Escuela Politécnica Superior

Diseño, simulación y control de un astronauta robótico humanoide

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:
Adrián Trujillo López
Tutor/es:
Jorge Pomares Baeza
Jose Luis Ramón Carretero

Universitat d'Alacant
Universidad de Alicante

Junio 2018

Diseño, simulación y control de un astronauta robótico humanoide

Autor:

Adrián Trujillo López

Tutores:

Jorge Pomares Baeza

Jose Luis Ramón Carretero

Dpto. de Física, Ingeniería de Sistemas y Teoría de la Señal

Grado en Ingeniería Robótica

Escuela Politénica Superior

Universitat d'Alacant
Universidad de Alicante

Alicante, Junio 2021

Resumen

Durante muchos años de expansión tecnológica el ser humano ha soñado y trabajado para llegar a explorar el espacio exterior, con el fin de ayudar en esta tarea y aportar una pizca de ayuda a todo el proceso se plantea un trabajo en el que pretende realizar el diseño de un robot astronauta con características humanoides que pueda trabajar en la estación espacial internacional ayudando a los astronautas en sus diferentes tareas de mantenimiento diarias. Por esta razón y como se desarrollará a lo largo del trabajo se realizan diferentes modificaciones en las características físicas del robot, así como proporcionarle manos de características como las humanas y quitarle las piernas, pero dándole una mochila propulsora para que pudiese moverse en algún momento.

Además de lo comentado anteriormente, se creará y diseñará una simulación en Gazebo en la que se incluirá el robot comentado junto con un modelo a escala real de la estación espacial internacional, para que se puedan realizar todo tipo de pruebas necesarias en un futuro. Para finalizar se utilizarán las tecnologías de ROS y Ros Control para implementar diferentes controladores cartesianos de trayectorias que utilicen características de visión por computador para realizar un ajuste más preciso del movimiento del brazo robótico. Se implementarán dos controladores cinemáticos y dos controladores dinámicos, los controladores dinámicos cuentan con la característica de que a partir de los términos utilizados se pueden obtener un valor de la velocidad en el espacio cartesiano que toma la base del robot a causa de las inercias provocadas al mover el brazo del mismo.

Agradecimientos

En estos agradecimientos me gustaría agradecerles a todas las personas posibles que han influenciado lo más mínimo en poder llevar a cabo el desarrollo de este trabajo. Pero primeramente quiero agradecer enormemente a mis tutores del TFG Jorge y Jose Luis porque siempre me han ayudado y han sabido dirigirme y guiarme en el desarrollo de este trabajo. Durante todo el proceso siempre tenían tiempo para reunirse conmigo si tenía alguna duda o necesitaba apoyo, esto es algo que puede parecer normal de primera impresión, pero para mí es una cosa que siempre agradezco y que puedo ver que no todo el mundo puede conseguir.

Además de todo esto siempre me han dejado mucha vía libre en cuanto a cómo quería realizar el desarrollo y han podido aconsejarme sobre qué pasos debería seguir en cada momento a pesar de que siempre eran mis decisiones. Quiero darles las gracias por esto también porque, aunque es mi trabajo nunca me he sentido solo y siempre se sentía como un equipo. Ojalá que, aunque pasen los años después de este trabajo podamos seguir manteniendo la relación tan buena de amistad y profesionalidad que hemos conseguido.

Además de mis tutores me gustaría agradecer a todos mis amigos de la carrera, porque siempre han sabido y podido estar conmigo cuando lo necesitaba y ayudarme durante todos estos 4 años, muchas gracias por haberme hecho la persona que soy ahora, sin ellos nunca podría haber llegado hasta este momento nunca y estoy feliz de que hayamos podido llegar todos juntos.

Por último y no menos importante, me gustaría agradecer a mi familia y mi pareja. Por todos los momentos de estrés que he podido ocasionarles y por el apoyo emocional que han sabido brindarme siempre.

Cada uno de ellos es un pilar muy importante en todo el proceso y no me gustaría dejar a ninguno atrás, muchas gracias a todos por estos años que me han ayudado a poder completar este trabajo.

Índice de contenido:

Índice de figuras

1. Introducción

Durante toda la historia los humanos se han sentido atraídos por las máquinas capaces de imitar las funciones de los seres vivos y que se asemejen a estos. De ello que los griegos ya tenían incluso una palabra *automatos* de la cual derivada la palabra actual **autómata**: se dice de la máquina que imita la figura y movimientos de un ser animado. A lo largo de la historia siempre ha habido personas que se han dedicado a realizar, inventar y perfeccionar diferentes sistemas automáticos cuya utilidad y funcionamiento ha sido adelantado a su época y algo disruptivo con la realidad.

Por ejemplo, durante los siglos XVII y XVIII ya se crearon ciertos ingenios mecánicos que tenían alguna de las características de los robots actuales. Estos dispositivos ya se empezaban a introducir en la vida de las personas de la nobleza y en las ferias como elemento de ocio. No fue hasta 1921 que se utilizó por primera vez palabra **robot** en una obra de teatro como ciencia ficción. Más adelante en 1945 Isaac Asimov publicó en una revista bien conocidas *tres leyes de la robótica*:

1. Un robot no puede perjudicar a un ser humano, ni con su inacción permitir que un ser humano sufra daño.

2. Un robot ha de obedecer las órdenes recibidas de un ser humano, excepto si tales órdenes entran en conflicto con la primera ley.

3. Un robot debe proteger su propia existencia mientras tal protección no entre en conflicto con la primera o segunda ley.

Gracias a estos casos lo que se tenía en el momento era únicamente una gran divulgación y difusión sobre la robótica que plantarían las bases para la llamada revolución industrial donde se podrá ver la mayor utilidad y ayuda de los robots en la sociedad hasta el momento. Sin embargo, la robótica se preocupa sobre el estudio de las maquinas que pueden remplazar a los humanos en la ejecución de tareas tanto en cuanto al esfuerzo físico y como en cuanto a toma de decisiones.

Por esto que, aunque la robótica industrial sea posiblemente la rama más desarrollada de la robótica existen multitud de ramas que la hacen todo lo interesante que es, así como la

robótica móvil, la robótica social, la robótica espacial, etc. Es en estos campos donde se puede ver lo diferentes que pueden llegar a ser todo tipos de robots y la diversidad de configuraciones que puede llegar diseñar el ser humano, pero a fin de cuentas siempre se ha dicho que "El mejor pasatiempo del ser humano es el propio ser humano". Por esto que a lo largo de la historia de la robótica el ser humano siempre se ha centrado en la robótica humanoide de la cual se va a dedicar cierta parte de este trabajo.

La robótica espacial es una rama relativamente nueva de la robótica, aproximadamente existe desde hace unos 40 años, así como toda la tecnología relacionada con el espacio exterior, pero de la misma manera esto le atribuye un encanto mucho mayor que a cualquier otra rama o campo de la robótica. La robótica espacial ha sido desarrollada principalmente por el extremado coste que tiene llevar a los humanos al espacio y por las condiciones tan hostiles del medio que hacen extremadamente difícil la supervivencia del ser humano en el espacio sin buen un soporte de sistema vital.

Actualmente la robótica espacial se encuentra en su gran mayoría combinada con la tele operación. Esto es debido a los avances que ha habido en cuanto a la tecnología de telepresencia a la hora de controlar un sistema robótico de manera remota ya que permite a los operadores manejar brazos robóticos en el espacio como si se encontraran allí mismo. Esta tecnología ha estado dando tan buenos resultados que incluso parece irrelevante seguir desarrollando robots totalmente autónomos.

Pero estas características dadas tan buenas por la telepresencia se quedan obsoletas a medida que el robot se aleja de la órbita terrestre puesto que en estos casos la comunicación directa se hace imposible de soportar por el sistema, existiendo por ejemplo un retardo en la comunicación de aproximadamente 15 min entre Marte y la Tierra. Esta razón es el principal motivo para que el ser humano se haya centrado en el desarrollo de robots cada vez más autónomos para poder explorar los lugares cada vez más alejados de la Tierra.

Igualmente, el desarrollo e innovación en este tipo de sistemas autónomos también es de gran utilidad para aumentar las capacidades de los robots teleoperador y la tecnología de telepresencia que actualmente gobierna en la órbita terrestre.

1.1. Motivación y Justificación

Personalmente nunca he estado especialmente involucrado con los temas espaciales ni la tecnología dedicada al espacio, siempre ha sido un campo que se escapaba de mis limites, pero a la vez siempre me ha transmitido mucha admiración y extrema curiosidad por la cantidad de cosas inciertas que hay en él.

Por esto que en el momento en que mi tutor me propuso trabajar en algo relacionado con este tema y un proyecto que llevaban entre manos junto con otra universidad de Inglaterra y de forma muy remota con la NASA no pude dejar pasar la oportunidad. El proyecto presentado en este trabajo es muy interesante y me ha ofrecido la posibilidad de aprender mucho sobre la robótica espacial actual y su historia.

La curiosidad siempre ha sido lo que me ha movido a hacer cosas nuevas y diferentes durante toda la vida y creo que a la mayoría de gente cuando se le propone un trabajo como este también estaría orgullosa de poder participar en él. Mi motivación para poder realizar este trabajo siempre ha sido esa, el pensar que puedo aportar mi pequeña parte tanto a la ciencia como a la exploración espacial y ayudar en cierta manera a agilizar el desarrollo de nueva tecnología en este campo.

1.2. Objetivos

El objetivo principal de este proyecto siempre ha sido crear un entorno de simulación para trabajar con el robot TALOS de la empresa PAL Robotics utilizando el framework de ROS y las capacidades que este ofrece, así como ROS Control. Y dejar un buen trabajo que pueda ser ampliamente utilizado y extendido por futuros alumnos o compañeros de oficio.

También se propuso crear varios diferentes controladores para poder trabajar en el espacio de manera óptima y analizar su funcionamiento. Sin embargo, estos objetivos eran bastante ambiciosos teniendo en cuenta el tiempo que se tiene para realizar el trabajo.

Primeramente, se situó el objetivo de modificar la estructura del robot TALOS original para adaptar a este al espacio, seguidamente más adelante se buscó la forma de incluir un modelo

de la Estación Espacial Internacional de la NASA en una simulación de Gazebo para que el robot pudiera realizar ciertas tareas de manipulación en la estación espacial.

Una vez construido el espació de simulación y se tuviera al robot correctamente simulado se propuso como siguiente objetivo crear algún controlador cartesiano de posición básico utilizando Ros Control. Y Finalmente, después de tener este controlador el objetivo fue adaptar y extender este controlador a algunos más complejos que tuvieran unas mejores características para funcionar en condiciones espaciales como puede ser: Seguimiento de trayectorias, localización del robot con visión por computador, características dinámicas del robot, etc.

1.3. Estructura

La memoria del TFG se divide de manera cómoda pensando en la lectura futura de cualquier persona que se interese por aprender en este campo de la robótica y del espacio.

Primeramente, se ha comenzado realizando una pequeña introducción sobre qué es un robot a nivel general, los comienzos del control de robots y los comienzos de la tecnología espacial, así como de los robots espaciales. En este apartado también se presenta la motivación para realizar este TFG, así como los objetivos propuestos en todo momento.

Seguidamente el lector se encontrará con 2 apartados en los que se explica el estado actual de la técnica y el desarrollo llevado a cabo en el proyecto. Como es lógico en el primer apartado de estos dos se plantean las diferentes soluciones que hay y ha habido para los problemas surgidos en el trabajo. Todos los conceptos explicados en el estado del arte han influenciado y ayudado a desarrollar el trabajo explicado en el apartado número tres. De manera que el lector puede intuir fácilmente en base a qué conocimientos se ha basado el razonamiento para desarrollar todas las partes del TFG.

Finalmente se presentarán dos apartados para tratar y presentar los diferentes resultados de los controladores implementados, así como las conclusiones obtenidas a partir de estos resultados.

2. Estado del Arte

En el siguiente capítulo se tratará sobre los métodos de control existentes y utilizados en la actualidad con el fin de poner en contexto el trabajo presentado en los apartados siguientes a lo largo del presente documento. Se comentarán y explicarán las técnicas más generales en cuanto a robots manipuladores y se hará cierto hincapié en los robots redundantes, es decir, aquellos que tienen más grados de libertad de los que se necesita para llegar a cualquier punto de su espacio de trabajo ya que el robot del que se ha hecho uso cumple estas características.

A nivel general en los siguientes apartados se explicarán los dos métodos o estrategias de control más utilizadas, el control cinemático y el control dinámico. Destacando las características que ofrece cada una y porque se debe utilizar cada una en debidas situaciones. Además, se hablará de las capacidades que ofrece realizar un control en el espacio de operación del robot tal y como se desarrolla en [1].

Después de esta breve contextualización sobre el control de robots manipuladores, se procederá a explicar el uso de la robótica en el espacio exterior. Primeramente, se realizará una breve introducción de los tipos de robots que se han utilizado en el espacio, se explicará las propiedades que poseen estos y que los hacen aptos para el cumplimiento de sus correspondientes tareas.

Seguidamente se hablará de la importancia de los robots espaciales en órbita y las especificaciones que deben cumplir en la actualidad, así como de la importancia de los robots humanoides para trabajar en la estación espacial. Por último, también se comentarán ciertos controladores específicos de robots espaciales en los que se utilizan métodos de visión por computador.

2.1. Control de robots

El control de robots es la base para cualquier sistema robótico por ello que se ha dedicado una gran cantidad de recursos y tiempo a investigar las diferentes áreas de este campo, la literatura encontrada sobre este campo claramente es muy amplia en los siguientes

subapartados se detallarán los conceptos más importantes que se han podido extraer de [1], [2].

Para realizar el control de un manipulador es necesario realizar y establecer ciertos modelos matemáticos, así como modelos cinemáticos, diferenciales y dinámicos. Estos son los que se explicarán en los siguientes apartados, mediante estos modelos es posible obtener las relaciones entrada/salida de las posiciones del robot en base a diferentes parámetros. Esta parte del control es básica para poder establecer las diferentes leyes de control al sistema robótico.

Además del modelado del robot, un apartado previo que se debe integrar en el sistema de control es la generación de trayectorias. Es necesario definir trayectorias suaves, dependiendo de la tarea a realizar se deberá estudiar el tipo de trayectoria que se deberá realizar y en qué espacio coordenadas deberá especificarse (cartesianas o articulares). Sobre la generación de trayectorias y cómo aplicarlas se puede leer más en detalle en [1].

De manera que finalmente el problema de controlar un robot un brazo robótico se puede resumir en la tarea de reducir un error en la referencia establecida por el controlador. La estrategia utilizada para reducir el error y el espacio de espacio de coordenadas utilizados será lo que se explicará en los siguientes subapartados.

2.1.1. Control cinemático

El control cinemático se encarga de obtener los valores articulares para llevar una cadena cinemática hasta la referencia deseada. El modelo cinemático de un robot relaciona los valores articulares de las articulaciones pertenecientes a la cadena cinemática con la posición en el espacio del extremo del robot. Sin embargo, en la literatura [1][2] también se necesita de un modelo diferencial que relaciona las velocidades articulares en las velocidades del extremo del robot.

Es en la fase inicial del control cinemático donde se debe especificar las trayectorias que deberían seguir las articulaciones del manipulador. En esta fase dependiendo de la tarea diferentes estrategias pueden ser llevadas a cabo, siendo posible generar la trayectoria en diferentes sistemas de referencia. Si la trayectoria genera diferentes valores articulares a lo

largo del tiempo el controlador puede utilizar directamente estos controladores, si se utiliza una trayectoria cartesiana se puede optar por utilizar el modelo cinemático del robot para obtener la relación entre cada posición cartesiana y sus correspondientes valores articulares y enviar estos al controlador o controlar el error cartesiano directamente en el espacio de operaciones como se explicará en el apartado 2.1.3 y se detalla en [1].

Una vez elegida la referencia y la trayectoria que deberá realizar el robot se debe emplear un bucle de control cinemático de alto nivel que realice el control para reducir el error de la posición del robot con la posición deseada especificada en la trayectoria. Para esta tarea lo más utilizado en la literatura y el método más básico que se puede utilizar es un bucle de control PID.

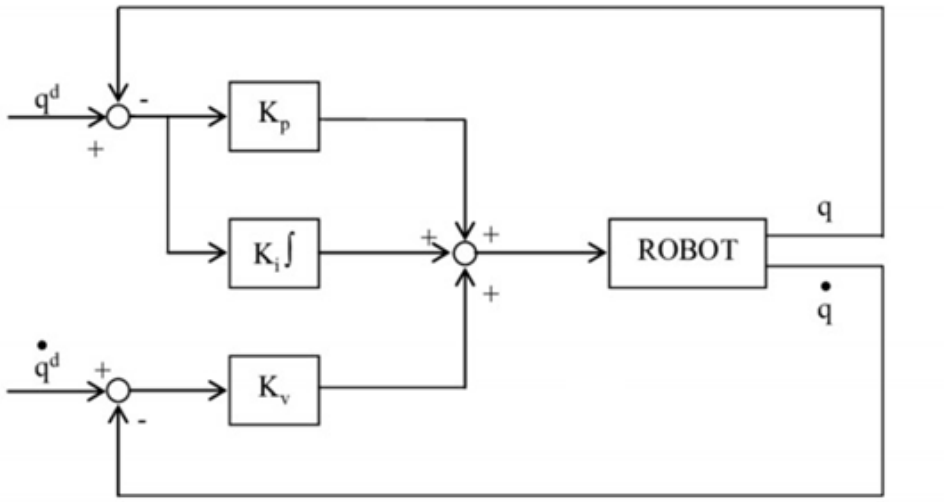

Figura 2.1: Bucle de control PID. **Fuente:** www.researchgate.net

En el bucle mostrado arriba en la figura 2.1 se puede observar cómo mediante una constante proporcional (K_p) se reduce el error en posición, una constante derivativa (K_v) se reduce el error en velocidad y finalmente mediante una constante Integral (K_i) se reduce el error en la integral de la posición. Es extremadamente frecuente encontrarse en la literatura con controladores que optan por eliminar esta constante integral y se implementa únicamente un bucle control PD.

Además, cabe destacar que como se ve en el bucle de control la retroalimentación debe introducir en todo momento la posición (q) y velocidad ($\dot{q}$) del robot para que el error pueda ser calculado.

El control cinemático no cuenta con las propiedades dinámicas del robot como puede ser las inercias generadas, las fricciones, etc. Por esta razón el control cinemático puede perder precisión en las tareas que requieran velocidades o aceleraciones elevadas. Para estas tareas se considera entonces que la mejor opción es optar por una estrategia de control dinámico.

2.1.2. Control dinámico

Por lo general a los robots se les demandan altas prestaciones de velocidad y precisión de movimiento. En el control cinemático explicado en el apartado anterior trata de seleccionar las trayectorias que idealmente deberá seguir el robot y teniendo en cuenta sus limitaciones intentará ajustarse lo mejor posible a las especificaciones del movimiento dadas por el usuario.

A diferencia de este, al control dinámico, explicado en este apartado, se le encarga la misión de procurar que las trayectorias realmente seguidas por el robot sean lo más parecidas a las propuestas como referencia. Para ello hace uso del conocimiento del modelo dinámico del robot. Normalmente este control se aplica a las trayectorias articulares del robot y pueden darse 2 tipos generalmente.

El control monoarticular y el control multiarticular, estos se distinguen en que el primero desprecia la interacción entre los grados de libertad del robot. Mientras que el multi articular considera el robot como el sistema multivariable que realmente es. La técnica de control básica en este tipo de controladores también está basada en un control PID, pero adaptado al control dinámico.

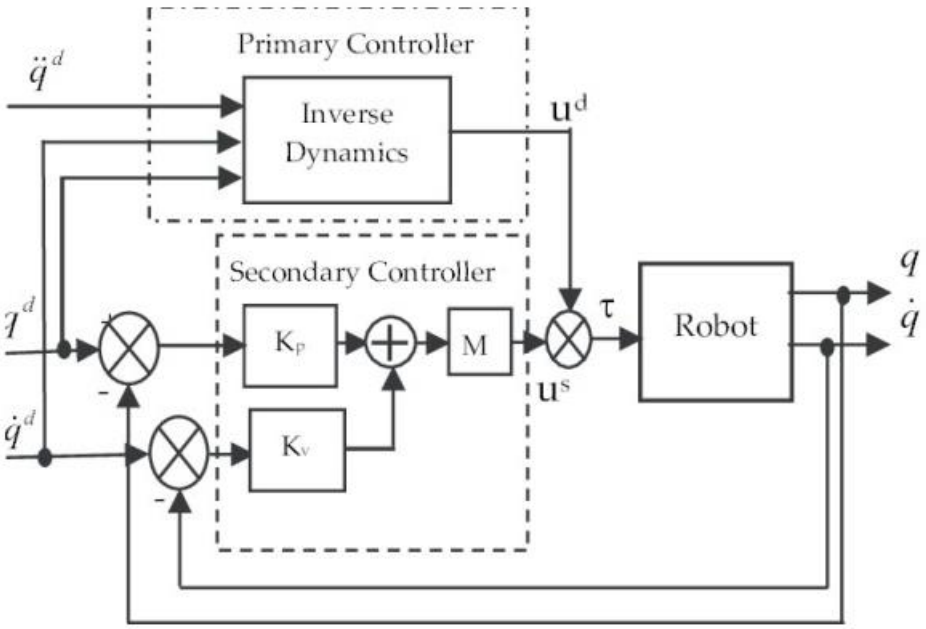

Figura 2.2: Bucle de control dinámico utilizando un PD. **Fuente:** www.researchgate.net

El modelo dinámico de un robot relaciona el movimiento del robot con las fuerzas aplicadas al mismo. Este modelo relaciona la localización del robot con diferentes tipos de parámetros como pueden ser las fuerzas, pares de las articulaciones y los diferentes parámetros dimensionales del robot. Generalmente para cualquier robot el modelo dinámico toma la siguiente forma:

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) \qquad (2.1)$$

Donde τ es el vector de pares articulares ejercidos en cada articulación del robot. $M(q)$ es la matriz de inercias del robot, $C(q,\dot{q})$ es la matriz de Coriolis y $G(q)$ es la matriz de gravedad que indica el efecto de esta fuerza sobre el robot en una posición q. Sin embargo, este modelo dinámico requiere una cantidad de cálculos que a nivel computacional suele ser una carga extremadamente grande. En [1][2] se hace referencia y se explican diferentes métodos para calcular este modelo como por ejemplo las dos más comentadas en la literatura son la formulación de Lagrange-Euler o la Newton Euler, aunque a la hora de calcular estos modelos actualmente se está haciendo uso métodos numéricos.

2.1.3. Control en el espacio de operaciones

Además de los esquemas comentados anteriormente existen otras soluciones con el fin de adaptar los esquemas de control al espacio de operaciones y así evitar trabajar con los valores articulares del robot y los errores derivados de este. Se ha decidido añadir este apartado para introducir los conceptos usados en el desarrollo del controlador posteriormente explicado. Lo explicado en este subapartado se puede encontrar referenciado en [1].

En una gran cantidad de aplicaciones las referencias vienen dadas en el espacio de operaciones cartesiano y la cinemática inversa es utilizada para transformarlas en trayectorias articulares y así poder usarlas, pero este esquema presentado a continuación permite trabajar directamente con las referencias en el estado de operaciones.

Todos los métodos suelen utilizar la cinemática inversa para computar las trayectorias deseadas y llevarlas al espacio articular, pero en este esquema de manera opuesta se utilizará la cinemática directa para llevar las coordenadas articulares del robot en cada momento a coordenadas cartesianas y poder reducir el error así en el espacio de operaciones. Normalmente este tipo de esquemas se utilizan siempre en casos en los que el efector final del manipulador esté restringido por el medio y este sea motivo de preocupación. En ciertos casos también es recomendable controlar las fuerzas que actúan sobre el efector final del manipulador.

Existen dos esquemas de control en espacio de operaciones, que son los más conocidos. El primero sería el *Esquema de control con el jacobiano inverso*, este esquema compara la posición cartesiana deseada dada en la trayectoria con la posición cartesiana del efector final, de esta comparación se debe obtener un error lo bastante pequeño que al ser multiplicado por la Jacobiana inversa del robot dará como resultado los pares articulares. Entonces las fuerzas generalizadas pueden ser computadas en base a esta desviación con una matriz de feedback de ganancias adecuada. El resultado general de este esquema es que de manera intuitiva se comporta como un sistema mecánico generalizado cuya constante de rigidez viene determinada por la matriz de feedback de ganancias de manera que el sistema cumple la función de reducir el error en valor articular hasta cero.

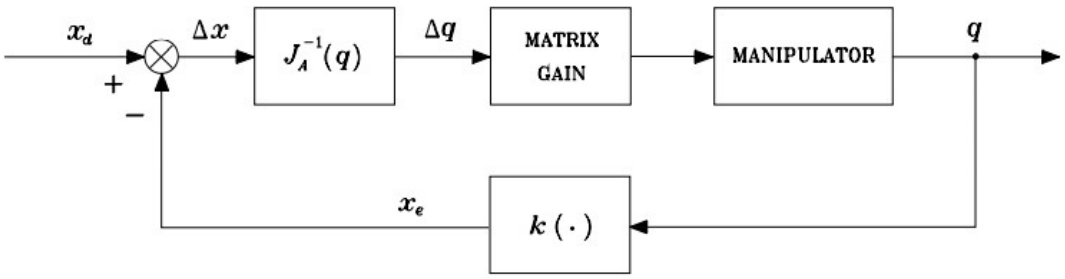

Figura 2.3: Esquema de bloques del control con la jacobiana inversa [1]

Donde x_d, es la posición deseada en cada momento. Δ_x es el error cartesiano calculado, utilizando la posición cartesiana actual dada por x_e. También se encuentra en el esquema $J_A^{-1}(q)$, que hace referencia al Jacobiano inverso del robot y el error en articular que viene dado por Δ_q. Y finalmente cabe destacar que la variable q hace referencia a la posición articular.

El segundo esquema de control conceptualmente análogo, es el llamado *Esquema de control de la Jacobiana traspuesta*. En este caso el error en cartesiano es tratado primero por la matriz de ganancias, la salida del este bloque puede ser considerada como una fuerza elástica generalizada de un muelle. El objetivo final de este tipo de control es llevar el error en cartesiano, el espacio de operaciones, hasta cero. Las fuerzas en el espacio de operaciones deben de ser transformadas en las fuerzas generalizadas del espacio articular a través de la traspuesta de la jacobiana para asegurar el comportamiento explicado.

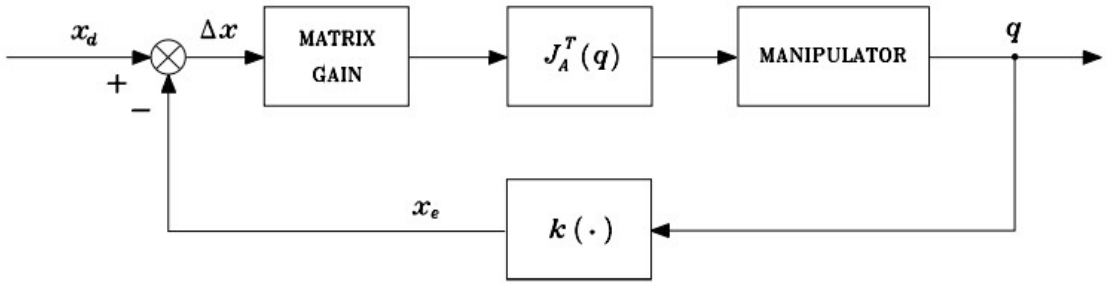

Figura 2.4: Esquema de bloques del control con la jacobiana traspuesta [1]

Puesto que las variables utilizadas en este esquema son exactamente iguales que a las comentadas en el esquema anterior no se considera necesaria su explicación. La única diferencia es el cambio de la Jacobiana inversa por la Jacobiana traspuesta $J_A^T(q)$,

2.2. Historia de los robots espaciales

La exploración espacial de nuestro sistema solar y galaxias lejanas en los rincones más apartados del universo es importante para conseguir desarrollar nueva tecnología y ciencia de más alto nivel y además de buscar la respuesta a tantas preguntas fundamentales que tenemos los humanos sobre el universo, incluyendo desde la formación, el origen de la tierra, la evolución de la vida y sobre todo la existencia de vida fuera de la tierra en cualquier otra parte del universo.

La robótica espacial juega y ha jugado hasta ahora un papel crucial en el futuro de la exploración espacial ya que permite la utilización de máquinas específicas para cada misión y que son capaces de sobrevivir las condiciones tan extremas del espacio mientras pueden realizar diferentes tareas fácilmente como exploración, construcción, mantenimiento y diferentes servicios.

Los robots siempre han aumentado las capacidades de los humanos para realizar cualquier tarea, en este caso el espacio no iba a ser una excepción. Los robots son capaces de expandir las capacidades humanas gracias a su capacidad de sobrevivir en entornos tan dañinos y peligrosos para el ser humano. Durante la historia del espacio se han desarrollado dos tipos diferentes de robots espaciales: Aquellos que trabajan en órbita y los que se dedican a la exploración de diferentes cuerpos no terrestres.

Los primeros comentados se encargan de tareas relacionadas con la reparación de satélites, construcción y unión de diferentes telescopios, captura y devolución de asteroides y ayuda en diferentes investigaciones científicas. El trabajo actual se encuentra más centrado en este tipo de robots.

Desde el final de la década de los cincuenta, se ha llevado a cabo una serie de misiones espaciales que han permitido aumentar los conocimientos y tecnologías anteriormente comentados. Desde el principio de los años sesenta la exploración espacial se ha centrado en mandar humanos a la órbita terrestre o incluso a la Luna [3]. Esta situación se convirtió en una especie de carrera por la conquista del espacio entre Estados Unidos y la unión soviética, que se vio potenciada por la situación de conflicto entre estos dos países durante la conocida *Guerra Fría*.

Todas estas primeras misiones eran cruciales para entender el ambiente al que iban a mandar a los diferentes astronautas. La primera misión en la que se utilizó un robot espacial fue en 1967 en la nave Surveyor 3 que aterrizó con éxito en la Luna. Esta misión utilizó un robot manipulador "*pala*" para recoger elementos de la superficie de la luna. Después de esta misión durante los años 70 se realizaron más misiones que utilizaban robots manipuladores con herramientas como taladros y similares.

El buen funcionamiento de estas misiones utilizando manipuladores produjo que se extendiera su uso a la gran mayoría de misiones orbitales. En cambio, para misiones planetarias, aquellas en las que un robot aterriza en un cuerpo no terrestre y se dedica a explorar el espacio se ha utilizado siempre un famoso robot móvil extremadamente robusto del cual se han diseñado nuevas versiones para cada misión con la última tecnología punta del momento, el robot Rover. Básicamente todo lo que la humanidad sabe sobre Marte y sobre la luna ha sido gracias a este robot móvil.

El futuro de la robótica espacial estará conquistado por estos dos tipos de robots por las capacidades que ofrecen y su gran ayuda a los astronautas humanos. También existe la posibilidad de que China está desarrollando su propia estación espacial la cual será lanzada durante la próxima década y pretende aumentar las capacidades y cantidad de los experimentos llevados a cabo en el espacio [3].

2.3. Propiedades y caracteristicas de los robots espaciales

Básicamente existen dos atributos esenciales para que una *astronave* sea considerada un robot espacial. Estos dos atributos son la locomoción y la autonomía [3]. Un robot espacial debe tener locomoción (o movilidad) para manipular elementos, agarrarlos, moverse, utilizar taladros y coger muestras. Además, dependiendo de la naturaleza de la misión los niveles de autonomía que se le otorgan a los robots suelen variar importantemente. Siendo normal que la capacidad de teleoperar a un robot disminuya a medida que el robot se aleja de la órbita terrestre.

Además de estas dos capacidades es considerado que un robot espacial debe contar con las otras muchas características clave, seguidamente se procederá a comentar y repasar todas las características consideradas en [4]:

- Manipulación: Aunque la manipulación es una característica básica en el campo de la robótica, la microgravedad del ambiente orbital requiere que se ponga especial atención en el movimiento dinámico del brazo manipulador y los objetos involucrados. Las reacciones debidas a la dinámica afectan la base del cuerpo, la dinámica del impacto cuando la mano o pinza robótica hace contacto con un objeto es un gran problema que se debe superar con esta característica.

- Movilidad (Locomoción): Esta característica es especialmente importante en los robots de exploración (Rover) ya que deben viajar por superficies totalmente desconocidas. Por lo que con esta característica se deben añadir todas las funcionalidades para que el robot pueda soportar las dinámicas de las superficies de otros planetas o satélites. Sin embargo, esta característica también es aplicable a robots astronautas que deban moverse por el mismo entorno que los astronautas humanos [5].

- Teleoperación y Autonomía: Hay un tiempo de retraso que no es para nada despreciable cuando se trabaja a distancia con robots en el espacio. Si el robot se encuentra dentro de la órbita terrestre el retraso puede estar cerca de unos pocos segundos dependiendo la posición y la órbita, pero nunca será un número extremadamente grande. En cambio, cuando se habla de misiones planetarias el retraso existente puede llegar a decenas de minutos fácilmente. Por ello la tecnología de teleoperación combinada con una gran autonomía es una característica extremadamente importante.

- Adaptación al medio extremo: Además del ambiente en microgravedad que afecta la dinámica de movimiento del robot existen otros muchos problemas a los que se debe hacer frente en el espacio exterior para poder aplicar soluciones de ingeniería. Estos incluyen problemas como temperaturas extremas, el vacío del espacio y las altas presiones, las atmósferas corrosivas, estar continuamente sometidos a radiación

ionizante, el polvo espacial y más problemas similares. Que los robots cuenten con esta característica es muy importante para poder aplicar todos los conocimientos de ingeniería en el espacio.

- Versatilidad: Esta característica es la meta final en todos los diseños de robots espaciales y especialmente remarcada en aplicaciones espaciales. Debido a la naturaleza de las misiones espaciales un robot debe de ser capaz de realizar todas sus tareas de manera individual utilizando los recursos que se le han dado, ya que en la mayoría de los casos no habrá ninguna persona para facilitarle las tareas. Por eso, esta característica es muy necesaria para que los robots puedan adaptarse al medio del espacio exterior

Además de estas características generales, hablando sobre las características específicas de los robots espaciales trabajando en la estación espacial o en relación con la misma, siempre es deseable que los robots tengan un aspecto y un modelado humanoide [5]. Esto es debido a que la estación espacial está diseñada para ser utilizada únicamente por humanos, cuando se empezó a construir esta no se pensó en la posibilidad de enviar robots al espacio, por lo que todos los sistemas de control existen están diseñados para que se puedan utilizar cómodamente por humanos ya sean botones, interruptores, joysticks, asas dentro de la estación para moverse, etc. Por esta razón cuando se ha decidido diseñar un robot para ser enviado a la estación espacial y poder trabajar con los astronautas se ha centrado el diseño en los robots humanoides, este es el caso del robot Robotnaut enviado por la NASA a la estación espacial y sobre el cual se comentarán unas características en el apartado siguiente.

2.3.1. Caso de estudio Robonaut

En este subapartado se ha decidido explicar el robot Robonaut por sus características en común con este proyecto y además para mostrar las razones que han influido en las decisiones tomadas sobre el modelo del robot diseñado para este trabajo.

Robonaut es un humanoide diestro que fue diseñado por la NASA como una máquina que puede asistir a los astronautas humanos realizando tareas de manipulación diestras usando unas manos sofisticadas similares a las de los humanos con múltiples grados de libertad. El

trabajo con la primera versión de Robonaut comenzó en el año 1997 y el primer Robonaut salió a la luz en 2002 y se llamó Robonaut 1 (R1). Durante 2006, se realizaron numerosos experimentos en este robot para demostrar que podía ser un asistente valido en la estación espacial internacional. La segunda generación de este robot se llamó Robonaut 2 (R2), fue sacada a la luz durante el año 2010 y esta versión contaba con una tecnología mucho más avanzada que la versión anterior por eso se convirtió en el primer robot humanoide en viajar a bordo de la estación espacial internacional en febrero de 2011, y donde todavía sigue asistiendo a los astronautas en diversas tareas. La última versión R2 se puede observar en la figura 2.5.

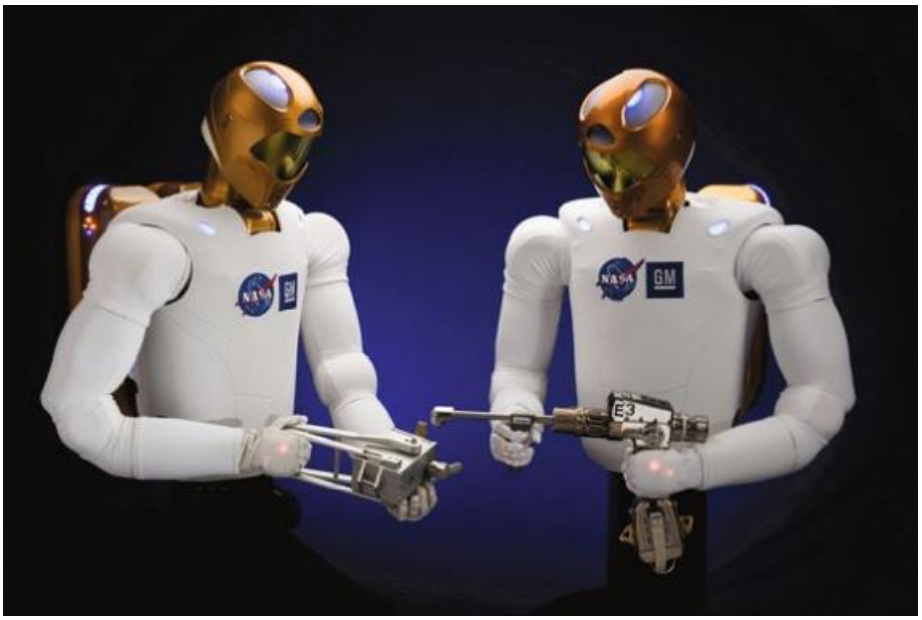

Figura 2.5: Robonaut 2 [4]

Las ventajas de utilizar un robot de apariencia humanoide son muchas. Empezando porque el robot puede adaptarse al espacio de trabajo y a las herramientas utilizadas por los astronautas. Esto no solo aumenta la eficiencia, sino que también permite que no sea necesario realizar modificaciones en la estación espacial para adaptar el uso de robots, lo que se trataría de una dificultad mucho mayor y coste económico enorme.

El robot Robonaut cuenta con 42 GDL (Grados de libertad) en total, cada brazo cuenta con 7 GDL y cada mano tiene dedos de 12 GDL. Tiene más de 350 sensores en total que se usan para el control basado en fuerza/torque para una manipulación diestra y también para

comportamientos seguros. Todas estas características se han intentado replicar o utilizar en el robot utilizado para el desarrollo de este trabajo.

Aunque el rol de Robonaut en la estación espacial está limitado a los experimentos dentro del laboratorio *Destiny* se han diseñado y establecido una multitud de planes sobre este robot para futuras misiones, algunas de estas podrían ser tales como: incorporarle una parte inferior del cuerpo con la intención de que se desplace dentro de la estación espacial, añadirle mejoras con el fin de que pueda ayudar a los astronautas en tareas fuera de la estación espacial o incluso combinarlo con alguna extensión móvil ya sean piernas o ruedas para que pueda ayudar en próximas misiones planetarias.

El concepto de robots de servicio o los robots de vuelo libre ha sido estudiado por muchos años, pero aun así todavía se han podido realizas unos pocos y limitados vuelos de validación orbita hasta la fecha. Se necesitarán más avances tecnológicos para poder seguir desarrollando más a fondo este campo, más información y detalles sobre este tema también pueden ser encontrados a lo largo de [4] y [5].

2.4. Controladores Visuales para robots orbitales

En este apartado se presentarán algunas soluciones del estado del arte en cuanto al control de robots espaciales. Con el fin de otorgar cada vez más autonomía a los robots para que puedan ser capaces de sustituir a los humanos en sus tareas en ambientes tan peligrosos como el espacio exterior se han dedicado una gran parte de recursos al estudio del uso de la visión por computador en los sistemas robóticos espaciales.

La visión por computador es capaz de proveer información al robot sobre el estado actual del medio a tiempo real, lo que es necesario para llevar a cabo cualquier tarea de manera semi o completamente autónoma. La visión por computador es una combinación de procesamiento de imágenes combinado con reconocimiento de patrones y análisis de imágenes. Es capaz de reconocer y determinar la orientación y posición de un objeto de interés que se encuentre en una imagen.

Varias aplicaciones presentadas en [6] han sido actualmente utilizadas en misiones espaciales, pero para que estas puedan ser llevadas al espacio exterior e integradas en un

sistema robótico deben pasar una gran cantidad de pruebas de robustez y ser aceptadas por un comité de expertos en visión por computador. A continuación, se procederá a explicar dos métodos planteados en [7] que cumplen con estas características y que se utilizan para el control de robots en aplicaciones como guiado de naves y guiado de robots de vuelo libre.

2.4.1. Guiado de naves espaciales utilizando visión por computador

La basura espacial se está convirtiendo en uno de los problemas que más preocupa a la comunidad espacial por su repercusión en el presente y el futuro de las misiones espaciales. Estos objetos considerados como basura espacial ocupan un espacio importante en la órbita de otros satélites o naves, lo que produce que exista un alto riesgo de colisión entre objetos.

Diferentes estrategias para la eliminación de basura y la definición y desarrollo de soluciones para realizar tareas de mantenimiento y rescate de satélites pueden verse en [8] y [9] respectivamente. Así mismo, el esquema realizado para la captura de grandes objetos indeseados en una órbita está bastante bien establecido [10], básicamente el perseguidor tiene que:

- Llegar a la misma órbita que el objeto
- Realizar las maniobras orbitales necesarias para aproximarse al objetivo
- Sincronizar su movimiento con el del objetivo
- Realizar las maniobras de unión necesarias
- Permitir el servicio en órbita y las maniobras de agarre

Para este bien establecido esquema la mejor opción es una solución basada en la visión por computador. Esta se incluiría en la etapa de acercamiento y sincronización del movimiento de la nave que intenta eliminar la basura. Diferentes ejemplos de sistemas de visión por computador se han diseñado para cumplir con esta tarea como puede ser el sistema Automático de transferencia de vehículos [11], actualmente en uso. Es capaz de reconstruir la posición relativa del vehículo perseguidor con respecto a la estación espacial internacional identificando las características visuales del objeto unido a la estación espacial.

Este tipo de métodos basados en visión por computador han sido probados y se ha demostrado su eficacia para dos elementos cooperativos, es decir que el elemento que va a ser recogido y enviado tiene una estructura de captura conocida y diseñada para poder unirse a otros elementos y además es un elemento capaz de comunicar su posición a la nave que realiza el acercamiento. Por esto mismo ha sido caso de estudio también la intención de extender estos métodos para elementos no cooperativos. La diferencia es que en este caso la posición del objetivo no puede ser comunicada en ningún momento y la estructura del objetivo a recoger es totalmente desconocida.

En estos sistemas de navegación el sistema debe abordar el problema mediante los sensores de abordo y algoritmos de procesamiento de imágenes a tiempo real. Además, a la hora de efectuar la maniobra de aproximación el sistema perseguidor debe ser consciente de la topología del objetivo y de sus características para poder acercarse lo más eficazmente posible al objetivo.

Para este tipo de maniobras la viabilidad de un sistema de navegación basado en cámaras monoculares ha sido verificada con éxito mediante el uso de dos filtros de Kalman como se puede ver en [12]. También se han diseñado aproximaciones en las que sensores LiDAR han sido usados para realizar este tipo de control de navegación [13].

2.4.2. Guiado de robots flotantes utilizando visión por computador

La investigación y desarrollo sobre los robots manipuladores que actúen en operaciones espaciales en satélites se ha incrementado en los últimos años. Actualmente, la utilización de robots manipuladores montados en satélites se puede ver categorizada en 6 tipos [14]:

- Montaje, mantenimiento y reparación.
- Despliegue, liberación y recuperación de naves espaciales.
- Apoyo en la actividad extra vehicular.
- Inspección.
- Repostaje.
- Cooperación multibrazo.

Los robots espaciales utilizados en órbita pueden ser de dos tipos básicamente, los robots de vuelo libre cuya posición de la base se puede controlar y mover libremente y los robots flotantes en los cuales no se puede controlar la posición de la base. Estos últimos serán los tratados en este subapartado.

En los robots flotantes la manera de actuar normalmente se basa en control en imagen obteniendo las características de una imagen obtenida por una cámara montada en el extremo del manipulador (Sistema Ojo en mano). El esquema de actuación para estas situaciones se basa en 4 etapas:

- Observar y planificar.
- Aproximación final.
- Impacto y captura.
- Estabilización post captura.

El controlador utilizado en este tipo de robots tiene las capacidades de usar la cinemática y dinámica del robot para poder ser capaz incluso de seguir una trayectoria con respecto al objeto observado sin necesidad de realizar un calibrado preciso ni un modelado exhaustivo. El controlador define la ley de control en el espacio de la imagen, esto significa que el controlador estará diseñado para reducir el error en la posición de las características de la imagen obtenida, no de la posición del brazo. Pero ya que la imagen obtenida y la posición del brazo están unidas, al reducir un error llevará a la reducción de la posición del brazo. Además, cabe mencionar que el controlador de este tipo presentado en [7] que cumple con las características mencionadas anteriormente, también proporciona directamente los pares que deben actuar en cada articulación y es capaz teniendo en cuenta la dinámica del robot dentro del control de imagen puede ser capaz de calcular los efectos producidos en la posición de la base durante el seguimiento de la trayectoria.

Además del controlador comentado también existen otros muchos enfoques para afrontar este problema de control como puede ser por ejemplo en [15] se presenta un enfoque clásico basado en imágenes, basado en posición y de conmutación para la captura de satélites de manera autónoma y utilizando un manipulador a bordo con cámaras binoculares con un sistema ojo en mano y sin considerar la dinámica del sistema. Como este otros muchos son presentados en [7].

3. Metodología: Bases teóricas y técnicas del desarrollo

Durante este apartado se va a explicar cuál ha sido el proceso durante la realización del trabajo, así como cuáles han sido las tecnologías utilizadas en cada parte del proceso y cómo funcionan estas. También se explicarán los controladores diseñados en el trabajo, así como sus bases matemáticas y las razones detrás de el porque se ha decidido tomar cada decisión.

Primeramente, se comentarán las características del robot utilizado, así como las modificaciones que se han llevado a cabo en su diseño para adaptarlo de la mejor manera posible al ambiente de trabajo en la estación espacial internacional o el espacio exterior. Seguidamente también se expondrá el procedimiento para realizar una simulación en Gazebo junto con la estación espacial.

Una vez se hayan explicado estos apartados, se seguirá explicando las tecnologías utilizadas para programar las funcionalidades del robot, así como para realizar la implementación de los controladores. Se comentarán las dos librerías utilizadas más importantes para el correcto funcionamiento del trabajo y finalmente se expondrán los dos controladores que mejores resultados han dado y su funcionamiento.

3.1. Diseño del astronauta robótico

El diseño principal del robot es el proporcionado por la empresa española PAL Robotics [16], ya que el robot en el que se basa el trabajo es el robot humanoide TALOS de esta misma empresa. Este robot humanoide en la totalidad de su cuerpo cuenta con 32 GDL, de los cuales 18 GDL pertenecen a la parte superior del cuerpo. Este robot fue elegido por las amplias características que ofrece tales como son los sensores de torque y la posibilidad de controlar sus articulaciones por par. También cuenta con una cámara en su cabeza que permite la posibilidad de implementar soluciones basadas en visión por computador. Y finalmente como se comentaba en el apartado del estado del arte también se tiene en cuenta la importancia de que se trate de un robot humanoide hace que sea más fácil la adaptabilidad al medio de la estación espacial internacional diseñada en su origen para el trabajo humano.

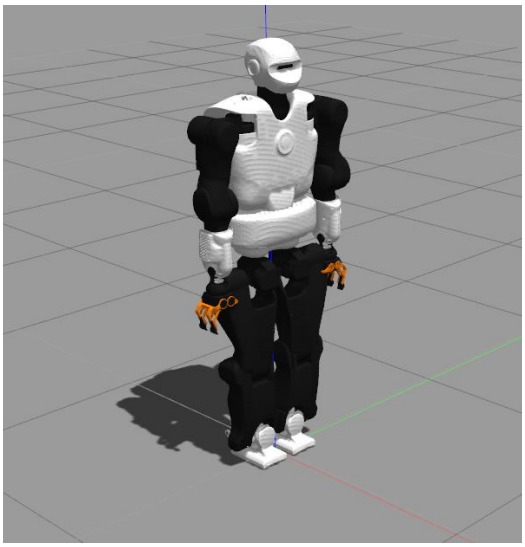

Figura 3.1: Robot Talos original. **Fuente:** Elaboración propia.

Para adaptar todavía más al ambiente espacial el robot TALOS se modificaron los archivos .urdf, los cuales guardan las características (tanto cinemáticas como dinámicas) del modelo del robot, proporcionados libremente por la empresa PAL Robotics. Mediante la modificación de estos archivos se consiguió eliminar las piernas del robot y se decidió que se le añadiría una mochila propulsora para dotar al robot de la capacidad de moverse libremente en el espacio. En la versión final del trabajo esta mochila propulsora se ha añadido como un elemento más del torso unido mediante una articulación fija, pero se trata de un elemento totalmente visual ya que no era el caso de este trabajo el desarrollar el sistema de movimiento del robot mediante propulsores. La mochila se obtuvo de una página de modelos 3D totalmente gratuita [17] en la que los usuarios publican sus modelos libremente visibles para todo el mundo, aun así, se realizaron algunos cambios en el modelo para adaptar su tamaño, su forma y sus características al torso del robot utilizado.

Finalmente, para que el robot fuera capaz de realizar sus tareas en la estación espacial se decidió cambiar las pinzas que normalmente utiliza este robot por unas manos robóticas más similares a las manos humanas, por razones de compatibilidad y adaptabilidad se decidió utilizar las manos Hey-5 de la empresa PAL Robotics [16], estas manos son manos con 5 dedos tal y como las humanas que poseen 19 GDL cada una. De manera que al añadir estas manos se ha dotado al robot de capacidad para trabajar más cómodamente en la estación espacial como lo haría un humano. En total el robot modificado para el uso en este trabajo quedaría como se puede ver en la figura 3.2 y contaría con un total de 37 GDL.

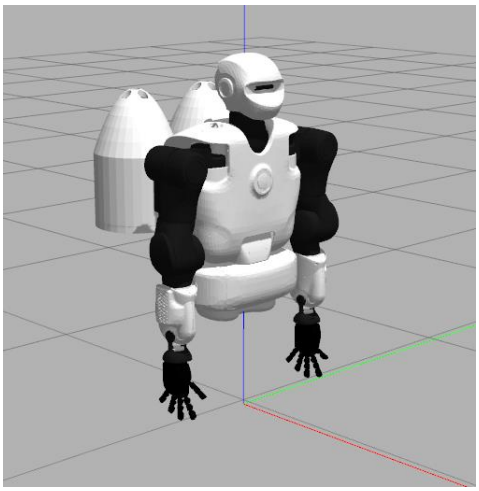

Figura 3.2: Robot Talos Astronauta. **Fuente:** Elaboración propia.

Además, cabe mencionar que durante todo el trabajo al robot modificado y utilizado para la simulación se le pasará a llamar Talos Astronauta. En el siguiente subapartado se pretende explicar de manera más detallada el funcionamiento de los archivos URDF para ayudar al lector a comprender mejor los cambios físicos que se han comentado en este apartado y como se han realizado.

3.1.1. Archivos URDF con Gazebo

Los archivos URDF son los archivos necesarios para que gazebo sea capaz de simular todos los aspectos de un robot. Por eso mismo su nombre es Unified Robot Description Format cuyas siglas son URDF. Este tipo de archivos se encarga de describir todas las características del robot que se quiera simular ya sean características cinemáticas o dinámicas.

Para describir todas las características del robot, en estos archivos se debe proporcionar una descripción de todas las articulaciones y eslabones que forman el robot y se deben especificar en forma de una estructura de árbol, de manera que cada elemento del robot estará conectado en cierta manera con un elemento anterior (padre) y un elemento siguiente (hijo), excepto para los elementos extremos (hojas del árbol). Con el fin de que se pueda entender el concepto de la estructura en árbol formada en los archivos URDF en la figura 3.3 se puede ver un grafo de la estructura del Talos Astronauta creado para este proyecto. En esta figura cada nodo hace referencia a un eslabón del robot, las aristas que unen los nodos muestran el canal de comunicación entre los sistemas de referencias de cada eslabón y finalmente cabe

destacar que los dos grandes árboles de las dos esquinas inferiores son cada una de las manos otorgadas al robot mientras que la columna central es el cuerpo superior del robot.

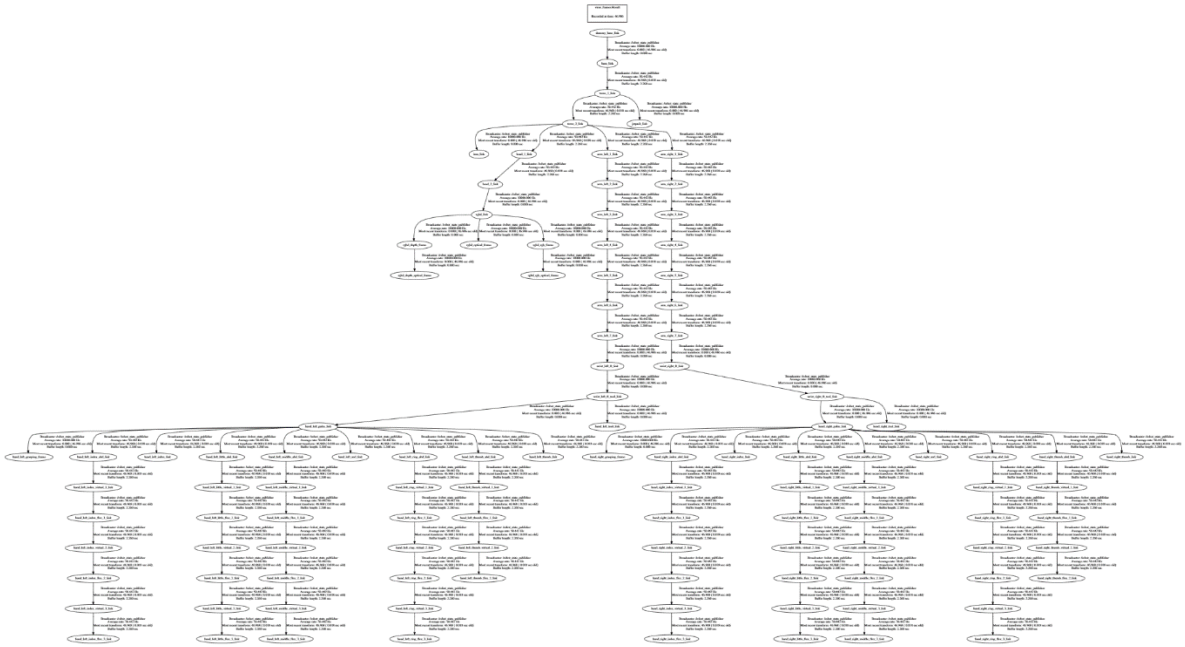

Figura 3.3: Estructura cinemática del Talos Astronauta. **Fuente:** Elaboración propia.

En el caso de tener articulaciones actuadas se debe declarar una interfaz llamada *transmisiones* para cada una de ellas, esta interfaz representa lo que sería una transmisión de un motor real, se encarga de traducir el tipo de dato recibido a la corriente eléctrica necesaria para mover el motor. Los tipos de transmisiones existentes en ROS y ROS Control trabajan con datos de posición, velocidad y torques.

A la hora de declarar los eslabones y articulaciones necesarios es muy importante tener en cuenta las relaciones cinemáticas que existen entre cada elemento de manera relativa. Este tipo de archivos nos permite declarar todas las características dinámicas como la masa, inercias, rozamientos, etc. También permite que se incluyan modelos CAD para cada componente que se quiera si es que se tiene un modelo de este tipo. Estas características nos permitirán realizar soluciones lo más próximas posibles a la realidad, lo que hará que el cambio entre la simulación y la realidad no sea excesivamente grande. Para un mejor entendimiento del funcionamiento de este tipo de archivos se puede ver un esquema del mismo en la figura 3.4.

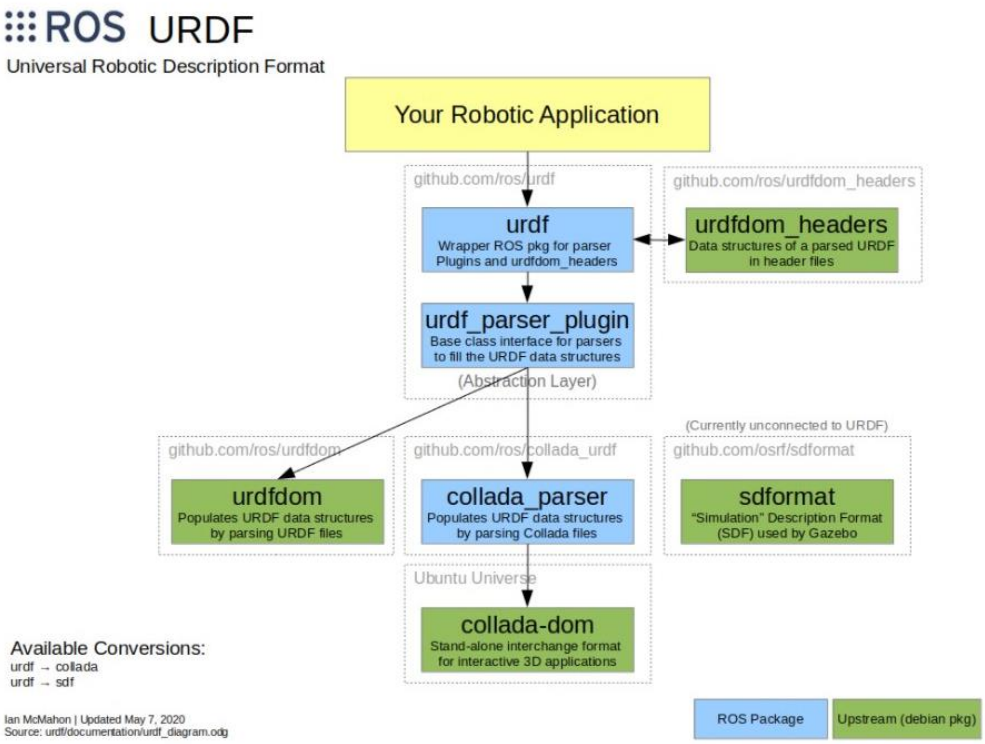

Figura 3.4: Esquema de funcionamiento del tipo de archivos URDF. **Fuente:** http://wiki.ros.org/urdf

En este trabajo este tipo de archivo es totalmente imprescindible ya que como se explicará más adelante las librerías utilizadas se encargan de leer estos archivos para obtener el modelo dinámico del robot.

3.2. Diseño de la simulación en Gazebo

Para poder adaptar un diseño de la estación espacial internacional en una simulación de gazebo se hizo uso del modelo ofrecido en la página oficial de la NASA [18]. Este archivo está público y se puede descargar, pero el formato disponible no es compatible con Gazebo. Para poder adaptar el modelo de estación espacial internacional a Gazebo se utilizó la aplicación Blender [19] que es un programa informático multiplataforma, dedicado especialmente al modelado, iluminación, renderizado, la animación y creación de gráficos tridimensionales.

El archivo descargado de la página de la NASA es un archivo .gLTF el cual contiene una gran cantidad de información sobre las diferentes partes que componen el modelo. Mediante el programa de Blender se modificó el modelo 3D de la estación espacial para que toda la masa de este se concentrará en el centro de masas de la estación espacial así reducir la información sobre el modelo para que el motor de físicas de Gazebo sea capaz de soportar la simulación. El modelo 3D de la estación espacial se puede ver en la figura 3.5.

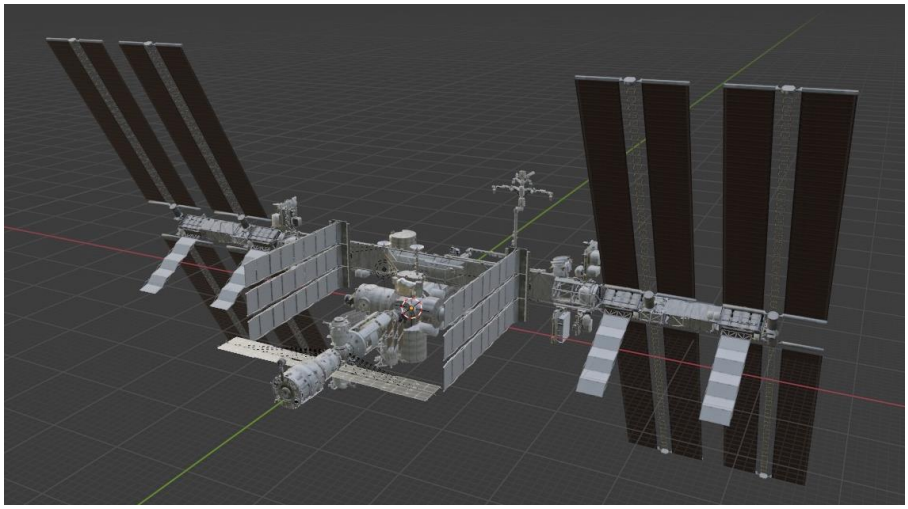

Figura 3.5: Modelo 3D de la estación espacial internacional visualizado en Blender.
Fuente: Elaboración Propia

Además de realizar este cambio también se decidió añadir una pequeña modificación a la estación espacial en la zona que el robot iba a trabajar en la simulación. Esta extensión fue añadir un código aruco para que el robot pueda ser capaz de localizar su base con respecto a este código aruco como será explicado más adelante junto con el controlador. En la figura 3.6 se puede ver el código aruco añadido a la estación espacial y también se puede observar un modelo más fino de asa, modificado para que el robot pueda realizar un agarre mejor y más consistente.

Figura 3.6: Código Aruco añadido en una zona de la estación espacial internacional.

Fuente: Elaboración propia.

Una vez realizadas todas las modificaciones sobre el modelo 3D, se debía exportar el archivo en un formato que Gazebo pudiera leer y procesar, por eso se exportó el modelo en un archivo collada cuya extensión es .dae. Se creó un directorio para guardar el modelo y que gazebo pueda entender, este directorio tiene la siguiente estructura:

- Carpeta *textures*: Guarda las imágenes de las texturas del modelo.
- Carpeta *meshes*: Guarda el archivo .dae con la información del modelo.
- Archivo *model.sdf*: archivo xml con la información sobre cuál debe ser el modelo visual y de colisiones utilizado por gazebo.
- Archivo *model.config:* Guarda la información sobre el creador del modelo y el nombre del modelo.

El directorio que contenga estos archivos deberá ser incluido en la carpeta ./gazebo/models (que se crea en cualquier ordenador en el cual se encuentre gazebo instalado) para que el modelo pueda ser utilizado en una simulación de gazebo.

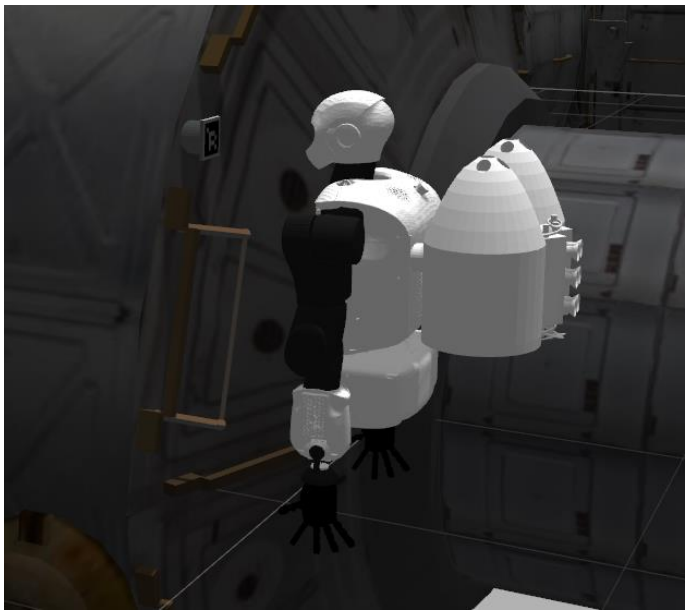

Figura 3.7: Talos Astronauta junto con la estación espacial en una simulación de Gazebo.
Fuente: Elaboración propia.

3.3. Tecnologías utilizadas

En este apartado se explicará de manera general cómo funcionan las tecnologías utilizadas para el desarrollo del trabajo. Para que se entiendan mejor todas las tecnologías se expondrán ciertos ejemplos basados en el trabajo realizado para facilitar al lector el poder seguir toda la explicación sin perderse. Primeramente, se hará una breve introducción a qué es ROS y qué posibilidades nos ofrece, seguidamente se hablará sobre las librerías de ROS Control y por último se explicará la utilidad de las librerías KDL y Pinocchio para el desarrollo de los controladores.

3.3.1. Framework de ROS

Robots Operating System o más comúnmente llamado ROS, es un middleware que está formado por una colección de frameworks centrado en el desarrollo de software para todos los diferentes tipos de robots. ROS se desarrolló originariamente en 2007 bajo el nombre de switchyard por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto del Robot con Inteligencia Artificial de Stanford. Desde 2008, el desarrollo

continuó principalmente en Willow Garage, un instituto de investigación robótica con más de veinte instituciones colaborando en un modelo de desarrollo federado.

A pesar de no ser un sistema operativo como tal, siempre ha sido considerado uno por las características que ofrece. Y es que ROS al igual que los sistemas operativos tiene la función de ofrecer al programador la abstracción del hardware, el control de dispositivos de bajo nivel, la implementación de funcionalidad de uso común, el paso de mensajes entre procesos y el mantenimiento de paquetes. Por estas razones son por las que siempre ha sido nombrado como un sistema operativo.

La estructura de ROS está basada en una estructura de grafos donde la comunicación entre nodos o procesos internos se realiza mediante publicación o subscripción a diferentes flujos de datos ya sean imágenes, estéreo, láser, control, actuador, contacto, etc. Esta estructura se puede ver en la figura 3.8.

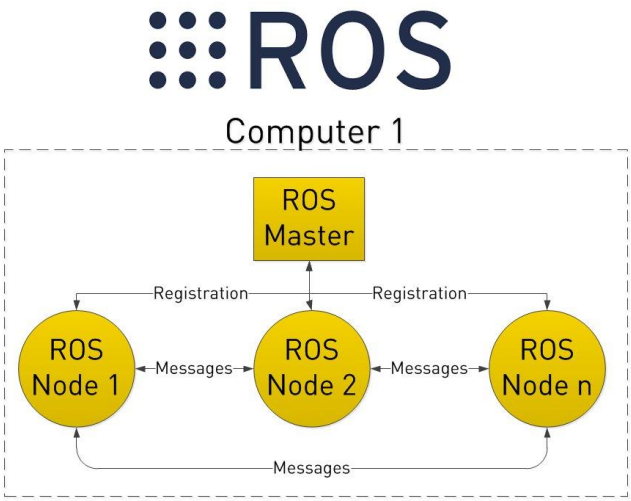

Figura 3.8: Estructura de comunicación de ROS. **Fuente:** http://dailyautomation.sk/04-ros-robot-operating-system-zakladne-principy/

ROS tiene dos partes básicas: la parte del sistema operativo (ROS) y los ros-pkg. Estos últimos se construyen de la contribución realizada por la comunidad de programadores que hay detrás de ROS. Mediante estos ros-pkg se ha conseguido adaptar la implementación de una gran multitud de algoritmos y funcionalidades tales como localización y mapeo simultáneo, planificación de trayectorias, percepción, simulación y el que explicará en el siguiente apartado Control de robot.

3.3.2. Ros con Ros control y su utilización con gazebo

Ros Control es el paquete que ha sido desarrollado por la comunidad de ROS para permitir el acceso de manera simple a los diferentes actuadores de los robots. Ros Control incluye una gran serie de paquetes imprescindibles que cumplen con distintas funcionalidades como una interfaz de control, administradores de controladores, transmisiones, interfaces hardware y una toolbox de control. El esquema general de ROS control se puede ver en la figura 3.9.

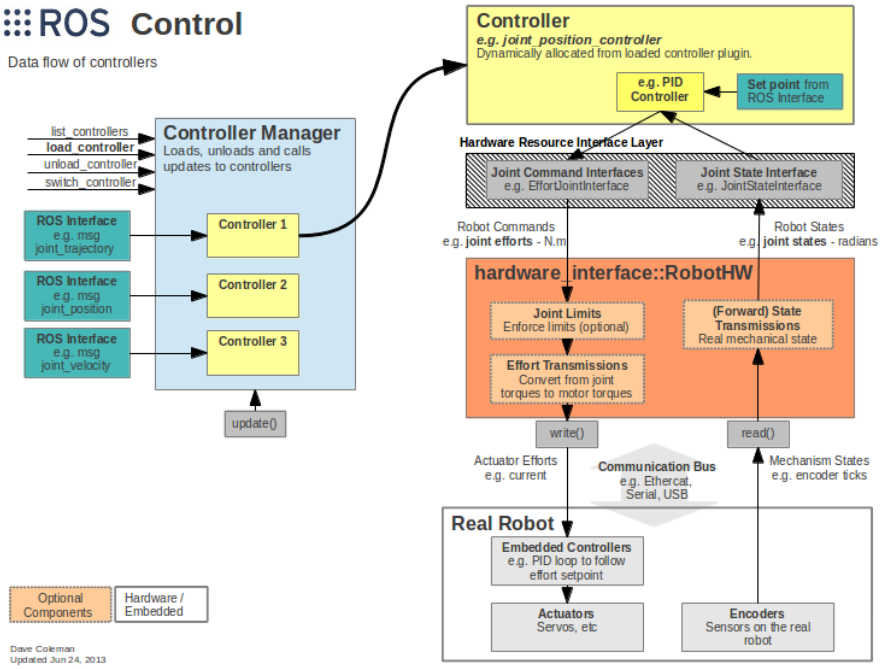

Figura 3.9: Diagramas de bloques de ROS Control. **Fuente:** http://wiki.ros.org/ros_control

Este esquema que vemos en la figura 3.9 nos permite diseñar nuestros controladores de una manera a más cómoda y se podría decir que a más alto nivel ya que esta funcionalidad se encarga de controlar directamente cada actuador, mientras que el programador puede centrarse en un control multiarticular sobre el robot de manera más sencilla. Ros Control se encarga de obtener los datos sobre el estado de las articulaciones y un punto destino como entrada desde el usuario o una tercera parte como se muestra en la figura 3.10 y envía los comandos adecuados a los actuadores como una salida. Con el fin de conseguir que el controlador llegue hasta la posición deseada la interfaz hardware implementa un control PID.

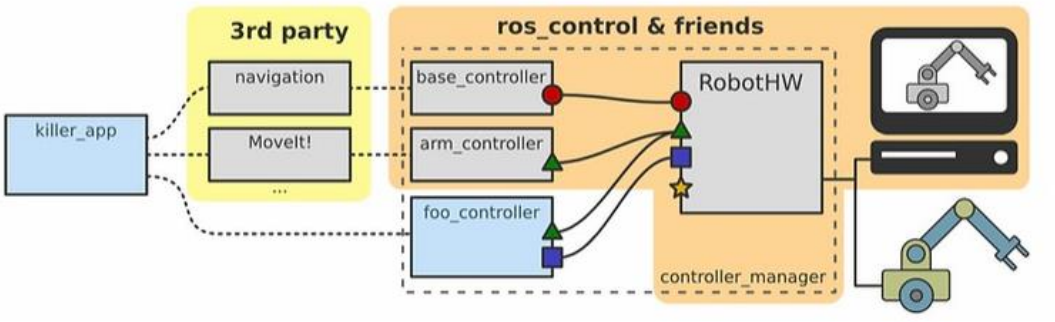

Figura 3.10: Esquema general del funcionamiento de ROS Control. **Fuente:** https://www.oreilly.com

Como ya se ha podido ver Ros Control tiene diversas características que son las que lo hacen realmente atractivo como puede ser: La capacidad de ejecutar bucles de frecuencias de hasta miles de hertzios, que le permite funcionar sin problemas a tiempo real. Su simple interfaz permite enviar las órdenes a los actuadores en diferentes referencias más entendibles a nivel humano. de Controla que los comandos enviados estén dentro los límites físicos del robot. Y para terminar un conjunto de controladores básicos listos para ser utilizados en cualquier robot.

Pero ciertamente es la capacidad de combinarse con el simulador de físicas de Gazebo lo que lo hace tan atractivo para cualquier proyecto. Esto permite que se puedan evaluar los controladores implementados sin la necesidad de tener un robot real, sin esta gran característica nada de este proyecto podría haber sido evaluado en lo más mínimo. En la siguiente figura 3.11 se puede observar la forma en la que se combinan ROS y Ros Control junto con el simulador Gazebo de una manera más esquemática para que sea más fácil de visualizar para el lector.

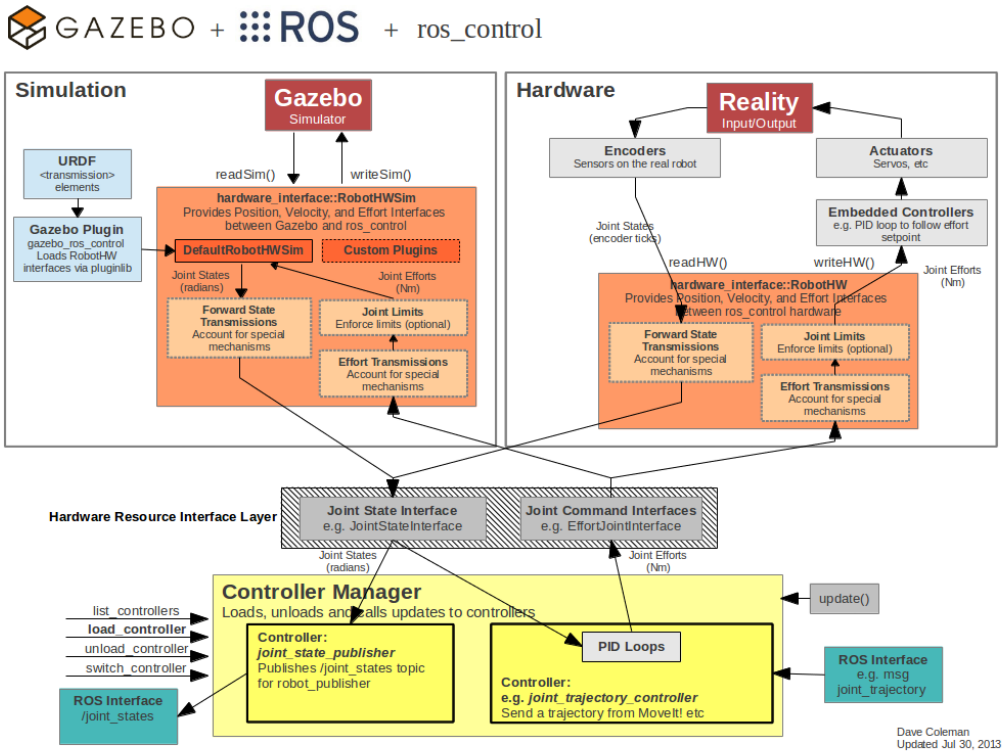

Figura 3.11: Esquema de conexión entre Ros, Ros Control y Gazebo. **Fuente:** http://gazebosim.org/ tutorials/?tut=ros_control

En este esquema se puede ver la importancia de los archivos URDFs, así como de un Plugin que es necesario añadir y configurar para que gazebo pueda simular los actuadores controlados, sin este Plugin sería imposible que gazebo moviera ninguna articulación del robot.

A continuación, se realizará una explicación y un tutorial genéricos de todo lo que hace falta para crear un controlador propio, para esto se utilizarán ejemplos tomados de los controladores realizados para este trabajo, ya que este proceso ha tomado una gran parte del trabajo completo.

- **Crear un paquete de ROS**: este debe tener como dependencias los siguientes paquetes: roscpp, pluginlib, controller_interface y hardware_interface.

- **Crear el código del controlador**: Para esto la única opción en cuanto a lenguaje de programación será utilizar C++. Para nuestro controlador se debe crear una clase la

cual debe de heredar de la clase *controller_interface* para obtener así una interfaz de control. Exactamente se debe heredar de la siguiente clase:

controller_interface::Controller<tipo_de_interfaz_hardware>

En el caso de los controladores de este proyecto se ha utilizado el tipo de interfaz hardware *hardware_interface::EffortJointInterface*, que proporciona Ros Control y nos permite utilizar los actuadores del robot utilizando comandos de par articular.

La clase creada debe contener al menos los siguientes métodos con las siguientes declaraciones:

- *bool init(hardware_interface::EffortJointInterface* hw, ros::NodeHandle& nd)*
- *void update(const ros::Time&, const ros::Duration&)*
- *void starting(const ros::Time&)*
- *void stopping(const ros::Time&)*

Estos métodos son los que hemos podido utilizar por herencia de la clase controller_interface, cada uno tiene una función específica y generalmente se programan para que cada uno realice una tarea como se explica a continuación.

- Función *Init ( )*: Normalmente se utiliza para cargar los parámetros específicos del controlador, para esto se hace uso de una gran característica de ROS que es el ros_param server. Tal y como indica su nombre, este es un servidor de parámetros en el cual se pueden declarar parámetros como si de variables se trataran. De manera que mediante el api de ros en esta función se realiza la tarea de buscar los parámetros que serán necesarios para configurar el robot, normalmente se recoge información sobre: la cadena cinemática, el efector final del robot, diferentes ganancias, tolerancias de errores etc. Todos estos parámetros se pueden cargar mediante la línea de comandos o utilizando un tipo de archivos que tienen como extensión .YAML. Estos archivos se cargan con el fin de poder encapsular y cargar todos los parámetros de cada controlador de manera más rápida. En la figura 3.12 se puede ver un ejemplo de la estructura de este tipo de archivos.

```
1.  right_arm_controller:
```

```
2.     type: "astronaut_controllers/CartesianTrajectoryController"
3.     end_effector_link: "wrist_right_ft_tool_link"
4.     base_link: "dummy_base_link"
5.     joints:
6.         - arm_right_1_joint
7.         - arm_right_2_joint
8.         - arm_right_3_joint
9.         - arm_right_4_joint
10.        - arm_right_5_joint
11.        - arm_right_6_joint
12.        - arm_right_7_joint
13.
14. goal_tolerance: 0.015
15. kp_value: 300.0
16. kv_value: 5.0
```

Figura 3.12: Archivo de configuración para el Controlador de trayectorias cartesianas multiarticular básico. **Fuente:** Elaboración Propia

- Función *starting ( )*: Indica el procedimiento que se debe llevar antes de que comience el bucle de control. Por ejemplo, en el caso del controlador cartesiano realizado en este trabajo se debe calcular la cinemática directa del brazo para así obtener la posición cartesiana del brazo y utilizar esta posición como punto final antes de que el robot reciba un objetivo final dado por el usuario. Mediante este proceso se consigue que el brazo se inicie en la posición inicial y se mantenga estable en esa posición. Si se estuviera realizando un control articular bastaría con pasar un comando de valor 0.0 a todas las articulaciones.

- Función *Update ( )*: En esta función se debe programar todo el comportamiento realizado en el bucle de control del controlador. Esto es muy variable según cada tipo de controlador.

- Función *stopping ( )*: De manera inversa a la función *starting* se debe indicar lo que se requiere que haga el controlador una vez que se haya detenido el mismo. Lo más normal es añadir medidas de seguridad para asegurar una parada controlada del robot.

- **Declarar el controlador**: Para que el código del controlador pueda ser compilado correctamente se debe crear un archivo de código C++ (*.cc o .cpp*) en el cual se deberá incluir la siguiente librería: *#include <pluginlib/class_list_macros.h>*. Además de esto se deberá declarar un objeto de la clase creada (la clase del

controlador) y por último se deberá declarar el controlador utilizando la siguiente función:

PLUGINLIB_EXPORT_CLASS(Objeto_de_la_clase,controller_interface::ControllerBase)

En este apartado se está haciendo cierto hincapié en el tema de realizar este paso en un archivo del tipo *.cc* o *.cpp* ya que lo más normal es declarar e implementar la clase en archivos de cabecera.

- **Crear el plugin del controlador:** Para que el controlador se pueda lanzar en un entorno de gazebo y el simulador sea capaz de simular las físicas relacionadas con los actuadores del robot manejados por Ros Control es necesario que se cree un plugin de gazebo. El siguiente plugin debe contener la información sobre el nombre que se le va a dar al controlador (elección del usuario), el tipo y tipo base que son los dos argumentos que recibe la función *PLUGINLIB_EXPORT_CLASS* explicada en el punto previo.

```
1.  <library>
2.     <class name="astronaut_controllers/CartesianTrajectoryController"
3.           type="position_controller::CartesianTrajectoryController"
4.           base_class_type="controller_interface::ControllerBase" >
5.          <description>
6.           The astronaut controller (Cartesian Controller) a target position is
    especified with the robot base_link reference. Also, a quintic trajectory is
    calculated and executed.
7.          </description>
8.     </class>
9.  </library>
```

Figura 3.13: Código del plugin para el Controlador de trayectorias cartesianas multiarticular básico. **Fuente:** Elaboración propia.

- **Modificar los archivos package.xml y CMakeLists.txt:** En cuanto a las modificaciones necesarias en el CMakeLists.txt es necesario compilar el objeto para que una librería sea creada, esto se puede realizar fácilmente añadiendo las siguientes 3 líneas de código. Que funcionan para que se compile el código como una librería.

```
1.  add_library(nombre_elegido_lib src/archivo.cpp)
2.  add_dependencies(nombre_elegido_lib ${${PROJECT_NAME}_EXPORTED_TARGETS}
    ${catkin_EXPORTED_TARGETS})
3.  target_link_libraries(nombre_elegido_lib ${catkin_LIBRARIES})
```

Figura 3.14: Ejemplo de código necesario para compilar el controlador de Ros Control. **Fuente:** Elaboración propia.

Además de los cambios necesarios en el CMakeLists.txt también será necesario añadir la información sobre el plugin creado al archivo package.xml, para esto también se muestra en la figura 3.15 un ejemplo de código que será igual en cualquier caso.

```
1.    <export>
2.      <!-- Other tools can request additional information be placed here -->
3.      <controller_interface plugin="${prefix}/plugin_name.xml"/>
4.    </export>
```

Figura 3.15: Líneas de código necesarias en el archivo package.xml para utilizar el plugin. **Fuente:** Elaboración Propia.

Cabe remarcar que en este archivo donde se puede leer "*plugin_name.xml*" se refiere al nombre que el usuario le haya dado al archivo creado con el código de la figura 3.12.

- **Crear el archivo YAML y Lunch:** Este paso es el último en el proceso, pero igualmente necesario que todos los demás. Lo primero será realizar el archivo con extensión .YAML el cual se utilizará para cargar los parámetros de controlador como bien se había explicado anteriormente. Un ejemplo de este archivo se puede ver en la figura 3.11. En este código es necesario escribir correctamente el nombre del controlador, este debe ser el mismo que el nombre que se haya declarado al crear el plugin en la figura 3.12. Los demás parámetros serán los que el programador requiera para el uso e implementación de su controlador.

  Por último, también será necesario un archivo con extensión .launch, este tipo de archivos es un tipo de archivos comúnmente utilizados en ROS y se utiliza para lanzar uno o varios nodos, especificando si se quiere o no parámetros y realizar el proceso de manera automática. En la figura 3.16 se puede ver un ejemplo de launch para lanzar un controlador de este trabajo.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
```

```
2.
3.  <launch>
4.
5.    <rosparam file="$(find nombre_paquete)/config/archive_configuración.yaml"
    command="load"/>
6.
7.    <node name="nombre_nodo” pkg="controller_manager" type="spawner" output="screen"
    args="right_arm_controller"/>
8.
9.  </launch>
10.
```

Figura 3.16: Archivo .launch para lanzar el Controlador de trayectorias cartesianas multiarticular básico. **Fuente:** Elaboración Propia.

En este archivo de ejemplo presentado se puede observar la manera en la que se cargan los parámetros de configuración del controlador y también la orden para cualquier controlador. Para esto en la línea 7 del código de la figura 3.16 se ve que se utiliza el paquete *controller_manager* para buscar en los parámetros cargados un controlador de nombre *right_arm_controller* que es el nombre que se le da en la primera línea de código de la figura 3.12.

Después de realizar todo este proceso el controlador estará listo para correr tanto en una simulación como un robot real. Es imprescindible realizar todos los pasos comentados de manera correcta para que el controlador pueda ser cargado y puesto en marcha sin problema.

3.3.3. Librerías utilizadas KDL y Pinocchio

En el trabajo presentado con este documento se ha hecho uso de varias librerías diferentes, pero las más importantes han sido la librería conocida como Kinematics and Dynamics Library (KDL) [17] y la Pinocchio [18].

En cuanto a la primera, la KDL, ha sido de vital importancia para obtener el modelo cinemático y diferencial del robot, un requisito que es imprescindible para realizar los cálculos necesarios para obtener los pares deseados de las articulaciones. Mediante el uso de esta librería se ha realizado el cálculo de la cinemática directa y el cálculo de la Jacobiana directa del robot, que nos permite obtener a partir del valor de velocidad articular el valor de velocidades cartesianas del extremo del robot. Además de estas funcionalidades, la KDL también ha sido utilizada para la creación de trayectorias, ya que nos permite interpolar

trayectorias entre los puntos que se elija por el usuario indicando las restricciones de velocidad y aceleración iniciales. Y por último esta librería también ofrece una gran cantidad de tipos de datos (puntos, vectores, frames) que permiten al programador realizar la tarea de manera más cómoda y mucho más organizada.

Por otra parte, también se ha hecho uso de la librería Pinocchio. En cambio, esta se ha utilizado para obtener el modelo dinámico del robot. Para ello la librería necesita leer un archivo URDF del mismo donde se especifique la descripción del robot con sus características dinámicas y cinemáticas. Para el caso de este trabajo se utilizó el mismo modelo del robot que en la simulación, pero se eliminaron del modelo las manos, de manera que así se podría utilizar el mismo código para el astronauta en caso de que llevará otro tipo de herramientas y además se evitaba un cálculo computacional excesivamente alto a causa de la gran cantidad de eslabones de los que se componen las manos y además despreciable para el movimiento del brazo.

Otra librería que debe ser remarcada en este apartado ya que ha sido de vital importancia para el proyecto ha sido la librería Eigen [19]. Eigen es una librería para llevar a cabo operaciones propias del álgebra lineal en programas de C++, en los cuales no existen tipos de datos como matrices o vectores en el sentido matemático. Internamente ambas librerías anteriores utilizan esta librería para realizar todos los cálculos matriciales necesarios, pero además también ha sido de gran utilidad para poder realizar todos los cálculos necesarios para obtener las matrices necesarias para implementar la ley de control en el controlador dinámico que será presentado en los apartados siguientes.

3.4. Controlador de trayectorias cartesianas multiarticular utilizando localización mediante visión por computador.

Para el desarrollo de este primer controlador se ha diseñado un controlador capaz de hacer que el efector final elegido siga una trayectoria en el espacio cartesiano o de operaciones y calculando con una ley de control el par necesario para todas las articulaciones del brazo robótico. Este controlador es capaz de detener el movimiento cuando se alcanza un umbral

de tolerancia de error con respecto del punto final de la trayectoria. Tanto esta característica, como las articulaciones que son controladas, como el efector final se elegirán desde el archivo de configuración .YAML explicado en los apartados anteriores.

El funcionamiento de este controlador se basa en que primeramente se crearán todos los modelos del robot, tanto cinemático como diferencial. El controlador explicado en este apartado es un controlador cinemático, por lo tanto, en este caso no se calculará exactamente el par que debe actuar sobre cada articulación, se calculará una aceleración que sirve como aproximación al par en la gran mayoría de casos, proporcionando un funcionamiento igual o mejor que utilizando par. En la función de *starting( )* que se ejecutará al inicio del controlador se utilizará el modelo cinemático del robot para calcular cual es la posición cartesiana del efector final del brazo robótico y así poder mandarla como primer objetivo a controlar durante el bucle de control. De esta manera se consigue que el robot mantenga el brazo estable en la posición inicial hasta que reciba cualquier otro objetivo al que mover el brazo.

Cabe destacar varios detalles sobre la inicialización de la cadena cinemática, primeramente en el archivo de configuración .YAML se debe indicar cuál será la base del robot, este eslabón elegido será la referencia para la cadena cinemática. Sin embargo, para que un eslabón sea la base de una cadena cinemática, la librería KDL no permite que este eslabón tenga inercia pero como es normal todos los eslabones del robot utilizado tienen su modelo dinámico con todos los detalles, por tanto ha sido necesario utilizar modificar el archivo URDF del robot Talos Astronauta para añadir junto al torso un eslabón nuevo que será llamado "*dummy_base_lin*k" se le ha dado este nombre para representar que es un eslabón que se encuentra en la base pero no tiene ninguna característica visual, de colisión, tamaño y viceversa por lo que no influirá en la simulación. Pero es necesario que se indique su existencia en el archivo .YAML para poder crear la cadena cinemática del robot.

Una vez explicado este detalle, se procederá a explicar el bucle de control principal. El bucle de control principal se ejecuta de manera continua con una frecuencia de 1KHz hasta que el controlador es desactivado. En este bucle se calcula la ley de control para conseguir que el efector final del robot siga una trayectoria cartesiana dada. Puesto que la trayectoria se debe realizar en el espacio cartesiano con el fin de tener un movimiento del brazo más restringido y controlado, el error será calculado con respecto a la posición cartesiana en cada momento

del efector final. Para esto se realizará una solución basada en el esquema de control que utiliza la jacobiana traspuesta del robot en la que primeramente al inicio de cada iteración se debe obtener la posición y velocidad cartesiana del efector final del robot, para ello se utiliza respectivamente el modelo cinemático directo y la jacobiana directa del mismo. En caso de que se haya recibido una trayectoria se sacará de esta los valores siguientes en función del tiempo de posición, velocidad y aceleración cartesianas que debe seguir el efector final. Con estos datos pasamos a calcular el error en la posición y la velocidad actual con respecto a la deseada que debería llevar en ese preciso momento de la trayectoria.

Estos cálculos serán imprescindibles para calcular la ley de control ya que la intención del esquema de control que se está utilizando será controlar estos errores hasta llevarlos a 0 en el espacio cartesiano. La ley de control obtiene las aceleraciones deseadas multiplicando la Jacobiana traspuesta del robot por la suma del error en posición y el error en velocidad que a su vez están siendo multiplicados por una constante proporcional y derivativa respectivamente. La siguiente ecuación 3.1 representa la ley de control descrita anteriormente.

$$\tau = J^T \cdot (Kp \cdot Ex \; + \; Kv \cdot Ev) \tag{3.1}$$

Donde $J^T \in \Re^{7x6}$ representa la matriz traspuesta de la Jacobiana del robot. Kp es la constante proporcional que multiplica al error cartesiano en posición que viene representado por $Ex \in \Re^6$. Kv es la constante derivativa que multiplica al error cartesiano en velocidad representado por $Ev \in \Re^6$ y finalmente el resultado es la aceleración representada como $\tau \in \Re^7$.

Puesto que el controlador diseñado no utiliza el modelo dinámico del robot, el controlador no tiene en cuenta las reacciones físicas del mismo con el medio. Ciertamente el comando de control utilizado en este controlador son aceleraciones y no pares, esta es una aproximación que se está realizando con el propósito de reducir el cálculo computacional añadido al introducir el modelo dinámico del robot para únicamente calcular el par. Además, como se presentará en el apartado de resultados el funcionamiento es correcto igualmente, ya que los movimientos en el espacio exterior nunca deben ser unos movimientos extremadamente rápidos, si no movimientos muy lentos y controlados.

3.4.1. Localización utilizando un código Aruco

Todo el proceso explicado en el bucle de control anterior es posible hacerlo sin mayor problema en un robot anclado al suelo ya que la base del robot nunca se mueve, está anclada al suelo. Sin embargo, en el caso de este trabajo no existe gravedad y el robot es un cuerpo flotando libremente en el espacio. Esto quiere decir que cualquier movimiento que este realice el robot producirá unas inercias que inevitablemente desplazarán al robot y por lo tanto a su base.

Puesto que la base del robot es la referencia de la cadena cinemática, si esta se mueve continuamente el efector final del robot nunca podrá llegar a alcanzar el punto deseado en el mundo, ya que hablando con respecto al robot el punto siempre está en la misma posición puesto que el robot se mueve junto con el punto. Este problema podría arreglarse diseñando un sistema de propulsión en la base que contrarrestará los efectos del movimiento en el robot. El problema es que esta solución además de ser económicamente más costosa de implementar también es mucho más complicada de implementar, aunque no es nada sobre lo que no exista desarrollo y que no esté planteando para otros sistemas robóticos en el ambiente espacial.

Por esta razón para el proyecto se optó por una solución basada en software que se pudiera realizar sin la necesidad de emplear tantos recursos como la solución de los propulsores. Esta solución se basa en calcular la posición de la base del robot con respecto a un código aruco que deberá estar instalado en una parte fija de la estación espacial. De esta manera el robot debe saber cuál es la transformación espacial que existe desde el código aruco hasta el punto al que el robot debe llevar el efector final. Mediante el uso de visión por computador con los códigos arucos es posible obtener la posición de la base del robot con respecto a este código, además como la transformación desde el código aruco a la posición objetivo es una transformación fija y predefinida se puede utilizar para obtener la posición final con respecto a la base del robot, aunque esta se esté moviendo. Cada vez que se mueva el robot la posición final del objetivo será actualizada a una diferente, esto va a provocar que surjan muchos problemas durante la generación de la trayectoria ya que esta irá cambiando durante el movimiento, pero será explicado en el próximo apartado 3.4.2.

Los códigos Arucos son marcadores ampliamente utilizados para la localización de cámaras en aplicaciones de realidad aumentada, realidad virtual y algunos trabajos de robótica como pueden ser [20], [21], [22], [23]. Para obtener la posición de una cámara en el espacio es necesario encontrar correspondencias entre puntos conocidos y su proyección en la imagen. El método utilizado en este trabajo utiliza los marcadores aruco como bien se ha remarcado a lo largo de la explicación. Son una librería, o un diccionario de marcadores también llamado en la literatura, en la que cada marcador se compone de una imagen binaria cuadrada y para cada uno se le corresponde un id, un número. En la Figura 3.17 se puede ver un ejemplo de código Aruco.

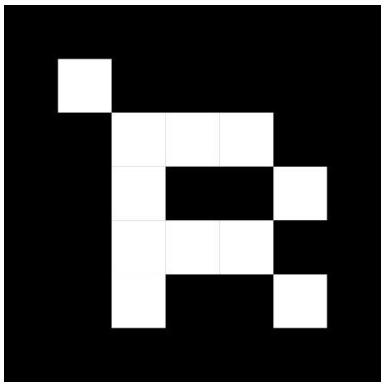

Figura 3.17: Código Aruco utilizado en el trabajo, con la ID 238. **Fuente:** https://chev.me/arucogen/

Para poder utilizar esta funcionalidad se ha utilizado el paquete de ROS llamado Aruco-Ros [24], que ha sido implementado por la empresa PAL Robotics [16] la cual es la misma que la creadora del robot Talos Original. Este paquete utiliza el método propuesto en [25] para obtener la posición de la cámara en el espacio, además también es capaz de transformar esta posición a cualquier eslabón del robot utilizando la funcionalidad TF [26] de ROS, que permite conocer en todo momento la posición y la relación entre todos los eslabones del robot. De manera que ha sido combinando todas estas funcionalidades que se ha conseguido abordar el problema del control sobre robots flotantes en el espacio.

3.4.2. Generación de trayectorias

Tal como se ha explicado en apartados anteriores las trayectorias generadas en este proyecto se realizan mediante la interpolación de puntos utilizando un polinomio de grado cinco para

ello. Utilizar un polinomio de grado cinco para realizar la interpolación de los puntos supone que podemos ser capaces de especificar como restricciones las posiciones, velocidades y aceleraciones tanto iniciales como finales que queremos que siga nuestra trayectoria. Esta es una propiedad extremadamente útil en este trabajo por lo que se comentará a continuación.

Tal y como se ha explicado antes el problema principal es que las inercias y fuerzas provocadas al mover cualquier parte del robot provocan que la base del mismo se desplace como reacción a los movimientos y nunca pueda alcanzar el objetivo del mundo real que se propone como consigna al controlador. Para esto se utiliza la solución de tener localizada la base del robot con respecto a un marcador Aruco fijo colocado en la estación espacial. Pero esto en ningún caso evita que la base del robot se mueva, por lo tanto, surge el problema de que el punto final al que se quiere llevar el brazo va estar por lo general continuamente alejándose de la base del robot.

Esto supone el problema de que no se puede crear una única trayectoria desde la posición inicial del efector final hasta la posición final, ya que esta se mueve con el robot. Para ello se ha diseñado e implementado una serie de funciones que crean una nueva trayectoria que irá desde la posición actual del efector final en cada momento hasta la nueva posición recibida con respecto de la nueva localización de la base del robot.

Primeramente, se tiene una función que recibe continuamente la posición de la base del robot con respecto al código Aruco y si esta cambia según un umbral (en cualquiera dirección u orientación) entonces se utilizará la nueva posición de referencia para calcular una nueva trayectoria. Para indicar el tiempo que debe durar la trayectoria se puede indicar desde el archivo de configuración del controlador .YAML. Sin embargo, el tiempo indicado solamente será una referencia de lo que tendría que durar una trayectoria completa desde el punto inicial hasta el final, pero como diferentes trayectorias van a ser trazadas durante la ejecución, esta referencia se utiliza para obtener el tiempo que deberá durar cada nueva trayectoria de manera proporcional a la distancia al punto final.

Para realizar esto, la primera vez que llega un punto de consigna donde mover el brazo se crea una relación proporcionalmente directa teniendo en cuenta el tiempo y la distancia entre los puntos en el espacio 3D. Utilizando esta relación inicial se calculará un nuevo tiempo

para cada nueva trayectoria que deba ser trazada durante el transcurso del camino. Normalmente esto aumentará el tiempo de la trayectoria total unos segundos.

Para calcular la distancia entre los frames (Matriz de transformación) de inicio de trayectoria y final de trayectoria y así poder calcular el tiempo que debe durar esa nueva trayectoria se calcula por separado la distancia entre los ángulos y la distancia 3D utilizando las siguientes ecuaciones.

$$R = F^{-1} \cdot S \tag{3.2}$$

$$\alpha = \cos^{-1}\left(\frac{(Traza(R) - 1)}{2}\right) \tag{3.3}$$

En 3.2 y 3.3 se puede ver las ecuaciones utilizadas para obtener un valor de diferencia angular α para 2 frames en el espacio 3D. Donde F $\in \Re^{3x3}$ es la matriz de rotación del frame final y S $\in \Re^{3x3}$ la del frame inicial.

$$d = \sqrt{(x_i + x_f)^2 + (y_i + y_f)^2 + (z_i + z_f)^2} \tag{3.4}$$

Mientras que en 3.4 se puede ver la ecuación para obtener la distancia entre los dos frames iniciales y finales donde x_i representa la posición inicial y x_f la posición final y de la misma manera la nomenclatura sigue para las demás variables. Por último, ya que el objetivo de estos cálculos es tener una medida de la distancia total que se va a trazar, se utilizará la media entre la distancia angular α y la distancia en el espacio cartesiano d para representar esta distancia total.

Además, cabe tener en cuenta puesto que para que la trayectoria pueda seguir un camino de manera fluida, sin sufrir perturbaciones en su aceleración o velocidad, y con tal de asegurar el seguimiento de la trayectoria se guardará en cada momento cuál es el valor de la velocidad y la aceleración de la trayectoria actual y cuando sea necesario crear una nueva trayectoria se utilizaran los últimos valores de la anterior como valores iniciales de la nueva. Por esta razón es por la que se necesitaba utilizar una interpolación quintica para la trayectoria.

3.4.3. Adaptación del controlador sin visión por computador

Además del controlador presentado anteriormente también se ha diseñado una versión del mismo que funciona sin añadir la característica de la localización utilizando el código aruco. Este controlador cartesiano recibe como consigna un punto en el espacio 3D y una orientación definida por los ángulos de Euler (Roll, Pitch y Yaw) a la que mover el efector final. También se enviará para cada punto un tiempo especificando el tiempo total que debe durar la trayectoria. Puesto que este controlador no utiliza visión por computador no cuenta con la característica de conocer la posición de su base por lo que la posición final del brazo nunca coincidirá con la referencia del mundo y por esta misma razón tampoco se realiza el cálculo de trayectorias varias veces durante el transcurso del movimiento, únicamente se calculará una única trayectoria hasta que se alcance la tolerancia configurada o el tiempo de trayectoria.

Este controlador se ha pensado para realizar las pruebas necesarias antes de llevar el mismo procedimiento a la simulación de la estación espacial, pero es un controlador totalmente válido para utilizarse en entorno en los que el robot se encuentre anclado al suelo y la base no se desplace. Además esta adaptación también permite a cualquier usuario utilizar el controlador para cualquier otra aplicación ya que este cuenta con la capacidad de saber cuándo se ha recibido un nuevo frame diferente para mover el brazo y también es capaz de comunicar al exterior cuando el efector final ha alcanzado la posición deseada según la tolerancia indicada por el usuario en el archivo .YAML de configuración. Mediante estas dos características permite que cualquier usuario pueda realizar una tarea en la que el brazo debe ir a diferentes puntos realizando una trayectoria diferente entre cada par de puntos.

3.5. Controlador dinámico de trayectorias cartesianas utilizando localización mediante visión por computador teniendo en cuenta los esfuerzos del robot en la base

Además del controlador explicado en el apartado anterior también se ha llevado a cabo la implementación de un controlador dinámico. Este controlador utiliza el modelo dinámico del robot para poder calcular los pares que deben actuar en cada articulación del robot siendo consciente al mismo tiempo de los efectos y las reacciones que se provocarán en la base del robot. Aunque no se ha llevado a cabo en este proyecto por falta de tiempo, este controlador ofrece la posibilidad de mediante unos propulsores o cualquier implementación válida, compensar el movimiento de la base y así poder incluso mejorar la precisión del movimiento del brazo. Esta ha sido la razón principal que ha llevado a la implementación de este controlador, ya que es extremadamente apropiado para el entorno espacial. La implementación realizada se ha basado en la misma explicada en [27].

La ley de control propuesta se basa en la siguiente ecuación presentada en 3.5.

$$\tau = H^* J_g^+ \left(\ddot{x}_d + Kv(\dot{x}_d - x) \; + \; Kp(\, x_d - x) \; - \; \dot{J}_g \dot{q} \right) + \; C_m^* \tag{3.5}$$

Donde $H^* \in \Re^{7x7}$ es una matriz compuesta a partir de las diferentes relaciones entre submatrices de inercia del robot. Esta matriz se compone a partir de la matriz de inercia de la base del robot (H_b) y la matriz de inercia acoplada entre el manipulador y la base (H_{bm}). A partir de todas estas se puede obtener $H^* \in \Re^{7x7}$ tal y como se muestra en 3.6.

$$H^* = \; H_b + H_{bm}^T H_b^{-1} H_{bm} \tag{3.6}$$

Estas matrices se pueden obtener de la matriz de inercia completa del robot que viene definida por todas estas submatrices como se puede ver en la ecuación 3.7.

$$H = \begin{bmatrix} H_b & H_{bm} \\ H_{bm}^T & H_m \end{bmatrix} \tag{3.7}$$

Además de H^*, también encontramos $J_g^+ \in \Re^{7x6}$ la cual representa la pseudo inversa del Jacobiano generalizado del robot, el cual se puede obtener mediante el Jacobiano del

manipulador y el jacobiano de la base, así como diferentes matrices de inercia tal y como se especifica en la ecuación 3.8.

$$J_g = J_m - J_b H_b^{-1} H_{bm} \tag{3.8}$$

Para realizar el cálculo de la pseudoinversa de la matriz jacobiana se ha utilizado el método de Moore-Penrose implementándolo directamente con la librería Eigen, para más datos sobre cómo funciona el método se puede consultar [28]. La componente $\dot{J}_g \in \Re^{6x7}$, se puede denominar como la derivada de la matriz Jacobiana generalizada del robot que viene dada por los términos derivados con respecto del tiempo de la ecuación anterior como se puede ver en 3.9. También, multiplicando a este término se encuentra la velocidad articular $\dot{q} \in \Re^7$.

$$\dot{J}_g = \dot{J}_m - \dot{J}_b H_b^{-1} H_{bm} \tag{3.9}$$

Además de estos términos comentados anteriormente, también podemos encontrar los términos que se han comentado en la ley de control del controlador presentado en el punto anterior. Estos son $\ddot{x}_d$, que hace referencia a la aceleración cartesiana deseada del efector final, $\dot{x}_d$ que hace referencia a la velocidad cartesiana deseada del efector final y x_d que hace referencia a la posición deseada del efector final. Para los términos que no cuentan con el subíndice *d*, se puede considerar que representan la misma variable, pero para la trayectoria actual del robot en cada momento, no para la deseada. También, podemos encontrar las constantes proporcional y derivativa respectivamente Kp, Kv.

Por último, se puede encontrar sumando al final de la ecuación planteada en 3.6 el término C_m^*, este término hace referencia a la matriz de Coriolis del robot, pero en el trabajo desarrollado se ha considerado este como nulo para poder realizar la implementación de manera más eficiente, computacionalmente hablando.

Mediante la ley de control explicada en los párrafos anteriores el controlador puede generar pares articulares para todas las articulaciones del brazo. Además, teniendo calculados todos los términos de la ley de control 3.5, también es posible calcular de manera fácil la velocidad a la que se movería la base a causa de los efectos del movimiento del brazo. Para realizar este cálculo se podría utilizar directamente la ecuación vista en 3.10.

$$V_b = -H_b^{-1} H_{bm} \dot{q} \tag{3.10}$$

Todo lo explicado anteriormente se ha realizado dentro de la función *update()* del controlador, en esta se utilizará la cinemática directa y la jacobiana directa para obtener tanto el error posición cartesiana como en velocidad. Mediante la planificación de trayectorias se obtiene la aceleración deseada en cada momento. Y es utilizando las características de la librería Pinocchio que podemos obtener todas las matrices necesarias tanto del modelo dinámico como diferencial en cada iteración. En el momento de obtener las características dinámicas del robot como las matrices de inercias y también las matrices Jacobianas generalizadas se han bloqueado todas las articulaciones que no pertenecen al brazo, para así facilitar los cálculos ya que el único brazo que se va a mover será el brazo derecho, de esta manera se puede decir que se ha hecho la suposición que cuando el brazo se mueve el resto del cuerpo actúa como un único cuerpo rígido sin articulaciones.

Además, este controlador también es capaz de ajustar la trayectoria que lleva el extremo del brazo siempre que se produzca un desplazamiento significativo en la base utilizando los mismos métodos explicados en el apartado anterior tanto como para realizar la el cálculo de la nueva trayectoria como para el cálculo de la posición de la base del robot utilizando el código Aruco instalado en la estación espacial internacional. Los apartados en los que ha sido explicado el funcionamiento interno de estos métodos han sido en el 3.4.1 y el 3.4.2.

En cuanto al resto de funciones esenciales para definir el controlador en Ros Control, la función *init()* al igual que en el controlador anterior se ha utilizado para cargar las características específicas del controlador leídas desde el archivo de configuración .YAML. Y en la función starting() se ha obtenido el frame inicial para estabilizar el brazo en la posición inicial y que así no se mueva hacia cualquier dirección sin sentido. Además, se ha cargado el archivo .URDF para que pueda inicializarse la librería Pinocchio y así poder calcular el modelo dinámico durante el bucle de control principal.

3.5.1. Adaptación sin visión por computador

Al igual que con el controlador cinemático presentado en el apartado 3.4. también se ha realizado una adaptación de este controlador dinámico para que se pueda utilizar sin incluir las capacidades del ajuste de trayectoria a medida que se desplaza la base del robot. En esta

adaptación se permite que el usuario envíe un mensaje al controlador con el punto de referencia, así como el tiempo que se desea que dure esta trayectoria. La posición en el espacio 3D debe ser definida por la posición en X, Y, Z y los ángulos Roll, Pitch y Yaw que se desea que tenga el efector final con respecto al sistema de coordenadas de la base. Exactamente igual que en la adaptación para el controlador cinemático presentada anteriormente. Para añadir esta capacidad, además de eliminar todas las características relacionadas con el ajuste de la trayectoria de manera continua y la localización de la base también se ha añadido un topic de ROS que recibe mensajes utilizando un tipo de mensajes propio definido por 7 variables representado las 6 dimensiones espaciales y el tiempo de trayectoria.

Mediante esta adaptación se consigue poder utilizar el controlador dinámico presentado en el punto justo anterior de una manera más libre y poder utilizarlo para diferentes aplicaciones que se quiera en un futuro, pues este cuenta también con una función para comparar cuando un nuevo punto final ha sido enviado y es capaz de comunicar cuando el controlador ha llegado a la posición final cumpliendo con la tolerancia indicada en el archivo de configuración .YAML.

4. Resultados

Para analizar los resultados de los controladores implementados ha sido necesaria la utilización de diferentes códigos y así como otros controladores para las diferentes partes del cuerpo que son necesarias utilizar en la simulación del comportamiento como las articulaciones del cuello y las manos.

La utilización de estas partes del cuerpo es imprescindible para la tarea seleccionada por el robot ya que se trata de agarrar un asa de la estación espacial internacional para así estabilizar su posición mientras pudiese en un caso hipotético realizar una tarea con el otro brazo.

4.1. Control de las manos del robot astronauta

Para dotar al robot astronauta de la capacidad de abrir y cerrar las manos se ha diseñado un servicio de ROS [29] el cual se encarga de enviar las órdenes correctas a los controladores de las manos para abrir o cerrar cada mano que puede ser elegida por el usuario.

Para ello se ha utilizado el controlador de trayectorias articulares de ros control que es el que ya se utilizaba de manera previa por la empresa de PAL Robotics [16], ya que estas manos robóticas también pertenecen a sus códigos públicos. Las manos utilizadas son llamadas Hey-5 y cuentan únicamente con 3 motores, uno controlará el dedo pulgar, otro el dedo índice y otro los dedos corazón, anular y meñique de manera simultánea. De manera que para que este servicio creado fuera fácil y cómodo de usar desde el exterior, se decidió crear un mensaje explícito y totalmente nuevo. En el mensaje se indicará la operación a realizar que puede ser abrir o cerrar y la mano sobre la que se debe actuar (izquierda o derecha).

Para actuar sobre cada articulación de la mano se ha hecho uso del paquete de ros *trajectory_joint_controller* [30]. Este paquete también proporciona un tipo de mensaje para crear trayectorias de manera genérica y es por tanto lo que se ha utilizado. Este mensaje necesita que se le indiquen cuáles serán las articulaciones a controlar, una posición articular para cada posición y una medida tiempo para llegar hasta esa posición, se puede indicar varias posiciones siempre que se indiquen diferentes tiempos de duración de trayectoria para cada una, pero en el caso de este trabajo únicamente se ha enviado una posición articular que cierra la mano lo mejor posible.

4.2. Control del cuello del robot astronauta

Para poder mover el cuello del robot y así poder estar observando continuamente el código aruco y mantenerse localizado se ha hecho uso de uno de los controladores base de Ros Control, exactamente se ha utilizado un controlador de posición básico controlado mediante un PID. El ajuste del PID se ha realizado de manera que la constante proporcional toma el valor de 500, la constante integral el valor de 10 y la derivativa 1. Esta configuración se ha realizado mediante un archivo .YAML al igual que con los controladores propios y el controlador se debe cargar mediante un archivo .launch.

```
1.  astronaut_head_tilt_controller:
2.    type: effort_controllers/JointPositionController
3.    joint: head_1_joint
4.    pid: {p: 500.0, i: 10.0, d: 1.0}
5.
6.  astronaut_head_pan_controller:
7.    type: effort_controllers/JointPositionController
8.    joint: head_2_joint
9.    pid: {p: 500.0, i: 10.0, d: 1.0}
10.
```

Figura 4.1: Código de configuración para los controladores de la cabeza. **Fuente:** Elaboración Propia

Como se puede ver en la figura 4.1 se ha hecho uso de dos controladores, uno se encargará de controlar el giro en el eje horizontal (*pan*) y otro se encargará de controlar el giro en el eje vertical (*tilt*). Cargando este controlador mediante el gestor de controladores de ros control, se ofrece al usuario una interfaz mediante la comunicación de topics para enviar los comandos requeridos a cada controlado. Será mediante esta función que se realizará el control de la cabeza del robot dentro del código.

4.3. Tarea a realizar

La tarea a realizar para obtener los resultados de los controladores y del trabajo realizado se basa en que el robot talos astronauta mire el código aruco con la cámara que posee en su cabeza moviendo el cuello y orientándose hacia una posición conocida en este caso. Al realizar esta acción el controlador recibirá la posición final a la que debe llevar el brazo y comenzará a realizar el movimiento, mientras tanto se estará comprobando desde el programa de la tarea si el controlador ha conseguido llegar a la posición final o no. Esto se comunicará desde el interior del controlador hacia el exterior en base a la tolerancia que se le haya configurado al controlador, en los ejemplos que se mostrarán a continuación se ha configurado para que el robot considere como que ha llegado a la posición final cuando al menos ha alcanzado un error del 0.015 en todas las coordenadas.

Una vez que se detecte que el robot ha conseguido llevar el brazo hasta la posición final se utilizará el servicio de ROS explicado en el subapartado 4.1 para cerrar la mano derecha, que en este caso se corresponde con el brazo que se está moviendo. Todo esto se realiza para poder demostrar que con el trabajo realizado el robot Talos astronauta es capaz de poder

trabajar en el entorno de la estación espacial, realizando las adaptaciones mínimas o nulas en el mismo gracias a sus características, además mediante esta tarea el robot es capaz de estabilizar su posición gracias a que está cogido a un asa en la estación espacial y ya podría mover el otro brazo para realizar cualquier tarea en caso de que fuera necesario sin peligro de que el robot se quede flotando en el espacio totalmente descontrolado.

4.4. Comparación de los controladores usados

En los siguientes subapartados se procederá a exponer todas las gráficas obtenidas sobre el comportamiento específico de cada controlador con las mejores constantes que se han podido encontrar para el funcionamiento de cada uno. Se comentarán todas las gráficas obtenidas y se explicará lo mejor posible qué es aquello que se puede deducir de cada una.

Para la obtención de todas las gráficas que se mostrarán en este apartado se ha utilizado la aplicación Plotjuggler [31] ya que cuenta con un paquete en ROS que facilita su uso. Además, para poder obtener los datos de cada controlador se han diseñado una serie de mensajes y topics específicos únicamente para poder dotar al controlador de la capacidad de comunicar los datos con el exterior y de esta manera se ha utilizado la característica de ROS las *rosbags*, para guardar todos los datos de los topics que se han creado y así poder cargar los datos de cada simulación de manera separada directamente con cada rosbag en la aplicación Plotjuggler.

4.4.1. Controlador de trayectorias cartesianas multiarticular utilizando localización mediante visión por computador

En este subapartado se comentarán las gráficas obtenidas del comportamiento del controlador de trayectorias cartesianas multiarticular, que utiliza el código aruco fijo instalado en la estación espacial para localizar su base y así mejorar la precisión del movimiento en cuanto a un marco de acción general. Para todas las gráficas mostradas se han utilizado unas constantes $Kp = 300\ y\ Kv = 0.1$. También se ha utilizado una referencia de tiempo de 9 segundos.

La primera gráfica que se comentará se puede observar en la figura 4.2, en esta se muestra el error de control que es el que se utiliza directamente en la ley de control. Puesto que este controlador va calculando nuevas trayectorias en cada momento el comportamiento de este error no es el que se podría esperar en un controlador normal, en cambio en todo momento el error que se obtiene es un valor muy pequeño.

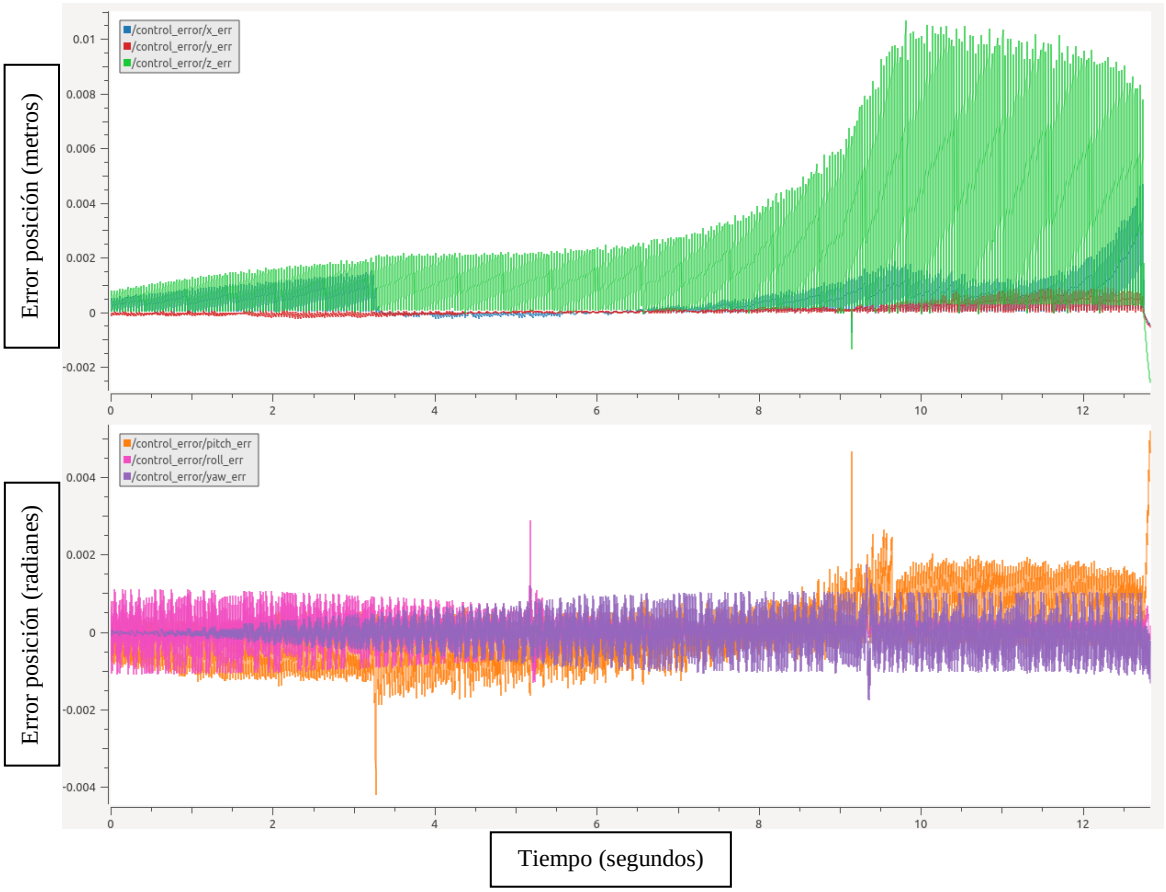

Figura 4.2: Error de control (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia

En esta gráfica presentada se puede ver que el mayor error de control que se obtiene en las coordenadas cartesianas se encuentra con un valor de 0.01 metros lo que no es un error grande, mientras que en la orientación del efector final se encuentra que el mayor valor no supera los 0.002 radianes. Sin embargo, esta gráfica no se puede tomar como un resultado exactamente representativo puesto que el aumento en el error posiblemente es debido al movimiento de la base en la dirección opuesta y al cálculo continuo de nuevas trayectorias. En cambio, en la figura 4.3 que se muestra a continuación se puede ver unos resultados más

representativos que son la reducción del error en todas las coordenadas desde la posición inicial hasta 0.

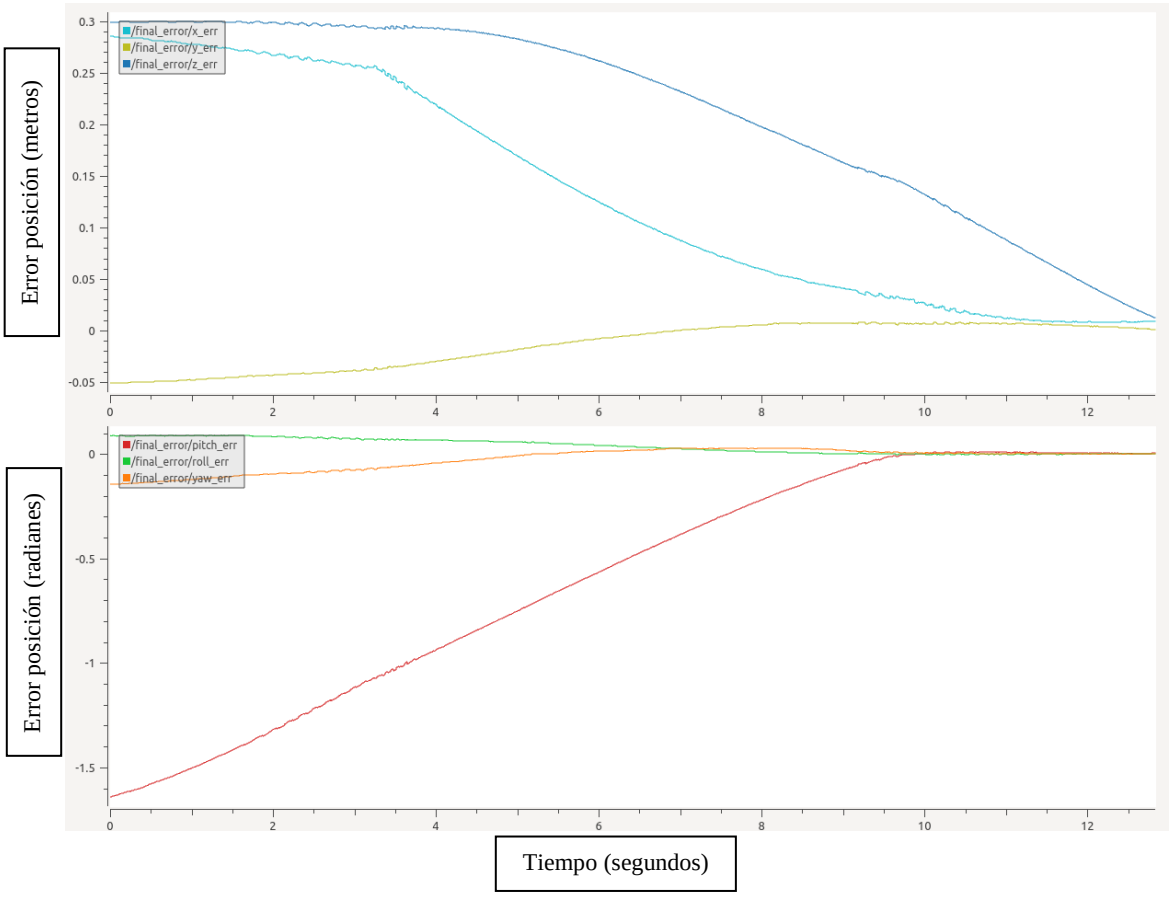

Figura 4.3: Reducción del error global a lo largo de toda la trayectoria (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia.

En esta gráfica sí que se puede ver de una manera más clara que mediante el controlador propuesto se puede reducir el error directamente en las coordenadas cartesianas hasta que este se convierte a 0. En la gráfica de arriba de la figura 4.3 se puede ver que el error no llega hasta 0 completamente ya que la tolerancia del controlador está configurada a 0.015.

Seguidamente en la figura 4.4 se muestra cuáles son los valores del par articular enviado a cada articulación durante el funcionamiento de la trayectoria realizada para agarrar el asa.

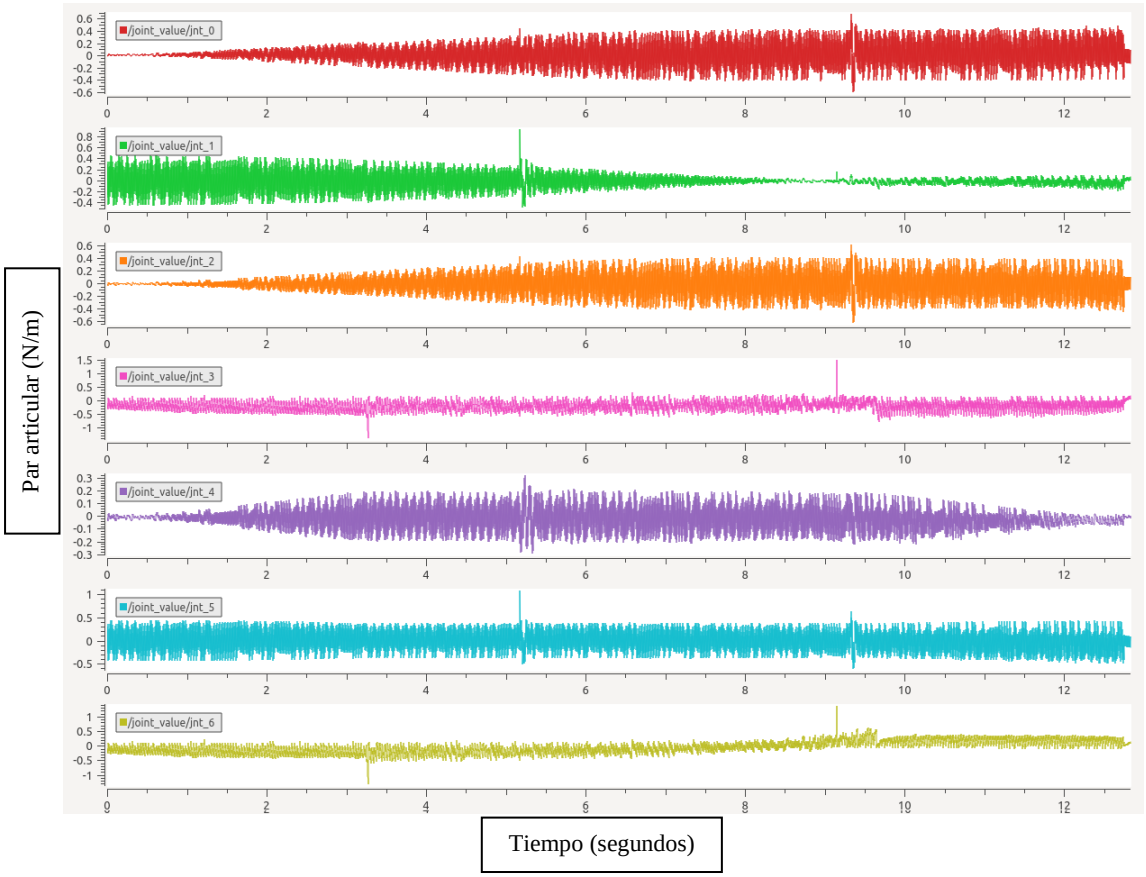

Figura 4.3: Pares articulares para cada una de las 7 articulaciones (orden descendente de 0 a 6). **Fuente:** Elaboración propia.

Los pares que se pueden ver en estas graficas mostradas en la figura anterior en ningún momento superan el valor de 1 N/m lo que se puede considerar como un par relativamente bajo y además se puede ver que son gráficas constantes lo que da una sensación de buen funcionamiento durante el transcurso de toda la trayectoria. Además de estas gráficas también se ha obtenido la evolución del error en velocidad utilizado en la ley de control. Sin embargo, al igual que en el error de control como se comentaba para la figura 4.2, esta gráfica, aunque no muestra un error excesivamente alto en ningún momento (ya que el error se obtiene entre puntos o velocidades continuas de la interpolación realizada) sí que se puede ver que sufre altas variaciones.

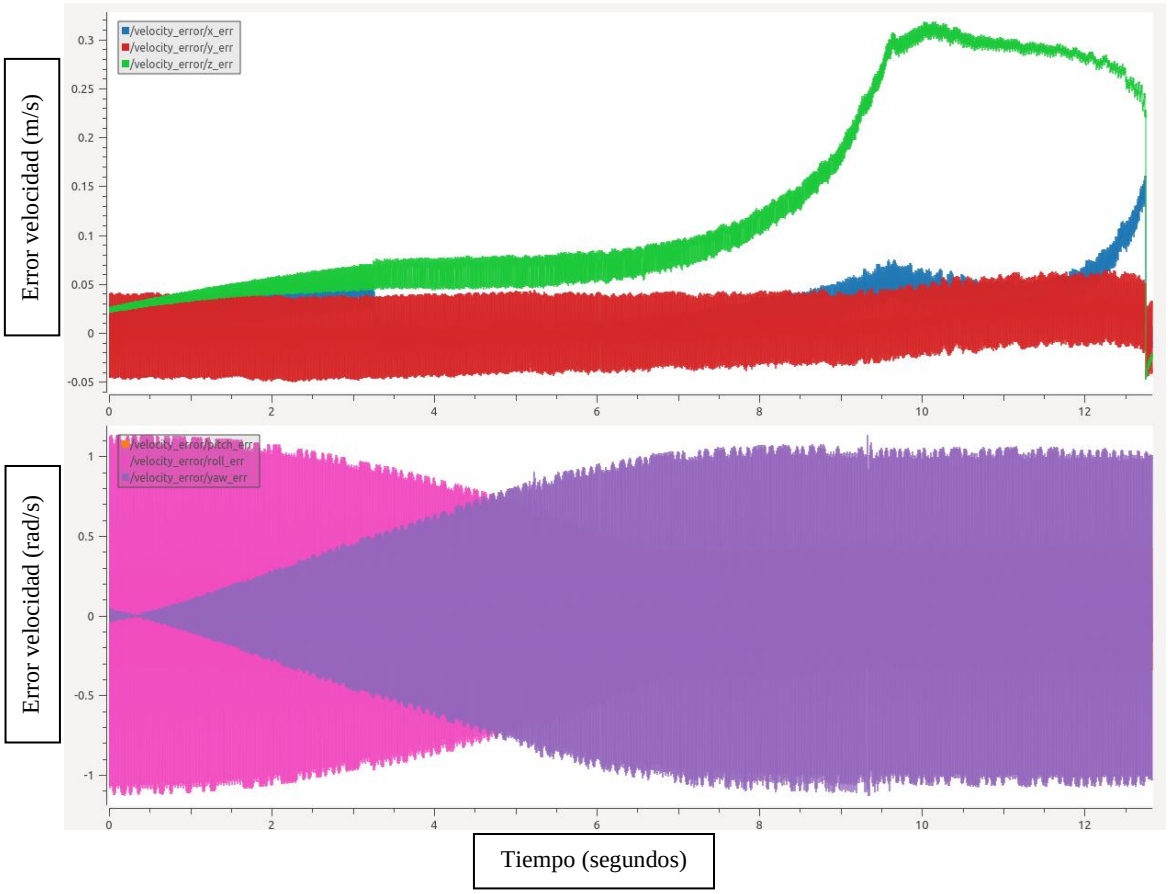

Figura 4.4: Error de velocidad cartesiana (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia.

Por último, cabe presentar las gráficas en las que se muestra la comparación entre la trayectoria cartesiana deseada durante todo el movimiento del brazo con la trayectoria realizada por el extremo del brazo robótico. Estos resultados se pueden ver en la figura 4.5 a continuación.

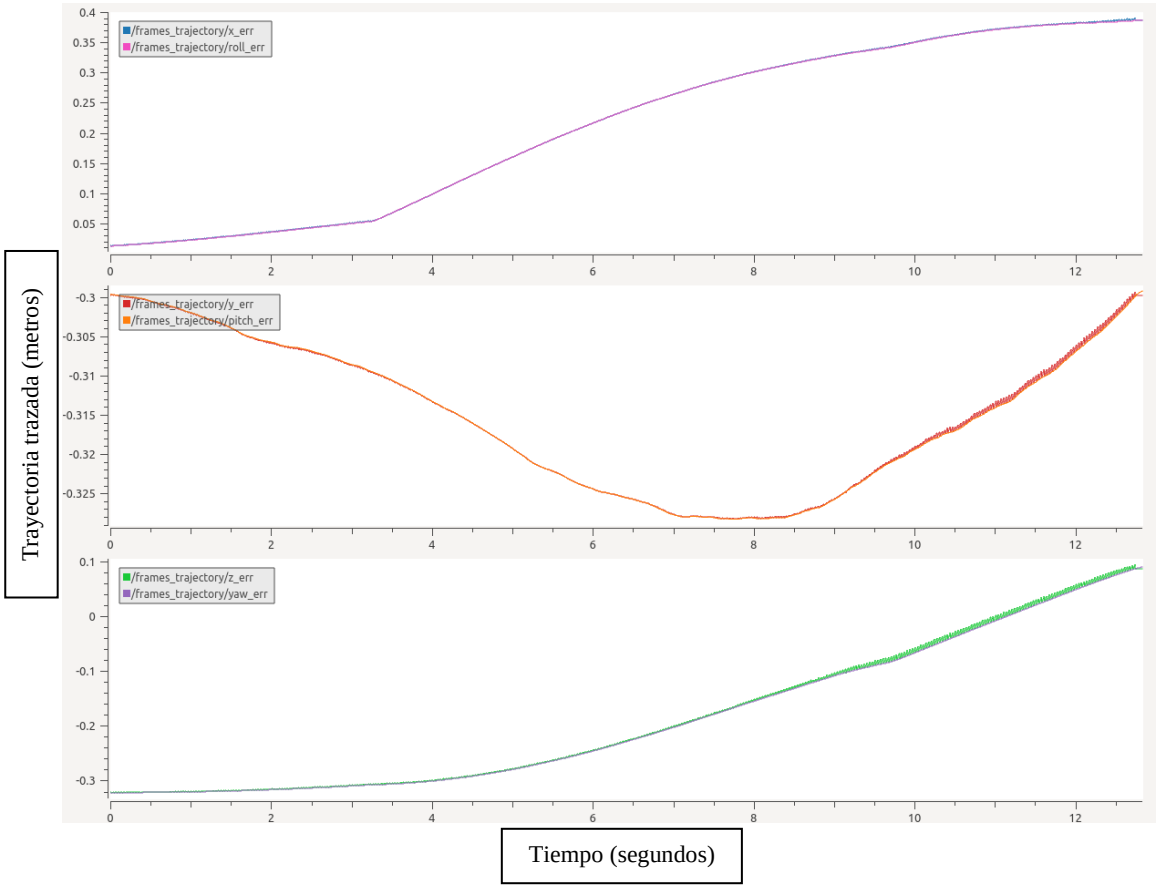

Figura 4.5: Seguimiento de la trayectoria cartesiana (Arriba coordenada X, en medio coordenada Y, abajo coordenada Z). **Fuente:** Elaboración propia.

En la figura 4.5 mostrada anteriormente, se puede observar el seguimiento de la trayectoria para las coordenadas X, Y, Z del efector final junto con cada una con la trayectoria deseadas que se ha calculado. Como se puede ver en todo momento las curvas se encuentran coincidentes lo que significa que se ha realizado un muy buen seguimiento de la trayectoria, esto viene dado gracias a que en la figura 4.2 se puede observar un error muy bajo durante todo el movimiento, incluso se puede apreciar que al final cuando el error aumentaba levemente en la gráfica superior de la figura 4.2 de igual manera en estas graficas las curvas no son tan coincidentes, pero es prácticamente inapreciable.

4.4.1.1. Adaptación sin visión por computador

En este apartado se procederá a explicar y comentar las gráficas obtenidas con el controlador equivalente al anterior pero que no cuenta con las características de ajuste de trayectoria a medida que se desplaza la base por las inercias producidas al mover el brazo del robot. Las gráficas serán las mismas que en el caso anterior, pero con los datos obtenidos en esta iteración. Para todas las gráficas mostradas se han utilizado unas constantes $Kp = 300$ y $Kv = 0.1$. También se ha utilizado una referencia de tiempo de 9 segundos.

Las primeras gráficas se muestran en la figura 4.6, en esta se puede ver el error de control calculado entre cada posición de la trayectoria calculada. En esta figura se pueden ver diferencias notables en comparación con el controlador anterior, pero la gran mayoría de estas son debidas que en este caso no se están calculando nuevas trayectorias continuamente.

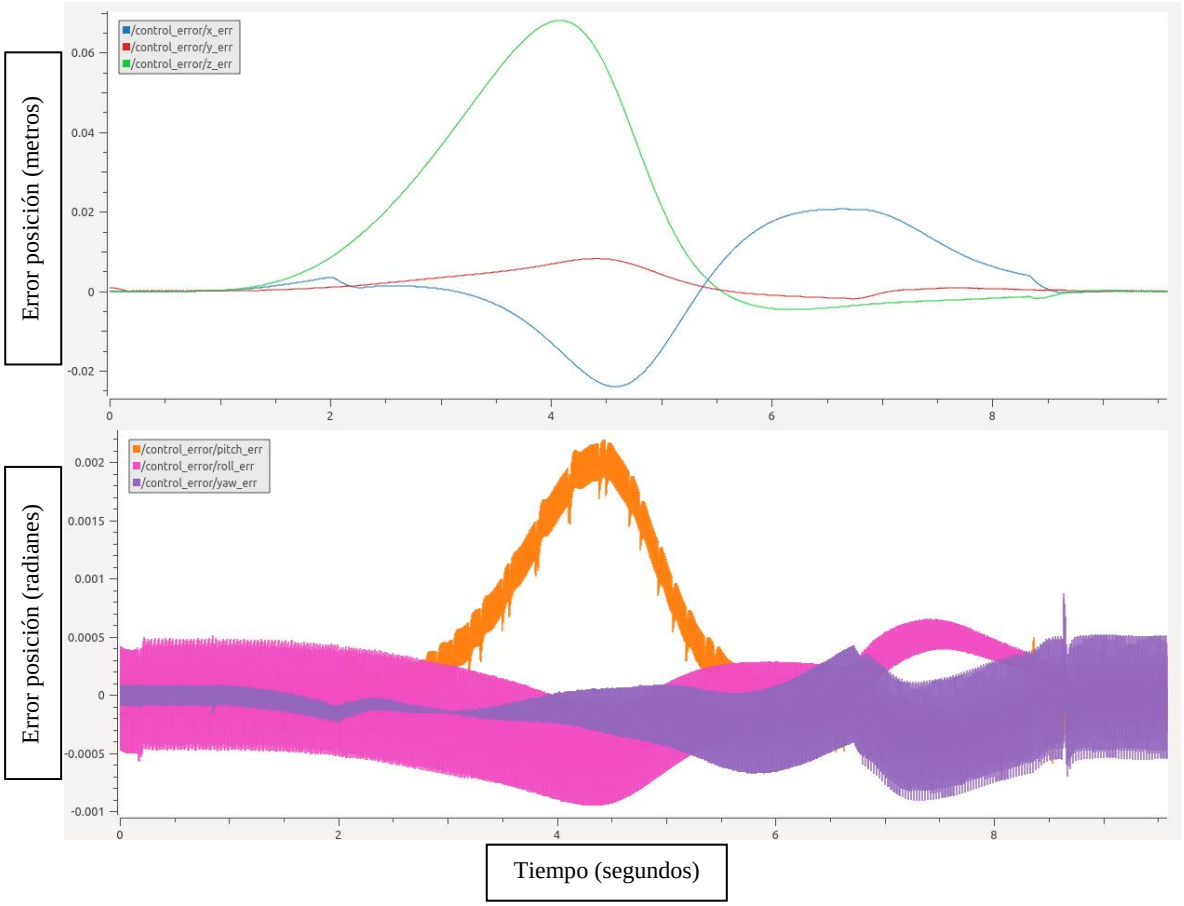

Figura 4.6: Error de control (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia

En la figura 4.6 se puede ver cómo al igual que en el caso anterior el error de control empieza a 0 debido a que como se ha comentado a lo largo del trabajo el controlador calcula su cinemática directa para estabilizar el brazo en la posición inicial. Sin embargo, en esta figura se puede ver como finalmente el error sí que es reducido hasta 0 sin ningún problema. También se observa que los valores de la gráfica inferior son un poco oscilatorios, pero teniendo en cuenta los valores tan pequeños entre los que se mueven se hace despreciable. De igual manera que en el apartado anterior seguidamente se presenta una gráfica donde se ve la reducción del error hasta llegar a la posición final.

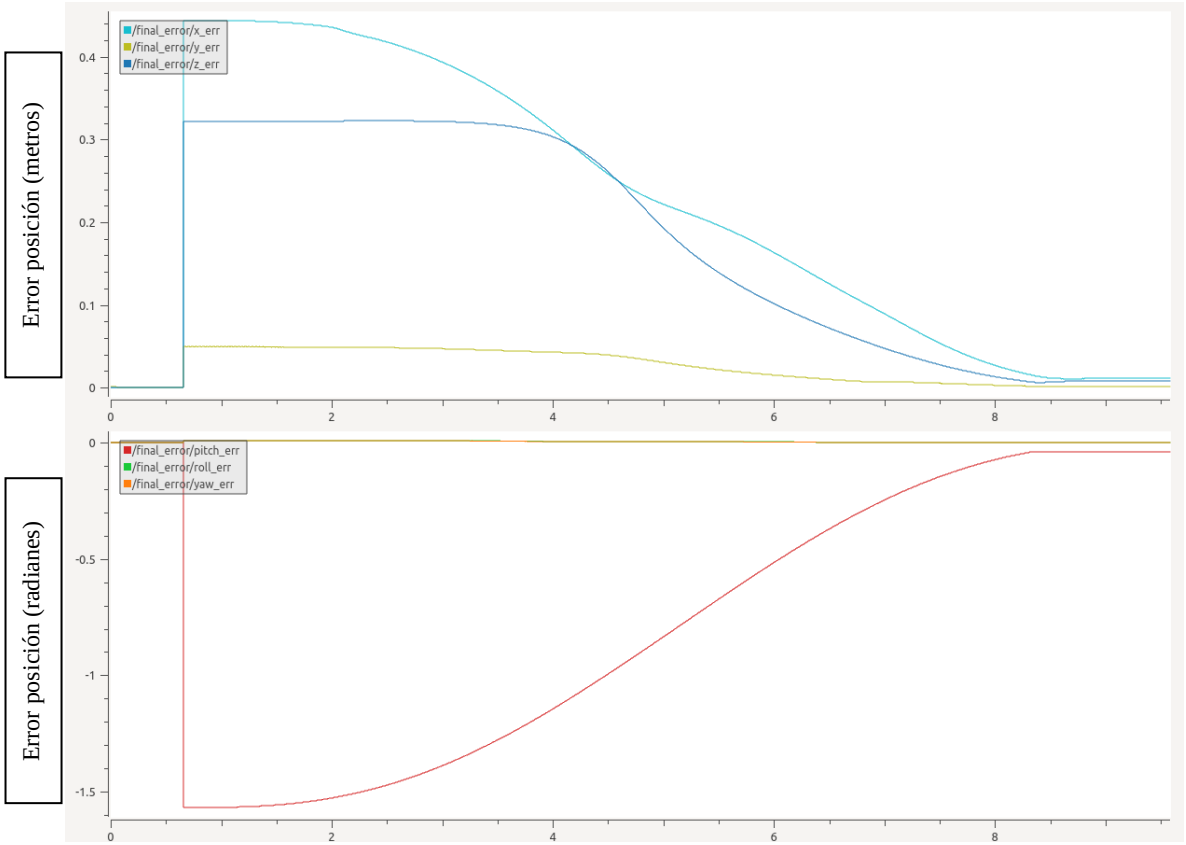

Figura 4.7: Reducción del error global a lo largo de toda la trayectoria (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia.

En la figura 4.7 se pueden ver unas discontinuidades notables que son debidas a que los datos han comenzado a tomarse antes de que se enviará un comando de posición para el controlador. Pero una vez que se recibe este comando se puede ver como el error va bajando y disminuyendo hasta llegar a 0 o al valor de tolerancia al final de la trayectoria. Las

siguientes gráficas que se quieren mostrar se pueden ver en la figura 4.8, en esta figura se observa el par calculado para cada articulación del brazo robótico a lo largo de la trayectoria.

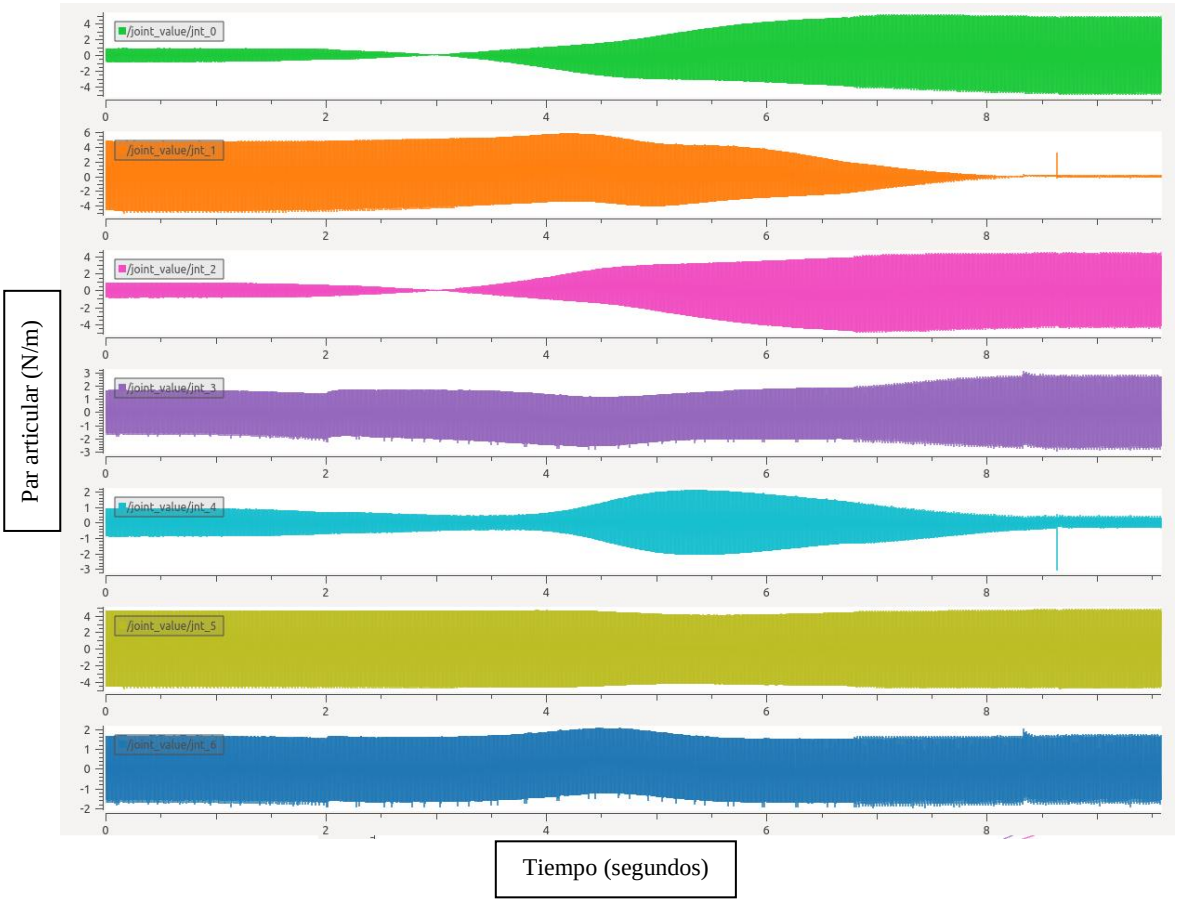

Figura 4.8: Pares articulares para cada una de las 7 articulaciones (orden descendente de 0 a 6). **Fuente:** Elaboración propia.

Los pares que se pueden ver en estas gráficas mostradas en la figura anterior llegan a unos valores un poco más altos que en el controlador anterior. Se cree que esto se puede estar dando por que en el controlador anterior se están restringiendo los valores de par al estar continuamente aplicando una nueva trayectoria o bien porque la posición a la que se debe llevar el brazo es ligeramente diferente. En cualquiera de los casos, los valores obtenidos no se ven como valores altos ya que el máximo de ellos no supera los 4 N/m.

Además de estas gráficas también se muestra a continuación al igual que en el caso anterior el error en velocidad que se ha calculado a lo largo de la trayectoria para cada punto de la

interpolación. Estas graficas se pueden ver en la figura 4.9 tanto para la velocidad cartesiana como para la orientación en ángulos de Euler.

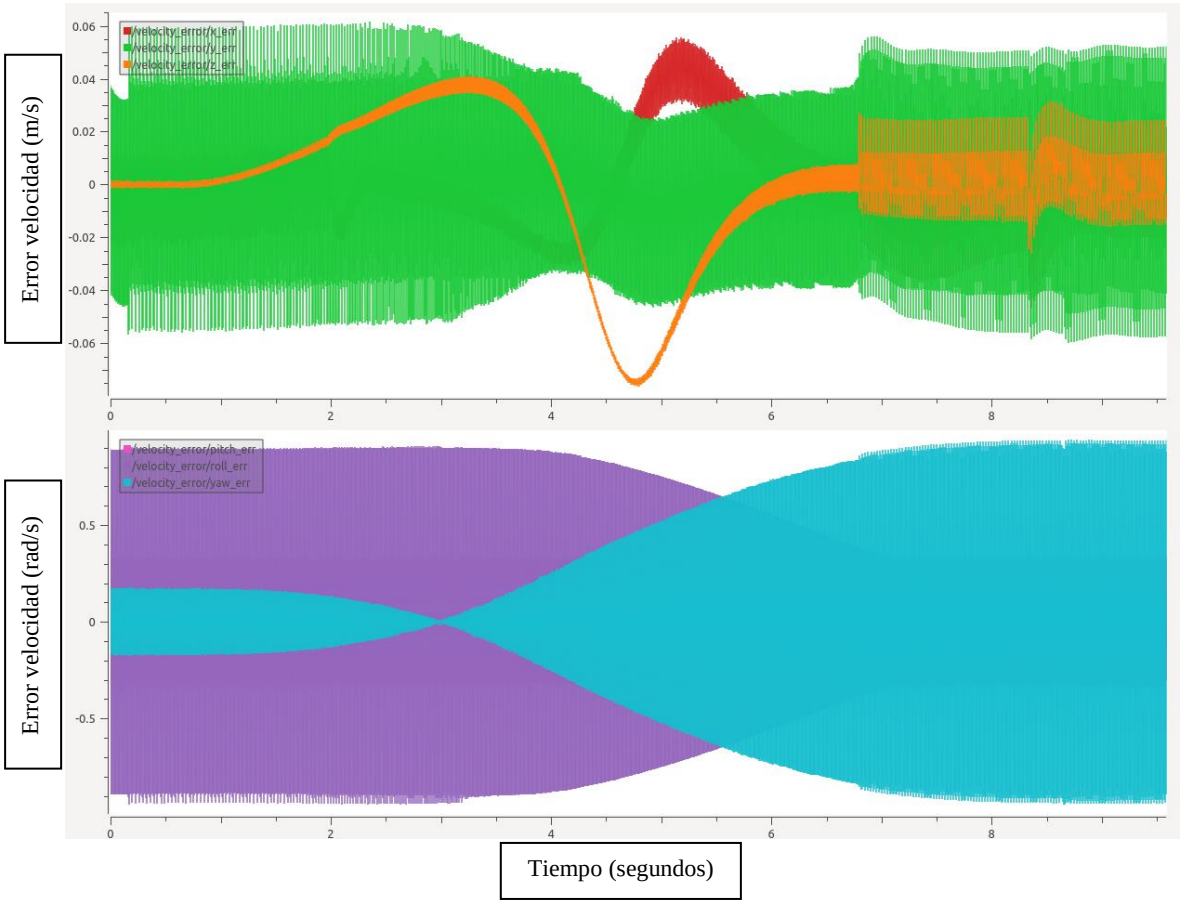

Figura 4.9: Error de velocidad cartesiana (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia.

Al igual que en el controlador anterior, se obtienen unos valores muy oscilantes y que varían de manera continua pero puesto que nunca llega a valores altos no se parece influenciar en la estabilidad del brazo. Por este motivo también se ha dejado una constante derivativa baja. A continuación, se mostrará las ultimas gráficas de este apartado en las que se observa cómo ha sido el seguimiento de la trayectoria trazada.

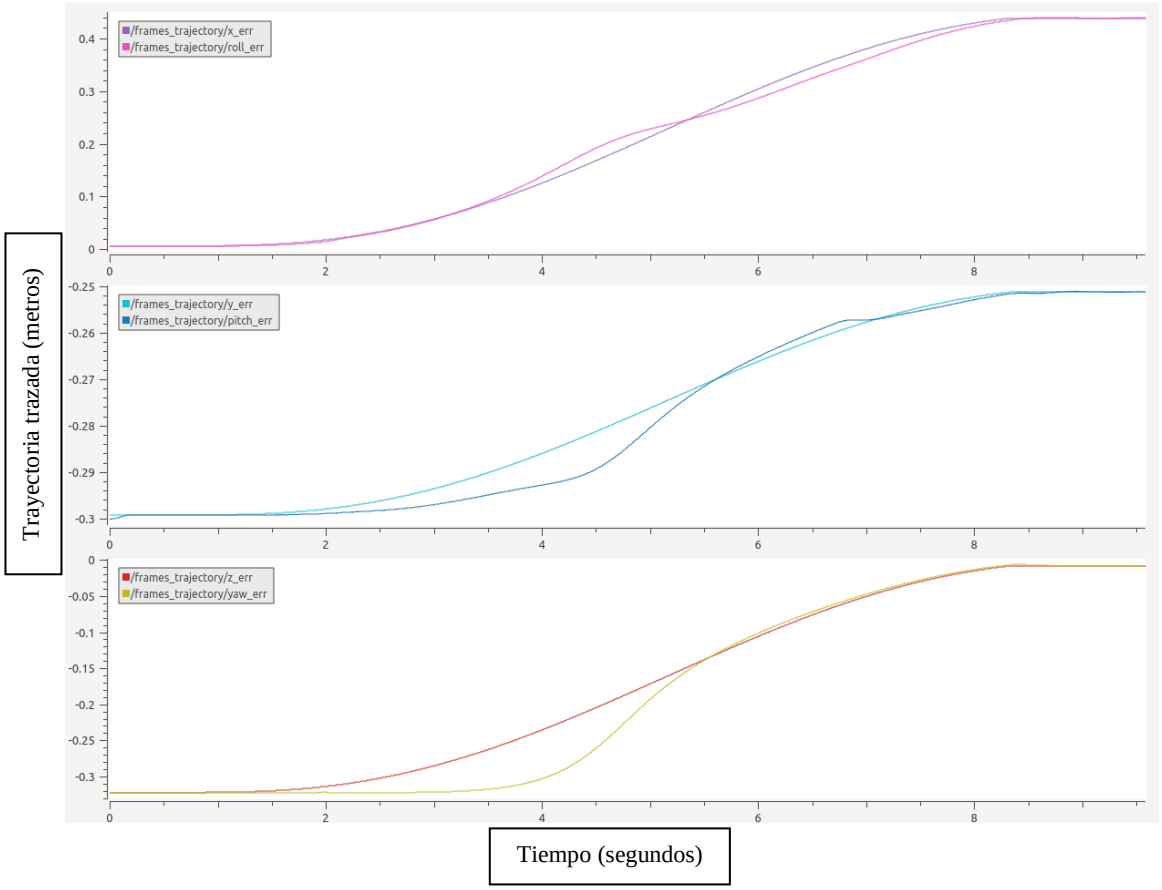

Figura 4.10: Seguimiento de la trayectoria cartesiana (Arriba coordenada X, en medio coordenada Y, abajo coordenada Z). **Fuente:** Elaboración propia.

En la figura anterior se puede observar que el seguimiento de esta trayectoria no ha sido tan exacto como en el controlador anterior, en este caso por el centro de la trayectoria se puede ver que para cada una de las coordenadas la trayectoria real trazada se desvía de la deseada, esto se cree que se piensa que es debido a que el brazo del robot puede pasar cerca de una singularidad del jacobiano y eso hace que la trayectoria trazada no sea perfecta. Esto se puede ver más claro durante la simulación observando el movimiento del brazo completo.

4.4.2. Controlador dinámico de trayectorias cartesianas utilizando localización mediante visión por computador teniendo en cuenta los esfuerzos del robot en la base

En este apartado se presentará un número de gráficas para poder analizar el comportamiento del controlador dinámico implementado en este trabajo. El comportamiento es muy similar al controlador explicado anteriormente ya que al estar trabajando en condiciones espaciales las trayectorias deben ser trayectorias lentas, por lo que las mejores características de un control dinámico no se ven de manera tan notoria. Para todas las gráficas presentadas en este subapartado se ha configurado el controlador con unos valores en las constantes de $Kp = 300$ y $Kv = 10$, además se ha utilizado una referencia de tiempo de 9 segundos.

Las primeras gráficas se presentan en la figura 4.11, en estas gráficas se puede observar cual es el desarrollo del error de control a lo largo del movimiento del brazo. Este error es el que se calcula entre las posiciones contiguas en la interpolación de la trayectoria.

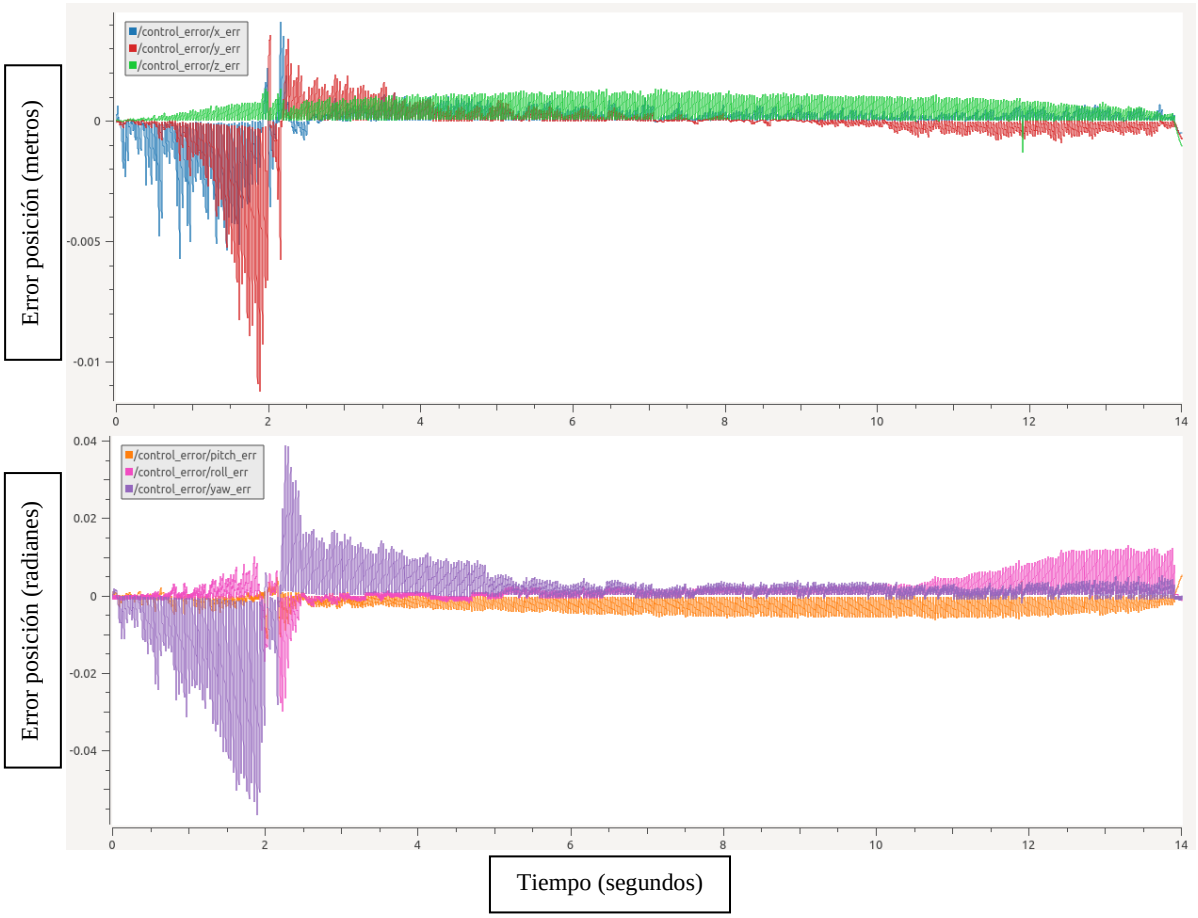

Figura 4.11: Error de control (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia.

Como se puede observar en las gráficas presentadas en la figura 4.11 el error siempre se encuentra entre unos valores muy bajos, también se puede ver que empieza en 0 por la misma razón que se ha explicado en los resultados del controlador anterior. Como una gran diferencia se puede ver que al comenzamiento del movimiento realizado el error pasa por un momento de inestabilidad en que el error es negativo y con un valor mayor que en el resto del movimiento. La razón de este pico de inestabilidad puede deberse a causa de las suposiciones hechas en la implementación del controlador dinámico o simplemente porque el movimiento se encuentra con una singularidad que no puede superar. Lo importante es que una vez pasan los dos primeros segundos el comportamiento consigue estabilizarse en un valor bastante bajo de error. Seguidamente en la figura 4.12 se puede ver el error hasta el punto final de la trayectoria.

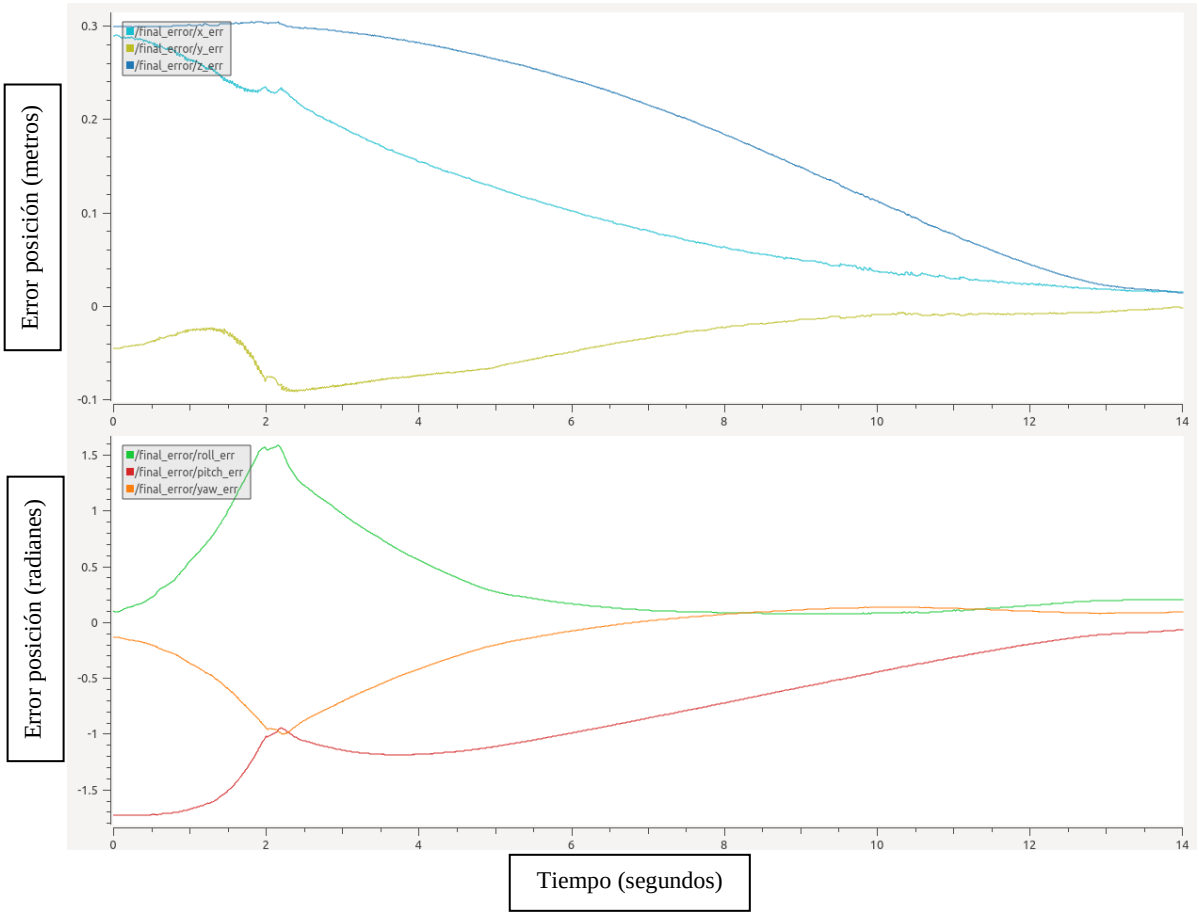

Figura 4.12: Reducción del error global a lo largo de toda la trayectoria (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia.

En esta figura presentada anteriormente se puede observar como el error finalmente es reducido hasta llegar a la tolerancia seleccionada en la configuración del controlador, en este caso hasta 0.015. También se puede apreciar que en los 2 primeros segundos se produce un cambio brusco en la curva debido también al leve momento de inestabilidad al principio y que se consigue estabilizar sin mayor problema. En la figura 4.13 se puede observar también el desarrollo del par calculado para cada articulación del brazo.

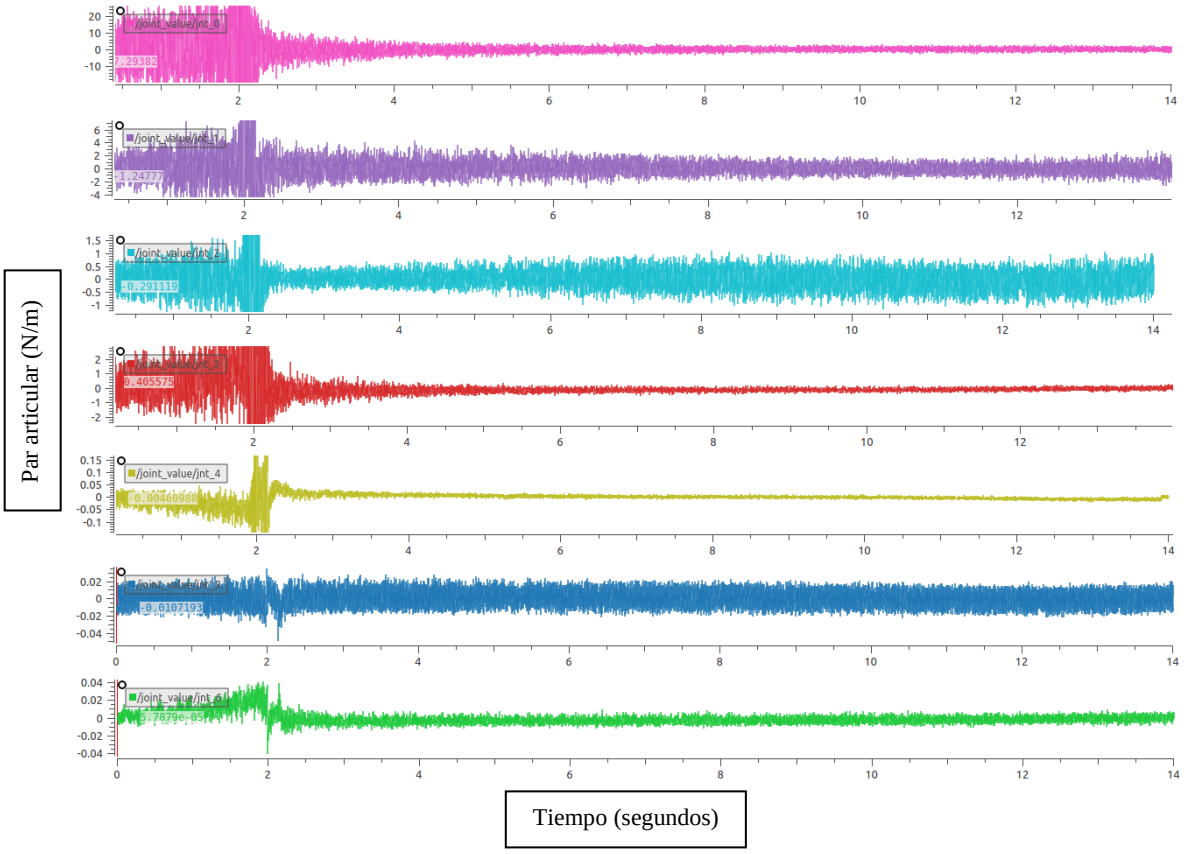

Figura 4.13: Pares articulares para cada una de las 7 articulaciones (orden descendente de 0 a 6). **Fuente:** Elaboración propia.

En la figura 4.13 se pueden comprobar todas las gráficas del valor de par enviado a cada articulación en cada momento, también se puede observar al igual que en las figuras anteriores que en los 2 primeros segundos los valores son un poco más inestables que durante el resto de la ejecución, pero tampoco se pueden ver valores excesivamente grandes. Únicamente se ha podido comprobar que existe un pico en el valor de los 2 segundos para todas las articulaciones a partir de donde el valor se estabiliza por completo. Después excepto para la primera articulación los valores no son muy altos.

A continuación, se pretende mostrar la evolución del error en velocidad a lo largo de todo el movimiento realizado. Este error es muy similar al error en velocidad mostrado en los dos controladores anteriores.

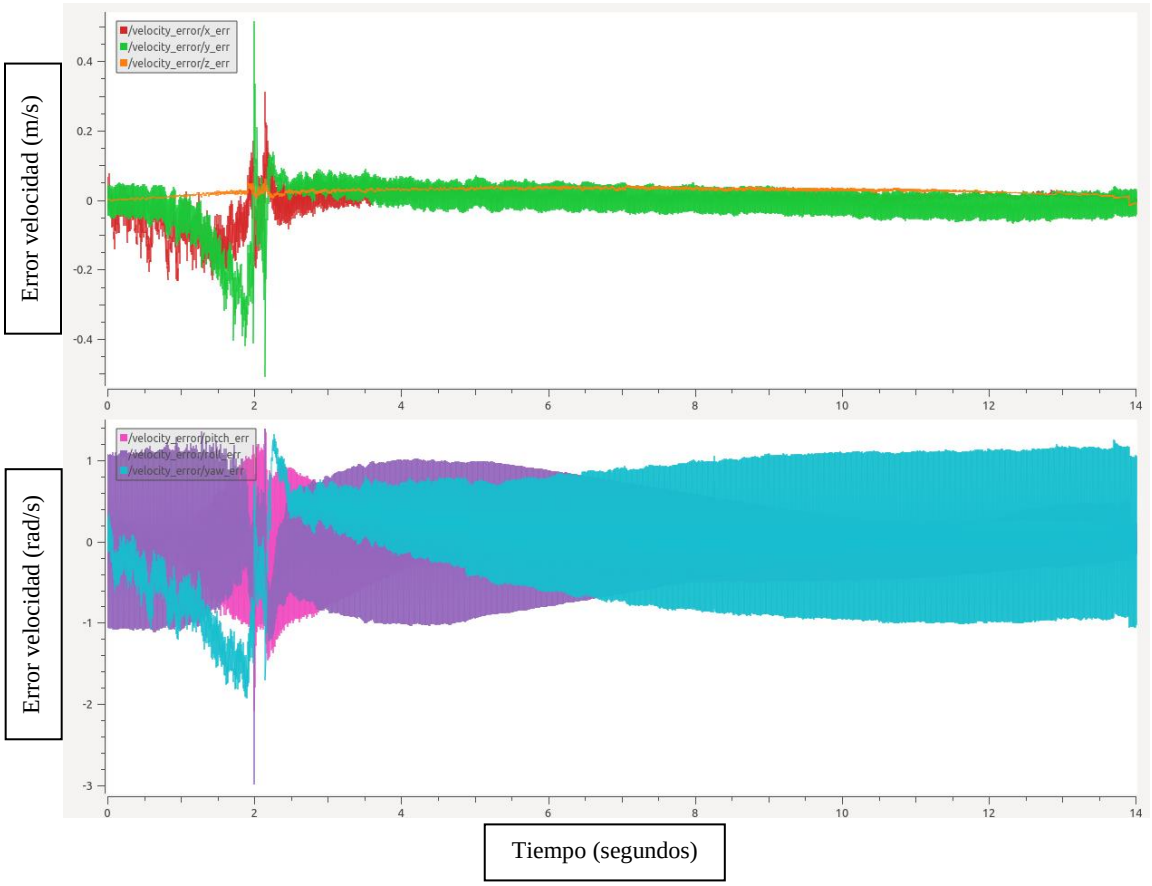

Figura 4.14: Error de velocidad cartesiana (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia.

Al igual que todas las gráficas mostradas en este controlador al pasar los 2 primeros segundos se puede observar una mejoría en la estabilidad, aunque los valores encontrados en ningún momento se pueden considerar excesivamente altos. Por último, para los resultados de este controlador se requiere mostrar la gráfica para comprar el seguimiento de la trayectoria deseada por el efector final utilizando este controlador, esto se puede ver en a figura 4.15.

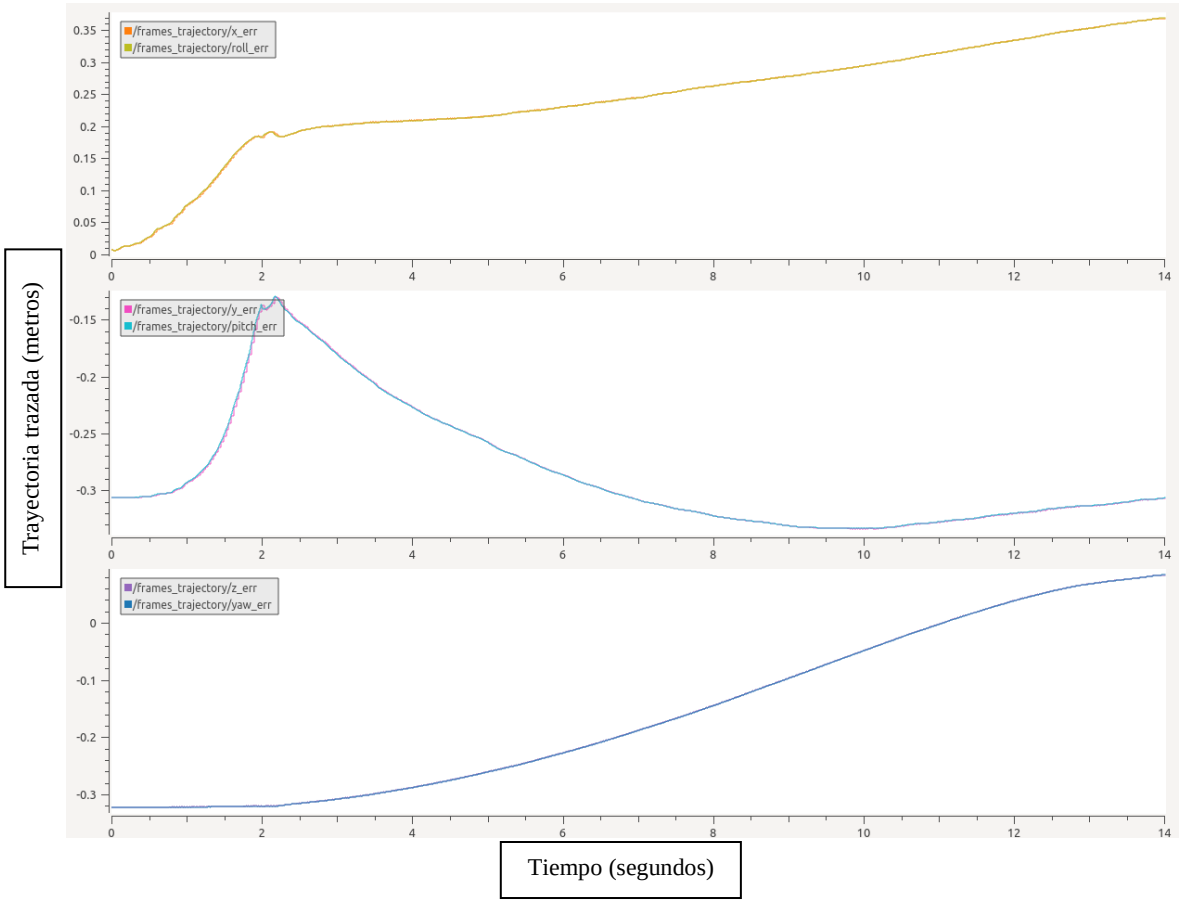

Figura 4.15: Seguimiento de la trayectoria cartesiana (Arriba coordenada X, en medio coordenada Y, abajo coordenada Z). **Fuente:** Elaboración propia.

En esta última figura se puede observar cómo el seguimiento de la trayectoria se realiza sin ningún problema puesto que las líneas en las 3 gráficas son totalmente coincidentes. Es cierto que la trayectoria calculada para la coordenada Y no concuerda con una trayectoria normal, de todas formas, viendo que el pico se encuentra en los dos segundos esto se supone que también es debido a ese leve momento de inestabilidad y al cálculo continuo de una nueva trayectoria. Sin embargo, tanto para la coordenada X como la coordenada Z se realiza un seguimiento ejemplar.

4.4.2.1. Adaptación sin visión por computador

Al igual que en los controladores anteriores también se han realizado las pruebas necesarias para comprobar el funcionamiento de la versión del controlador dinámico anterior que

funciona sin la necesidad de utilizar el código aruco. Todos los resultados serán presentados en este subapartado y para ello se ha configurado el controlador para que funcione con unas constantes $Kp = 100\ y\ Kv = 850$, además el tiempo de trayectoria que se ha utilizado para estas pruebas es de 9 segundos. Cabe mencionar que para que este controlador pudiera funcionar de manera correcta y reducir el error hasta el nivel de tolerancia deseado de 0.015 únicamente se ha podido utilizar estos valores de constantes. En cambio, en el controlador presentado en el apartado justo anterior podía funcionar con este ajuste de igual manera, pero el resultado era ligeramente mejor utilizando el ajuste presentado.

Primeramente, tal y como se ha estado haciendo hasta ahora se comentarán las gráficas del error de control a lo largo de toda la trayectoria. Estos datos se pueden observar en la figura 4.16.

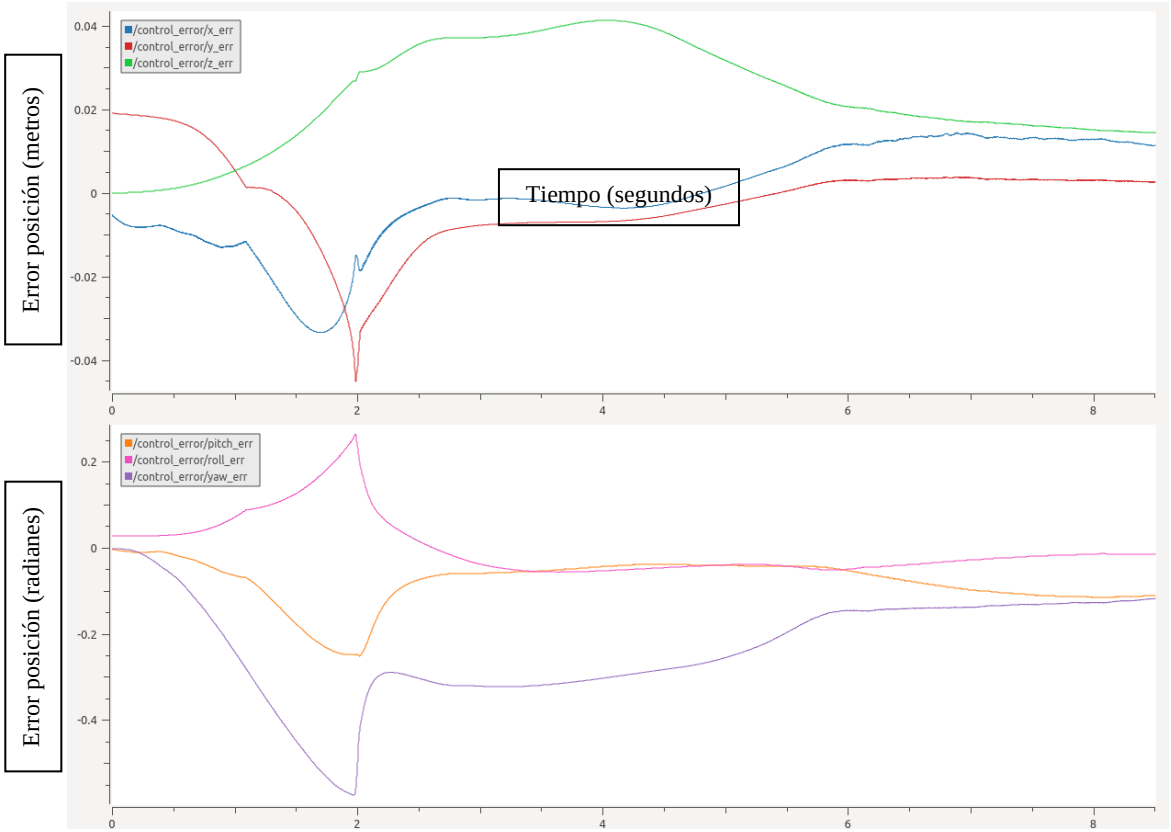

Figura 4.16: Error de control (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia.

En la figura 4.16 el error de control a lo largo de toda la trayectoria se pueda observar, en cierta manera más variante que el caso anterior, aunque los valores no sean valores altos como siempre. Además, cabe destacar que también se puede comprobar que durante los 2 primeros segundos se produce cierto momento de inestabilidad que enseguida es subsanado por el controlador. De igual manera el controlador es capaz de reducir el error y mantenerlo en un valor lo suficientemente bajo. A continuación, se puede observar las gráficas de la evolución del error con respecto del punto final de la trayectoria completa.

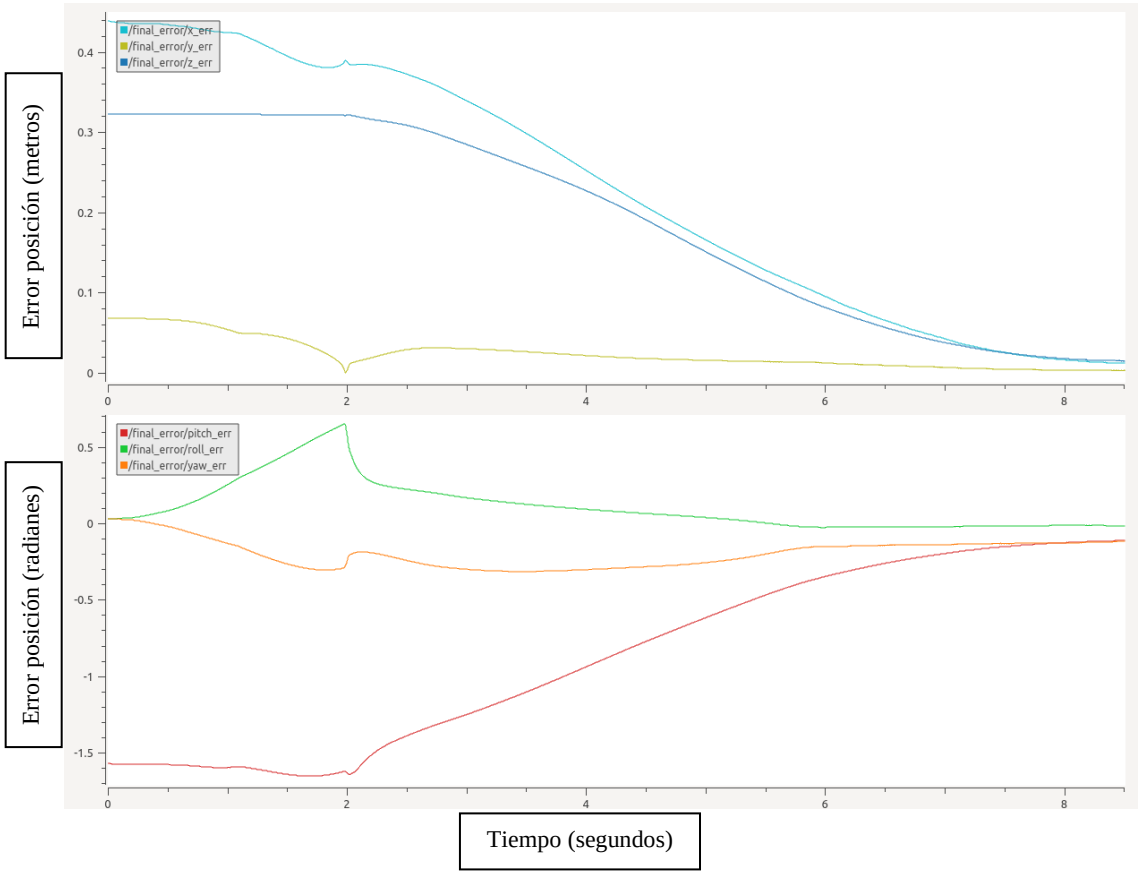

Figura 4.17: Reducción del error global a lo largo de toda la trayectoria (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia

En esta figura 4.17 se pueden ver las gráficas tanto para el error en coordenadas cartesianas como para la orientación del efector final en ángulos de Euler. En las dos gráficas se puede observar cómo finalmente el controlador es capaz de reducir el error en el espacio cartesiano sin mayor problema hasta 0 o hasta el valor de tolerancia deseado. También se puede ver que parece que el momento de inestabilidad tiene menor influencia en este controlador.

Seguidamente se puede pasar a ver los valores de par calculados para cada articulación del brazo robótico controlado en la figura 4.18.

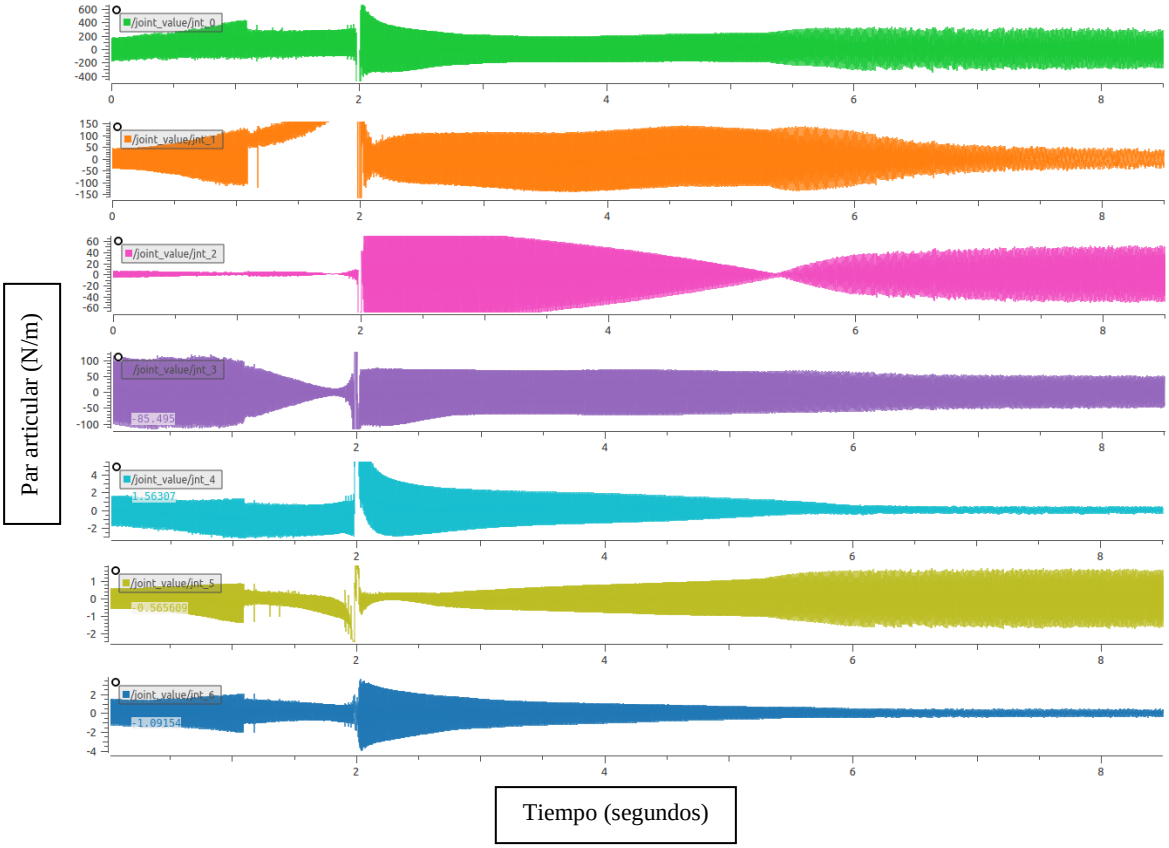

Figura 4.18: Pares articulares para cada una de las 7 articulaciones (orden descendente de 0 a 6). **Fuente:** Elaboración propia.

Para el caso de este controlador ha sido el único de los 4 presentados que se han podido observar unos valores altos de par. Sin embargo, esto no significa que el control realizado no es estable, esto es a causa de que en este caso se han necesitado de unas constantes con valores más altos que en los otros casos, pero igualmente los valores concuerdan con los del resto de controladores ya que como se puede observar las articulaciones 0 y 1 son las que dan los valores más altos, seguidamente la 2 y la 3 dan unos valores siempre más bajos y los demás todavía más bajos, además la forma y la evolución son muy similares. Para completar los resultados obtenidos también se debe mostrar cómo ha sido la evolución del error en velocidad obtenido con este controlador.

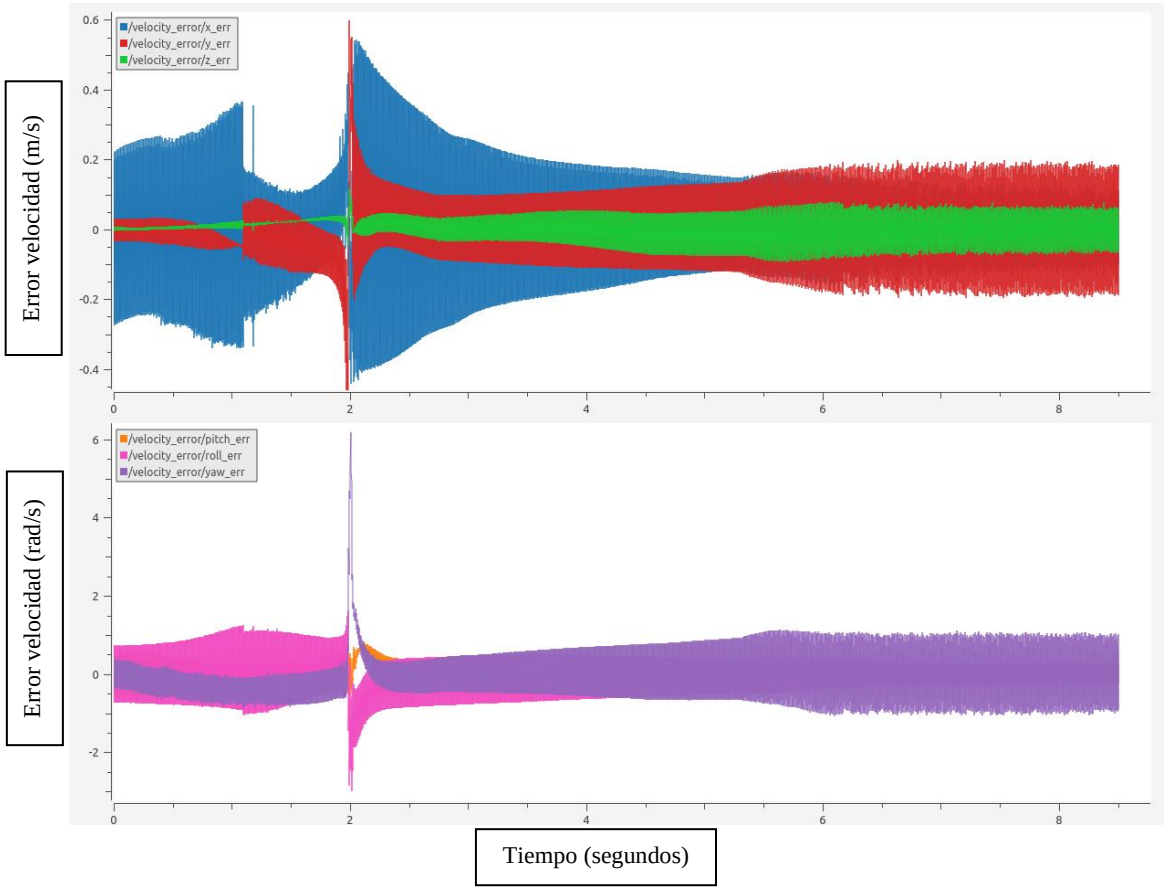

Figura 4.19: Error de velocidad cartesiana (posición cartesiana arriba y orientación en ángulos de Euler abajo). **Fuente:** Elaboración propia.

En este caso puesto que la ganancia derivativa de nuestro controlador es ciertamente superior a la proporcional se puede ver que el error en velocidad es un error mucho más constante que en el resto de los controladores, lo que es totalmente coherente. También podemos ver el pico de inestabilidad a los dos segundos igual que se ha ido viendo, pero a partir de este momento los valores no son extremadamente altos y siempre constantes.

También, como se ha hecho en los casos anteriores se han presentado unas gráficas en las que se puede observar de manera muy rápida y cómoda como es el seguimiento de la trayectoria deseada en el espacio de operaciones. Para ello se puede mirar la figura 4.20 en la cual se muestran las 3 variables cartesianas X, Y, Z del efector final a lo largo de todo el tiempo y al mismo tiempo la trayectoria calculada.

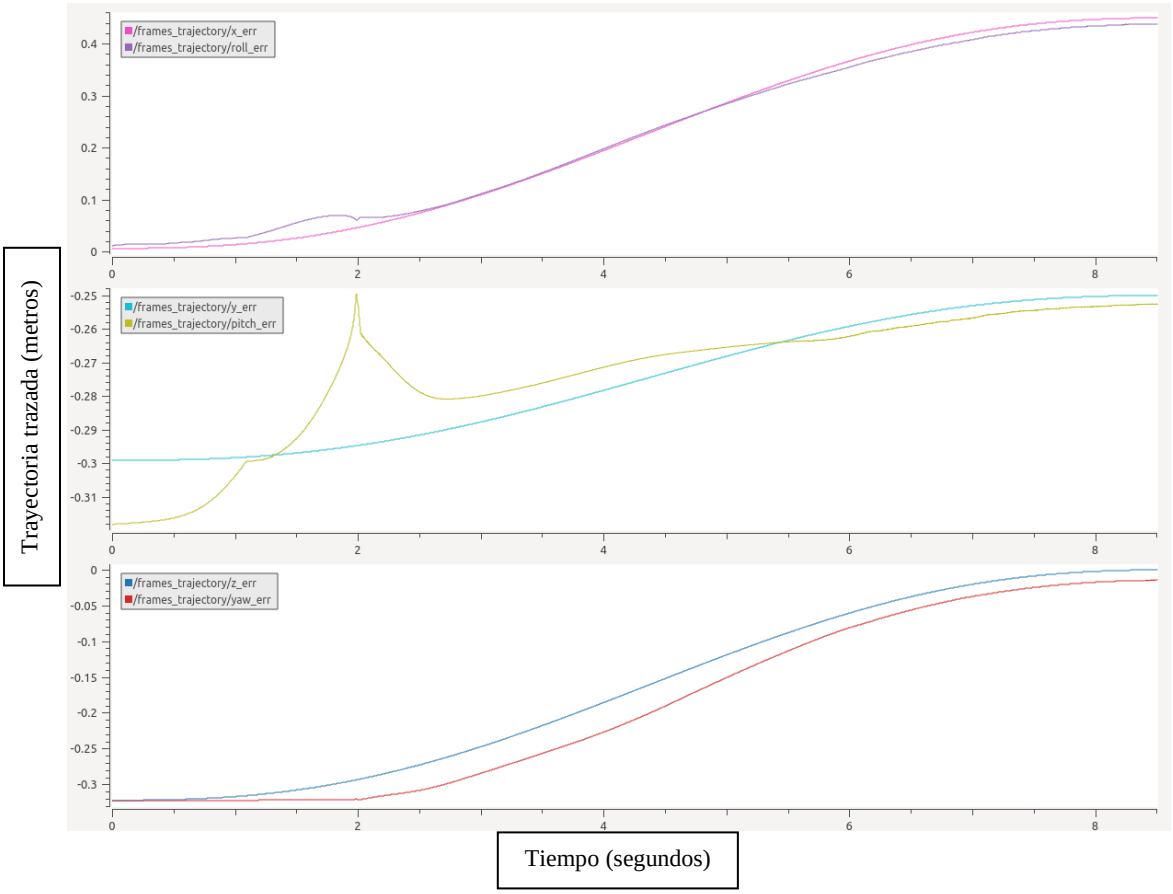

Figura 4.20: Seguimiento de la trayectoria cartesiana (Arriba coordenada X, en medio coordenada Y, abajo coordenada Z). **Fuente:** Elaboración propia.

En la figura 4.20 se puede ver cómo a causa de la inestabilidad producida los dos primeros segundos el seguimiento de la trayectoria en algunas coordenadas es extremadamente pobre, sin embargo, al final del trayecto siempre consigue seguir la trayectoria y estabilizarse como se puede ver en la coordenada X y Z.

4.5. Resultados de entorno de simulación y tarea realizada

Este apartado se ha realizado con el fin de mostrar al lector el resultado de todo lo explicado a lo largo de este trabajo. En este apartado se mostrarán diferentes figuras en las que será posible observar el aspecto final de la simulación y entorno de pruebas creado en Gazebo con el robot Talos Astronauta y la estación espacial internacional.

Además de esto, también se intentará mostrar una serie de figuras de cómo es el movimiento del brazo realizado en todas las pruebas anteriores. Para ello se han tomado varias fotos del robot moviendo el brazo hacia la posición deseada y además varias fotos de como se ha realizado el agarre del asa final.

Primeramente, en la figura 4.21 se puede ver cual es resultado de la simulación creada para establecer un buen entorno de pruebas que sea lo más realista posible.

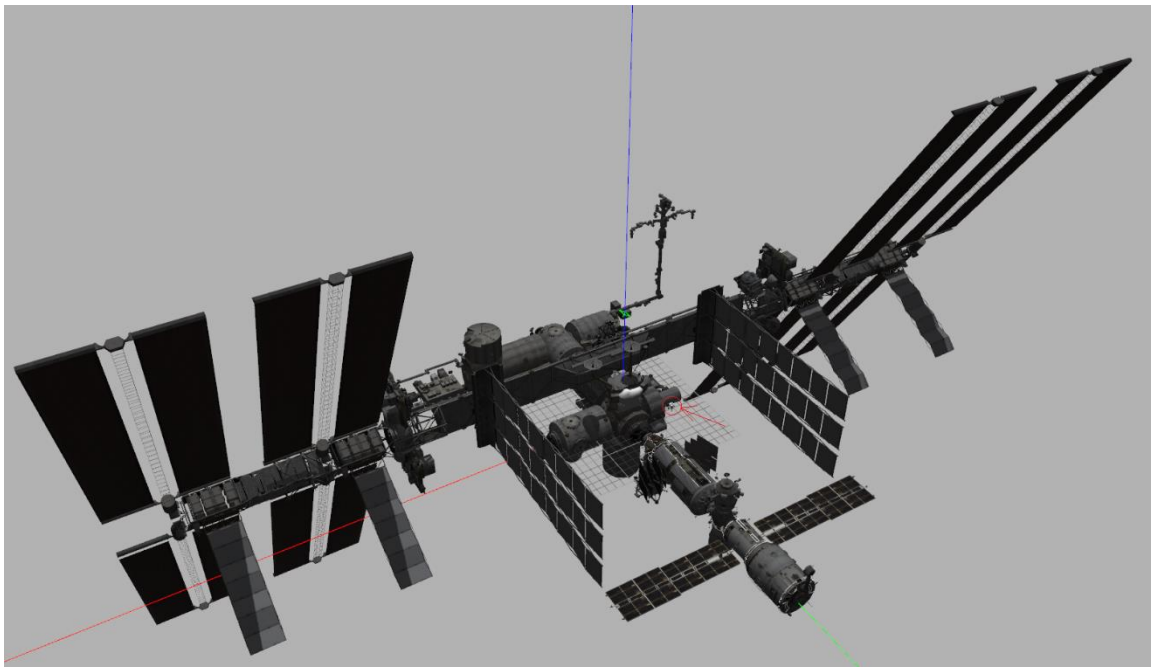

Figura 4.21: Simulación en Gazebo con la estación espacial internacional completa y el robot Talos Astronauta. **Fuente:** Elaboración propia

En la figura 4.21 anteriormente mostrada se puede ver en el entorno de simulación que las dimensiones de la estación espacial son desmesuradas en comparación con las del robot diseñado al igual que lo son en comparación con las de un humano. Para que se pudiera observar completamente la estación espacial se ha debido hacer una foto desde un punto tan lejano que el robot ha quedado prácticamente inapreciable, por ello este se ha señalado con una flecha y un círculo de color rojo.

Seguidamente se pasará a mostrar las figuras que pretenden ayudar al lector a visualizar el tipo de movimiento realizado por el brazo del robot para agarrar un asa de la estación espacial y así poder entender mejor las gráficas mostradas en los apartados anteriores. La primera figura 4.22 muestra la posición inicial del Talos astronauta.

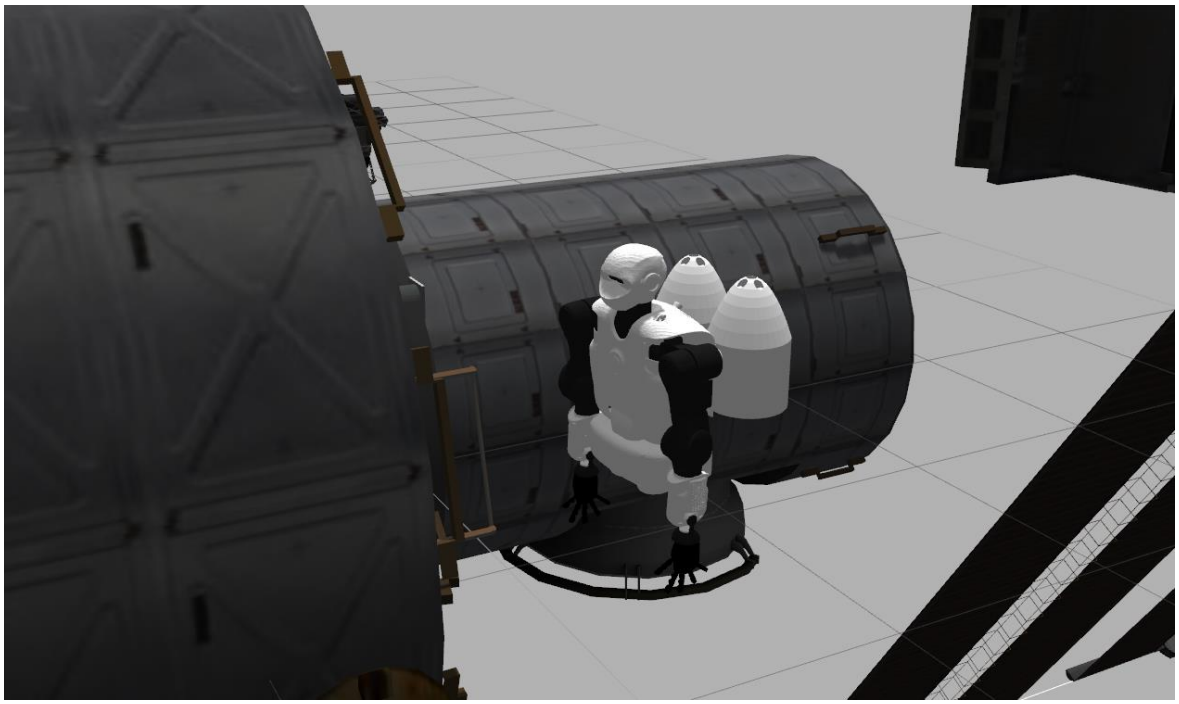

Figura 4.22: Posición inicial del Talos Astronauta antes de comentar el movimiento.

Fuente: Elaboración propia.

En la tarea general a realizar el próximo movimiento será que el robot mueva la cabeza correctamente para poder observar el código Aruco y así comenzar el movimiento. Esto se puede ver en la figura 4.23.

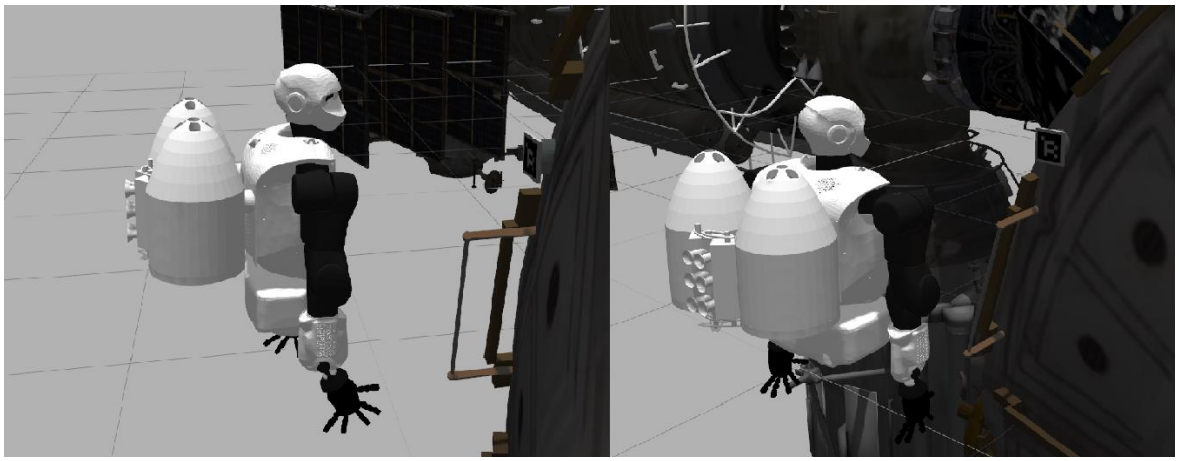

Figura 4.23: Robot Talos Astronauta moviendo la cabeza para observar el código Aruco.

Fuente: Elaboración Propia.

En las siguientes figuras se mostrará cuál es el movimiento realizado por el brazo del robot hasta alcanzar la posición deseada donde deberá comenzar a cerrar la mano para realizar un correcto agarre.

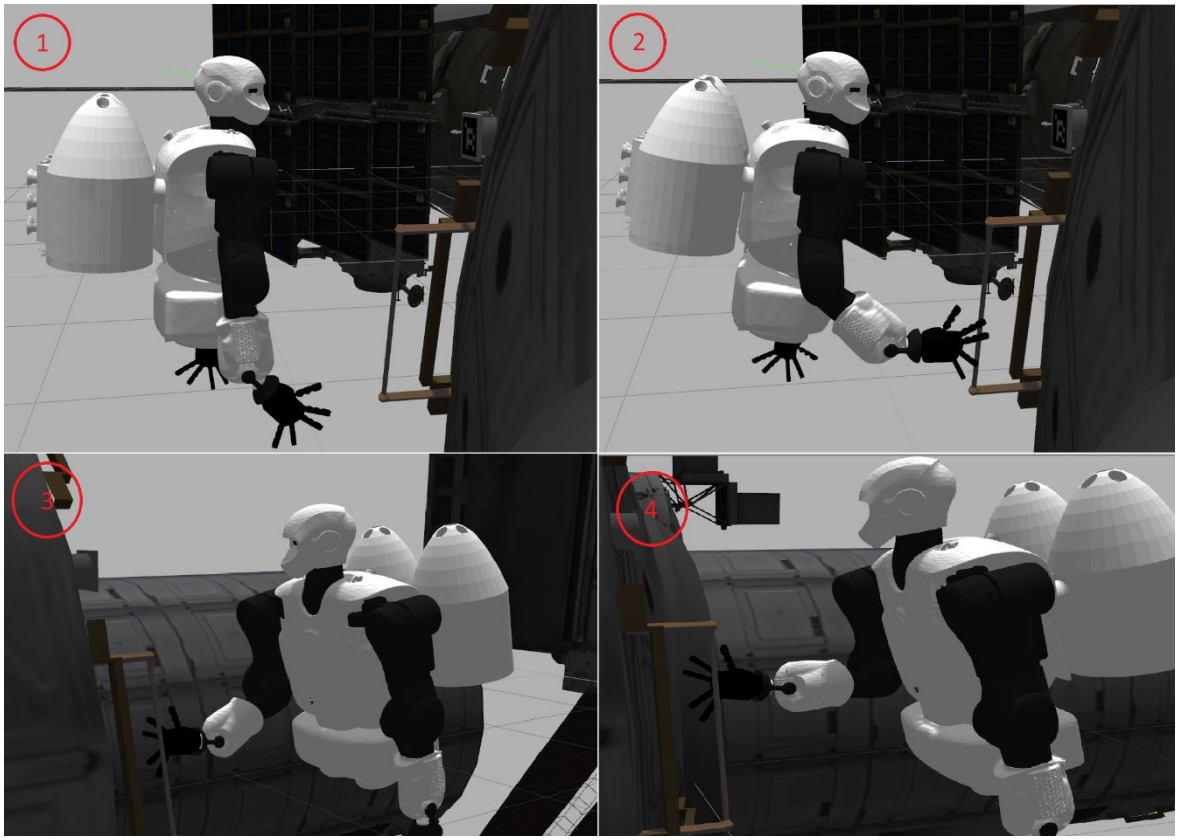

Figura 4.24: Movimiento realizado por el brazo del robot Talos Astronauta. **Fuente:** Elaboración propia.

Una vez realizado el movimiento, el controlador avisará de que se ha llegado a la posición deseada según la tolerancia configurada y el programa de la tarea comenzará a realizar el agarre del asa. Este proceso de cierre y agarre con la mano se puede ver en la siguiente figura 4.25.

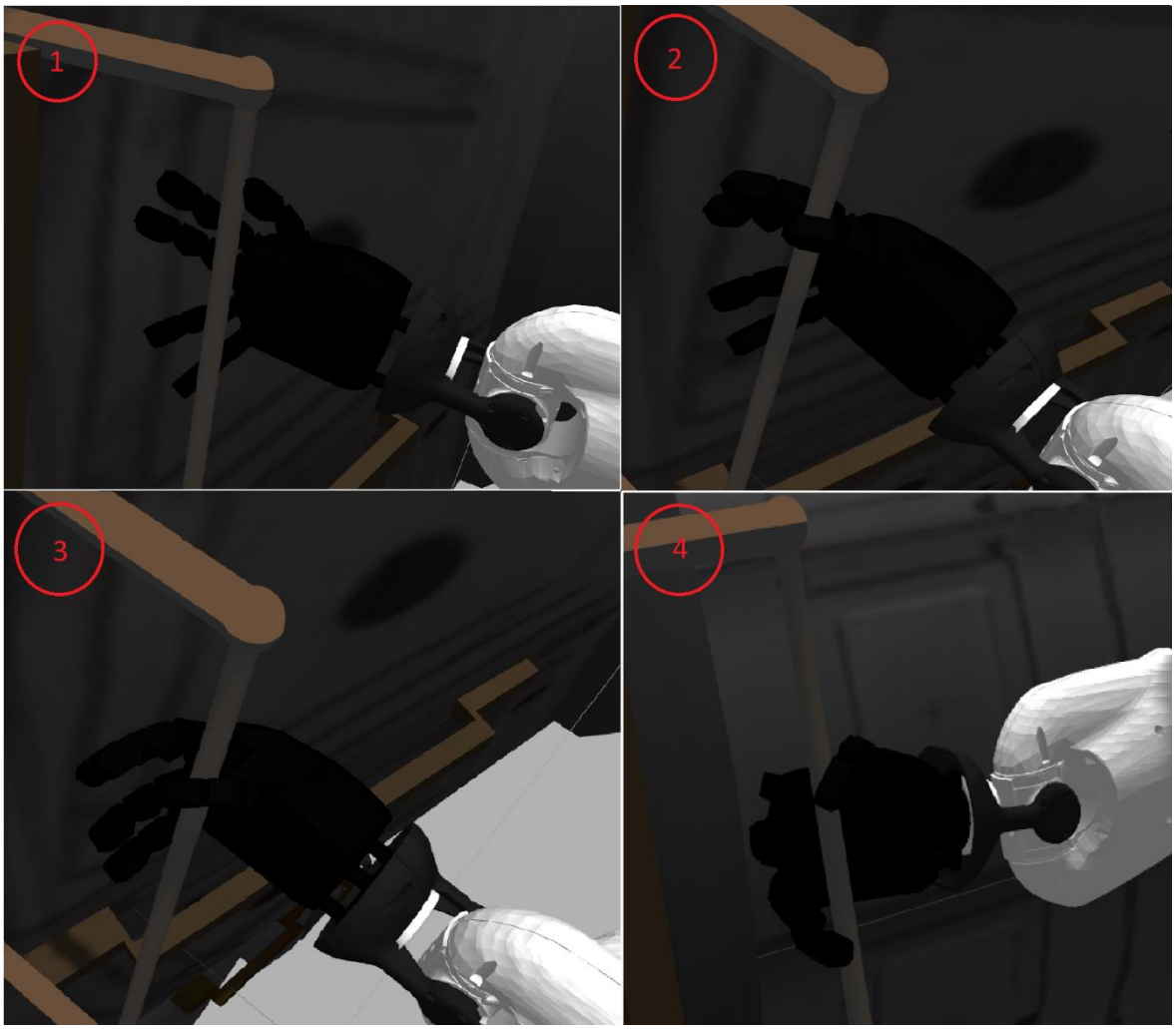

Figura 4.25: Proceso de agarre del asa con la mano del Talos Astronauta. **Fuente:** Elaboración Propia.

Por último, para finalizar con el cumplimiento de la tarea se va a mostrar en la figura 4.26 el estado del robot Talos Astronauta una vez que ha cogido el asa y ha estabilizado su posición. Se puede observar en las figuras mostradas que el resultado final es suficientemente estable y el robot no entra en ningún estado de inestabilidad al agarrar el asa por las fuerzas de reacción que pudieran ocasionarse.

Figura 4.26: Talos Astronauta sujetando el asa de la estación espacial y estabilizándose.
Fuente: Elaboración Propia.

Como se ha podido observar en todas las figuras mostradas en este apartado, el robot Talos Astronauta es capaz de completar la tarea asignada de agarrar un asa de la estación espacial internacional, consiguiendo así estabilizar su propio cuerpo durante la realización de diferentes trabajos que pueden ser realizados con el brazo restante.

5. Conclusiones

El trabajo realizado ha conseguido completar los objetivos propuestos desde el principio, se ha conseguido adaptar un robot humanoide a las condiciones de trabajo en la estación espacial y preparar una simulación en la que el robot diseñado pueda trabajar en el entorno de la estación espacial internacional. Además, se ha conseguido proponer dos controladores, uno cinemático y otro dinámico con los que se obtiene una solución al problema del control de brazos robóticos en robots flotantes en el espacio. Además de los dos controladores específicos para este problema también se han implementado otros 2 controladores que pueden funcionar de manera independiente sin proporcionar una solución a este problema como punto de partida para diferentes usuarios que pretendan aumentar las capacidades de este trabajo. Además de los controladores propuestos y explicados en el TFG se han implementado otros varios pero que no cumplen con los requisitos y objetivos del trabajo pero que también estarán disponibles para quien quiera usarlos en el futuro.

Finalmente, sobre los resultados obtenidos durante el comportamiento de los 4 controladores se puede decir que es un resultado coherente y con sentido. Los controladores son capaces de seguir una trayectoria cartesiana y de ajustar esta mediante el cálculo de nuevas trayectorias. Los errores son siempre relativamente constantes y con valor bajo lo que da un resultado más estable y con menos perturbaciones. Además, los pares calculados para enviar a las articulaciones siempre están cerca de un valor similar para todos los casos.

Por estas razones se puede decir que los resultados obtenidos son buenos y conformes con lo que se esperaba al plantear los objetivos del trabajo. Pero como en todos los trabajos siempre existen diferentes detalles o trabajos futuros que pueden mejorar y añadir valor al trabajo realizado. Los que se plantean en el siguiente punto solamente son unos cuantos de los que se cree que en base a las posibles debilidades y estado del arte actual en el tema podrían aumentar el valor del trabajo realizado en un futuro.

5.1. Trabajos futuros

El trabajo realizado se puede ver como una puesta en marcha y primeros pasos de lo que podría ser un gran entorno de simulación y pruebas. La estructura para crear diferentes controladores está totalmente creada y ha sido explicada en este TFG por lo que añadir nuevos controladores tanto cinemáticos como dinámicos y analizar sus comportamientos no debería ser una tarea excesivamente difícil, pero sí que podría ser interesante para desarrollar una primera versión de los controladores que se quieran utilizar en un futuro en este tipo de robots.

Una de las cosas que también puede aumentar en gran valor el trabajo sería diseñar un plugin de gazebo para controlar el movimiento de la base del robot provocado por las inercias. Esto debería realizarse de manera necesaria utilizando el controlador dinámico explicado en el trabajo, se debería añadir los cálculos necesarios para obtener el valor de la velocidad de la base en cada momento y permitir que el controlador se comunique de alguna manera con el plugin para simular las fuerzas necesarias para corregir la posición de la base a medida que el robot mueve el brazo. Esto solamente es un planteamiento posible que se ha pensado, pero por razones de tiempo no se ha podido implementar esta característica, por lo que se ha

decidido dejar planteada para que futuros usuarios de la aplicación la implementen como mejor decidan.

Como posibles trabajos futuros también se ha pensado que sería interesante realizar diferentes programas para aumentar las capacidades de manejo de las manos robóticas y dotar así al robot con una mayor diversidad de acciones que pudiera realizar en el entorno de la estación espacial. Por último, si los futuros usuarios desean mejorar u optimizar cualquier parte del código ya realizado también sería una tarea que aumente valor al proyecto y sería bien recibida en cualquier momento.

Las capacidades que ofrece este trabajo para mejorar y ayudar a la realización de diferentes pruebas son extremadamente amplias, por lo que se espera que una gran cantidad de gente se interese por él y piense personalmente cómo podría mejorarlo.

6. Referencias

[1] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. Robotics: Modelling, Planning and Control. Springer-Verlag, 2010.

[2] Antonio Barrientos, Luis F. Peñín, Carlos Balaguer, and Rafael. Aracil. Fundamentos de Robótica. McGraw-Hill, 2007.

[3] Gao, Y., & Chien, S. (2017, 06 28). Review on space robotics: Toward top-level science through space exploration. Science Robotics, 2(7), 12. 10.1126/scirobotics.aan5074

[4] Yoshida K, Nenchev D, Ishigami G, Tsumaki Y. Space robotics. In The International Handbook of Space Technology. Springer Berlin Heidelberg. 2014. p. 541-573

[5] Wilfried Ley, Klaus Wittmann, Willi Hallmann. Handbook of Space Technology. Wily Editors. 2009.

[6] Jasiobedzki,, P., & Anders, C. (1998). Computer Vision for Space Robotics: Applications, Role and Performance. IFAC Proceedings Volumes, 31(33), 95-102. https://doi.org/10.1016/S1474-6670(17)38393-3

[7] Pérez Alepuz, J. (2017). DYNAMIC VISUAL SERVOING OF ROBOT MANIPULATORS: OPTIMAL FRAMEWORK WITH DYNAMIC PERCEPTIBILITY AND CHAOS COMPENSATION (Doctor). Universidad de Alicante.

[8] Shan, M., Guo, J., & Gill, E. (2016). Review and comparison of active space debris capturing and removal methods. Progress in Aerospace Sciences, 80, 18–32. http://doi.org/10.1016/j.paerosci.2015.11.001

[9] Graham, A. R., & Kingston, J. (2015). Assessment of the commercial viability of selected options for on-orbit servicing (OOS). Acta Astronautica, 117, 38–48. http://doi.org/10.1016/j.actaastro.2015.07.023

[10] Felicetti, L., Gasbarri, P., Pisculli, A., Sabatini, M., & Palmerini, G. B. (2016). Design of robotic manipulators for orbit removal of spent launchers' stages. Acta Astronautica, 119, 118–130. http://doi.org/10.1016/j.actaastro.2015.11.012

[11] De Rosa, D., & Curti, F. (2006). Visual Monitoring of Space Rendezvous: A Structure From Motion Problem. In AIAA/AAS Astrodynamics Specialist Conference and Exhibit. American Institute of Aeronautics and Astronautics. http://doi.org/doi:10.2514/6.2006-6762

[12] Song, L., Li, Z., & Ma, X. (2014). Autonomous rendezvous and docking of an unknown tumbling space target with a monocular camera. In Proceedings of 2014 IEEE Chinese Guidance, Navigation and Control Conference (pp. 1008–1013). IEEE. http://doi.org/10.1109/CGNCC.2014.7007346

[13] Aghili, F., Kuryllo, M., Okouneva, G., & English, C. (2011). Fault-Tolerant Position/Attitude Estimation of Free-Floating Space Objects Using a Laser Range Sensor. IEEE Sensors Journal, 11(1), 176–185. http://doi.org/10.1109/JSEN.2010.2056365

[14] Flores, A., Ma, O., Pham, K., & Ulrich, S. (2014). A review of space robotics technologies for on-orbit servicing. Progress in Aerospace Sciences. Elsevier Ltd.

[15] Jin, M., Yang, H., Xie, Z., Sun, K., & Liu, H. (2013). The ground-based verification system of visual servoing control for a space robot. In 2013 IEEE International Conference on Mechatronics and Automation (pp. 1566–1570). IEEE. http://doi.org/10.1109/ICMA.2013.6618147

[16] PAL Robotics, https://pal-robotics.com

[17] Kinematics and dynamics library (kdl) documentation. http://docs.ros.org/

kinetic/api/orocos_kdl/html/index.html

[18] J. Carpentier et al., "The Pinocchio C++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," 2019 IEEE/SICE International Symposium on System Integration (SII), 2019, pp. 614-619, doi: 10.1109/SII.2019.8700380.

[19] Eigen: C++ library for algebraic calculus. documentation. http://eigen.tuxfamily.

org/index.php?title=Main_Pag

[20] R.T.Azuma,Asurveyofaugmentedreality,Presence6(1997)355–385.

[21] H.Kato,M.Billinghurst,MarkertrackingandHMDcalibrationforavideo-

based augmentedrealityconferencingsystem,in:InternationalWorkshopon

AugmentedReality,1999,pp.85–94.

[22] V.Lepetit,P.Fua,Monocularmodel-based3Dtrackingofrigidobjects:a

survey,in:FoundationsandTrendsinComputerGraphicsandVision,2005,

pp. 1–89.

[23] B. Williams,M.Cummins,J.Neira,P.Newman,I.Reid,J.Tardós,Acomparison

of loopclosingtechniquesinmonocularSLAM,RoboticsandAutonomous

Systems57(2009)1188–1197.

[24] Ros aruco package documentation. http://wiki.ros.org/aruco_ros.

[25] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., & Marín-Jiménez, M. J. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition, 47(6), 2280–2292.

[26] Foote, T. (2013). tf: The transform library. 2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA). IEEE.

[27] Controlador Jorge

[28] Barata, J. C. A., & Hussein, M. S. (2011). The Moore-Penrose pseudoinverse. A tutorial review of the theory. Recuperado de http://arxiv.org/abs/1110.6882.

[29] Ros Services. http://wiki.ros.org/Services

[30] Joint Trayectory Controller. http://wiki.ros.org/joint_trajectory_controller

[31] Plotjuggler. http://wiki.ros.org/plotjuggler