicados en el programa Motive:

- Primero se debe eliminar de la escena cualquier marcador y hacer que solamente sean visibles los marcadores de la barra de calibración.
- Se debe mover la barra a lo largo de la escena para que todas las cámaras puedan ver los marcadores. El programa Motive indica el momento en el que se puede parar.
- Después se debe colocar en la escena la escuadra de calibración para indicar el plano 0.
- Finalmente se guarda y se aplica el archivo de la calibración.

4. PRUEBAS Y AJUSTE FINALES

4.1. SIMULACIÓN

Para la validar la implementación de los controladores se debe realizar una simulación por medio del modelo de Simulink. Se utilizarán los nodos del controlador implementado sustituyendo los nodos de los encoders y actuadores reales por los simulados: EncodersSimulink y MotorsSimulink del paquete pr_sensors_actuators.

Se deben guardar por medio del ordenador auxiliar los datos relevantes del control y se considerara valida la prueba cuando los datos de la salida de la posición de las articulaciones y de las acciones de control sean similares a las simulaciones previas realizadas con los controladores implementados en Simulink.

4.2. REPETICIÓN DE EXPERIMENTO OFFLINE

La siguiente prueba a realizar consiste en sustituir el nodo de los encoders reales por el nodo que lee un fichero y reproduce sus iteraciones, de manera que realmente se está reproduciendo un experimento. Este componente es pr_aux::Replayer. Al

utilizarse el este componente, el controlador calculará los valores de la acción de control a partir de unos valores de posición no medidos, si no correspondientes a los de un experimento o simulación anterior. Si los nuevos valores de la acción de control son similares a los del experimento reproducido, la prueba se considerará válido.

4.3. PRUEBA CON LOS ROBOTS REALES

4.3.1. ENCODERS

Antes de utilizar los controladores con los programas reales, para cada uno de los tres robots, se debe comprobar el buen funcionamiento de los Encoders. En primer lugar, se deben liberar los frenos (en el caso de los robots de 4 grados de libertad) y después se debe leer el valor de posición obtenido en las articulaciones. Para cada pata, se debe repetir el siguiente proceso:

- Mover manualmente la pata una distancia de aproximadamente 20 centímetros.
- Medir la distancia que se ha movido con una regla.
- Comprobar que la lectura de posición a partir de los Encoders coincide con la medida anterior.
- Devolver la pata a la posición inicial.
- Comprobar que el nodo de los encoders ofrece una medida de posición 0.

Si la distancia medida con la regla es la misma que la proporcionada por el nodo de los encoders y después al devolver la pata a la posición inicial la medida es 0, se considera la prueba como válida.

4.3.2. CONTROL DE UNA ÚNICA ARTICULACIÓN ACTIVA

Para comprobar el buen funcionamiento de todos los elementos, tanto hardware como software básico, de los robots de 4 grados de libertad se dispone de trayectorias que mueven únicamente una sola pata. Para ejecutar esta trayectoria, la plataforma móvil debe estar desmontada. Se debe ejecutar esta trayectoria con un controlador PD, PD+G o algebraico para cada una de las patas y verificar que la respuesta es correcta.

4.3.3. IMPLEMENTACIÓN DE UN NUEVO CONTROLADOR

Cuando se implemente un nuevo controlador, después de haber superado las pruebas de simulación y de repetición del experimento offline, es posible utilizarlo con los robots reales. Se deben almacenar todos los datos relevantes (como mínimo posición de referencia, posición real y acción de control) por medio del ordenador auxiliar. Una vez guardados los datos, se debe valorar la respuesta del controlador por medio de los siguientes criterios:

- Error de posición

$$JIAE = W_{IAE} \cdot \sum_{i=1}^{n\_patas} \sum_{j=1}^{n\_iteraciones} 0.01 \cdot \left| q_{ref(j,i)} - q_{medida(j,i)} \right|$$

- Incremento de la acción de control

$$JIADU = W_{IADU} \sum_{i=1}^{n\_patas} \sum_{j=1}^{n\_iteraciones} \left| u_{(j,i)} - q_{(j-1,i)} \right|$$

$$W_{IAE} = 1 \; ; \; W_{IADU} = 3.8 \cdot 10^{-6}$$

Se valorará que el valor de ambas métricas JIAE y JIADU sea el mínimo posible.

2021

Presupuesto

DOCUMENTO 3

AUTOR: JESÚS FERRÁNDIZ ALARCÓN

TUTORA: MARINA VALLÉS MIQUEL

Documento 3: Presupuesto

Ref	Ud	Descripción	Precio	Cantidad	Total
Materiales					
PC1	U	Ordenador industrial	1.100,000 €	1	1.100,000 €
PC2	U	Teclado	40,000 €	2	80,000 €
PC3	U	Ratón	20,000 €	2	40,000 €
PC4	U	Monitor	100,000 €	2	200,000 €
PC5	U	Ordenador auxiliar	700,000 €	1	700,000 €
IN1	U	Sensor de fuerza F/T Sensor delta	3.475,000 €	1	3.475,000 €
IN2	U	Tarjeta de adquisición PCI-1784U Advantech	447,090 €	1	447,090 €
IN3	U	Tarjeta de adquisición PCI-1720U Advantech	408,270 €	1	408,270 €
OPT1	U	Cámaras Optitrack Flex13	1.000,000 €	10	10.000,000 €
OPT2	U	OptiHub 2	300,000 €	2	600,000 €
LI1	U	Licencia de Motive	621,000 €	1	621,000 €
LI2	U	Licencia de MATLAB	2.000,000 €	1	2.000,000 €
				Subtotal	19.671,360 €

Ref	Ud	Descripción	Precio	Cantidad	Total
Mano de obra					
MO1	h	Ingeniero electrónico	20,000 €	100	2.000,000 €
MO2	h	Ingeniero de control	20,000 €	600	12.000,000 €
MO3	h	Operario	15,000 €	100	1.500,000 €
				Subtotal	15.500,000 €

Desglose de tiempo invertido por actividad		Cantidad	Ud
Ingeniero de Control			
Selección de hardware y software		40	h
Desarrollo de componentes para sensores y actuadores		70	h
Desarrollo de librerías para los robots		80	h
Desarrollo de controladores de posición		50	h
Integración con Optitrack		60	h
Desarrollo de controlador para escapar de una singularidad		50	h
Diseño e implementación de controladores de fuerza		60	h
Implementación de la interfaz gráfica		40	h
Implementación de controladores basados en inteligencia artificial		70	h
Desarrollo de experimentos y evaluación de los resultados		80	h
	Total	600	h

Ingeniero de electrónico			
Mantenimiento de los cuadros de control		40	h
Desarrollo de experimentos		60	h
	Total	100	h

Operario			
Mantenimiento de los robots		30	h
Desarrollo de experimentos		70	h
	Total	100	h

Ref	Ud	Descripción	Precio	Cantidad	Total
Maquinaria					
MQ1	h	Máquina taladradora de agujeros	4,000 €	0,2	0,800 €
				Subtotal	0,800 €
Medios auxilares					
%			4%	35.172,160 €	1.406,886 €

TOTAL PRESUPUESTO DE EJECUCIÓN MATERIAL	36.579,046 €
Beneficio industrial (6%)	2.194,743 €
TOTAL PRESUPUESTO DE EJECUCIÓN POR CONTRATA	38.773,789 €
IVA (21%)	8.142,496 €
TOTAL PRESUPUESTO BASE DE LICITACIÓN	46.916,285 €