

**UNIVERSIDAD TECNOLÓGICA DE PANAMÁ  
FACULTAD DE MECÁNICA**



**DESARROLLO DE ENTORNO EN ROS2 PARA EL CONTROL Y  
SIMULADO DE UN ROBOT CONTINUO ACCIONADO POR TENDONES**

**PRESENTADO POR:**

**GABRIEL TUZLACI**

**E-8-155428**

**DIRECTORA:**

**DRA. ILKA BANFIELD**

**TRABAJO DE GRADUACIÓN PRESENTADO PARA OPTAR AL  
GRADO DE INGENIERÍA MECÁNICA**

**PANAMÁ, 2024**

**UNIVERSIDAD TECNOLÓGICA DE PANAMÁ**  
**FACULTAD DE MECÁNICA**  
**DESARROLLO DE ENTORNO EN ROS2 PARA EL CONTROL Y**  
**SIMULADO DE UN ROBOT CONTINUO ACCIONADO POR TENDONES**

Dra. Ilka Banfield

Directora

Mgr. Carlos Plazaola

Jurado Calificador

Dr. Huberto Rodríguez

Jurado Calificador

## **Tabla de Contenido**

1	Introducción.....	7
1.1	Definición del problema.....	7
1.2	Objetivo General .....	13
1.3	Objetivos Específicos .....	13
1.4	Delimitaciones .....	15
1.5	Estructura del trabajo .....	17
2	Modelado de la varilla flexible .....	18
2.1	Estado del arte .....	18
2.2	Teoría de varillas de Cosserat.....	19
2.3	Desarrollo del paquete de simulación Elastica .....	31
3	Modelo de fuerzas y momentos para accionamiento de tipo tendón.....	37
3.1	Estado del arte .....	37
3.2	Implementación en PyElastica .....	39
3.2.1	Funcionamiento de PyElastica .....	39
3.2.2	Planteamiento y desarrollo del módulo de accionamiento por tendones.....	43
3.2.3	Estructura como módulo de forzado externo para el accionamiento por tendones .....	50
3.3	Experimentos y validación.....	60
3.3.1	Objetivos de los experimentos .....	61
3.3.2	Calibración de la simulación .....	62
3.3.3	Materiales .....	63

3.3.4	Experimento No. 1 .....	64
3.3.5	Experimento No. 2 .....	69
3.3.6	Experimento No. 3 .....	72
4	Implementación en ROS2 .....	77
4.1	Nodo de interfaz gráfica (GUI) .....	80
4.1.1	Integración en el sistema .....	80
4.1.2	Funcionamiento.....	82
4.2	Nodo de simulación.....	87
4.2.1	Integración en el sistema .....	87
4.2.2	Funcionamiento.....	90
4.3	Nodo de visualización .....	92
4.3.1	Integración en el sistema .....	93
4.3.2	Funcionamiento.....	94
4.4	Nodo de control.....	96
4.4.1	Integración en el sistema .....	97
4.4.2	Funcionamiento.....	98
4.5	Nodo de ajuste de ganancias .....	103
4.5.1	Integración en el sistema .....	104
4.5.2	Funcionamiento.....	105
4.6	Resultados y análisis.....	110
4.6.1	Caso de lazo abierto .....	110
4.6.2	Caso de control retroalimentado .....	110
4.6.3	Caso de ajuste de ganancias para el control retroalimentado ....	115

5	Conclusión.....	117
6	Anexos .....	119
6.1	Mediciones y simulaciones del experimento No. 3.....	119
6.2	Resultados de los ensayos de control.....	123
6.3	Código desarrollado e implementado.....	127
7	Referencias bibliográficas .....	128

## **DEDICATORIA**

Dedico este trabajo a mis padres, por su amor y apoyo incondicional; a mis profesores, por su sabiduría y guía; y a mis amigos, por su aliento y compañía.

Cada uno de ustedes ha sido una pieza clave en mi crecimiento, tanto académico como personal, y estoy eternamente agradecido por todo lo que han hecho por mí a lo largo de este proceso.

# 1 Introducción

## 1.1 Definición del problema

La robótica es un área de la ingeniería que comprende una gran cantidad de temas y disciplinas. Tiene como meta la creación de máquinas inteligentes que pueden asistir a sus usuarios humanos en la realización de tareas normalmente repetitivas, de manera más precisa y acertada. Existen diversos tipos de robots, diseñados para cumplir tareas específicas con alta precisión y rapidez, los cuales se utilizan hoy en día para lograr tareas de manufactura en general, ensamble, control de posición o trayectoria, agricultura, minería e incluso confección de alimentos, entre otras industrias. Estas aplicaciones para la robótica han sido desarrolladas a lo largo de los años, yendo desde el uso de Unimate en 1958 para la ayuda en la línea de ensamblaje de General Motors, hasta el uso de robótica suave para realizar cirugías teleoperadas en pacientes (Hockstein et al., 2007) utilizando sistemas como Da Vinci Robot System, uno de los más reconocidos actualmente (Burgner-Kahrs et al., 2015).

Tradicionalmente, la robótica consiste en series de eslabones rígidos, conectados por uniones que usualmente tienen solamente un grado de libertad (Robinson & Davies, 1999). Estos robots suelen ser puestos en movimiento por actuadores que, de manera similar, suelen tener solamente un grado de libertad. Si bien se pueden combinar los eslabones y actuadores para crear robots que pueden cumplir tareas más complejas con varios grados de libertad, estos pueden llegar a convertirse en máquinas muy pesadas, muy grandes o simplemente muy rígidas para ciertas tareas. Sin embargo, para tareas que requieran de rapidez de ejecución y precisión, este tipo de robot “tradicional” se destaca.

En contraste, para ambientes complejos y tareas donde se requiere mejor maniobrabilidad (tales como ambientes hostiles, delicados, suaves, inaccesibles, entre otros), delicadeza o versatilidad, estos tipos de robot rígidos tienden a

disminuir su desempeño. Para abordar esta limitación inherente a los sistemas robóticos convencionales, se puede plantear el uso de elementos continuos, que, por definición, ofrecen una cantidad infinita de grados de libertad. La búsqueda de elementos continuos controlables ha estado creciendo recientemente, teniendo como meta la resolución de problemas de maniobrabilidad en ambientes complejos (Sun et al., 2019), menor rigidez de los elementos, manipulación de objetos suaves, entre otras aplicaciones.

Existen varios antecedentes, casos de estudio y aplicaciones actuales para la robótica continua, tales como:

- La evacuación de hemorragia intracerebral mediante el uso de un robot continuo del tipo tubo concéntrico (Burgner et al., 2013). El robot utilizado consiste en dos tubos, el primero de ellos siendo el exterior que es recto para acceder al coágulo de sangre en el cerebro de manera lineal, mientras que el segundo tubo interno para aspiración es recto con una sección curva para remover la hemorragia desde adentro. La actuación de este robot se hace externamente.
- Cirugía endoscópica funcional para las fosas nasales (Burgner-Kahrs et al., 2015). El robot utilizado consiste en un endoscopio de cuatro grados de libertad y un robot continuo de cinco grados de libertad para biopsia. Ambos el endoscopio y el robot para biopsia son accionados mediante tendones y vértebras (cilindros) a lo largo de sus longitudes. La habilidad de visualización de ambientes poco accesibles mediante un endoscopio que puede lograr configuraciones curvas, ofrece una gran ventaja a la hora de realizar el proceso quirúrgico, aunque existe un intercambio de menor resolución al utilizar este tipo de endoscopio (Burgner-Kahrs et al., 2015).
- Agarre de objetos mediante un manipulador subactuado, utilizando un brazo robótico continuo que se deforma y envuelve al objeto a ser manipulado, mostrado en (Giri & Walker, 2011) con el uso de su propuesto Octarm. Este tipo de manipulador es muy interesante puesto que, como fue

demostrado en el trabajo presentado por los autores, el manipulador se puede adaptar fácilmente a diferentes objetos con diversas formas y tamaños, agarrándolos suavemente sin estrujarlos. Esto es en contraste a manipuladores robóticos tradicionales que se limitan a la forma y tamaño de su manipulador. El robot presentado en (Giri & Walker, 2011) consiste en una serie de tres secciones, cada una consistiendo en tres actuadores neumáticos posicionados a 120 grados con respecto al otro. La variación en presión en los actuadores neumáticos causa que se genera una curvatura constante en la sección respectiva. Este tipo de robot es muy atractivo para el manejo de objetos delicados y/o suaves, y las aplicaciones para este tipo de robot pueden incluir usos en la industria y manufactura.

- La inspección y/o reparación de motores de turbinas de gas mientras estos sigan montados a las alas de aviones (Dong et al., 2017). Este robot continuo altamente flexible y delgado de 25 grados de libertad puede desenvolverse de un rollo y, mediante un movimiento de alimentación, introducirse al espacio sumamente congestionado y apretado. Luego, con sus últimos seis grados de libertad, puede realizar trayectorias complejas con un manipulador y su respectiva cámara para permitir intervenciones en sitio para el área de baja presión del compresor de una turbina de gas. El método de actuación de este robot es mediante cables y discos, también conocido como tendones y vértebras.
- Robot para la exploración de cuevas (Siles & Walker, 2009). Basado en un robot segmentado, se compone de una serie de módulos flexibles huecos, que se unen en una cadena y que permiten movimiento entre sí. Esto permite que todos los componentes electrónicos se puedan almacenar adentro del robot, mientras que este puede deformar para ajustarse al entorno en el que se encuentra debido a su elevada cantidad de grados de libertad. La locomoción de este robot se logra mediante ruedas movidas por motores o “whegs” que son una especie de rueda que se conforma de

varias patas, para mejorar la tracción y maniobrabilidad en un terreno impredecible de una cueva. La delgadez y habilidad de deformarse permite con mayor facilidad la exploración de ambientes hostiles, peligrosos y difíciles de navegar.

- Robot para manejo versátil de objetos delicados o con formas irregulares (Qadoori Fenjan & Fathollahi Dehkordi, 2024). Este robot utiliza un sistema de vértebras con secciones extensibles y tendones para lograr su accionamiento y poder posicionarse en donde se requiera con precisión. Además, su manipulador en el extremo del robot utiliza un diseño de manipulador suave hecho de silicona, accionado mediante un sistema neumático que permite que este envuelva al objeto y lo pueda transportar. El diseño del robot contempla tenerlo montado verticalmente, teniendo su base arriba y dejando colgar el robot hacia abajo. Esto puede presentarse como una alternativa más accesible en plantas de manufactura puesto que permiten mejor economía de espacio de trabajo.

Si bien existen aplicaciones y casos en los que resulta muy beneficioso utilizar un robot continuo, esto trae consigo problemas en el modelado analítico: la no-linealidad de las deformaciones incurridas en el elemento, la elasticidad del material, susceptibilidad a perturbaciones por el ambiente externo (Wang et al., 2021), al igual que problemas a la hora de evaluar el desempeño del control: menor certeza y precisión (Russo et al., 2023) al igual que peor repetibilidad, en comparación a un actuador rígido.

Estos son nuevos desafíos que no se presentaban en sistemas tradicionales y rígidos. Además, debido a la naturaleza de los sistemas (que no son discretos), no pueden ser modelados por métodos discretos, como se suele hacer para robots tradicionales, debido a su hiper redundancia. Por lo tanto, es necesario recurrir a diferentes métodos de modelado para capturar la compleja mecánica de medios continuos de estos sistemas. Ha habido diferentes enfoques en cuanto al modelado de estos sistemas flexibles y suaves, tal como las teorías de varillas de

Kirchoff (Sun et al., 2019) y Cosserat (J. Zhang et al., 2022), cinemática de cuerpos pseudo rígidos (PRB kinematics), formulación absoluta de coordenada nodal (Absolute nodal coordinate formulation), entre otros (Russo et al., 2023).

La búsqueda de un modelo que pueda capturar la mecánica del sistema flexible debe tener como objetivo la simulación del sistema, de manera que pueda ser utilizado para lograr un esquema de control factible. Sin embargo, muchos de los métodos de simulación para este tipo de aplicación se consideran muy exigentes computacionalmente o sobre simplificados (Naughton et al., 2020), lo que requiere encontrar un equilibrio entre el uso de los recursos computacionales y la precisión de los resultados de las simulaciones. Ha habido diferentes enfoques para la simulación de las estructuras que componen a los robots continuos, tales como el método de elementos finitos con diversas modificaciones o mejoras, herramientas de simulación numérica inspiradas en geometría diferencial basada en estructuras delgadas (Huang et al., 2020), o simulaciones involucrando varillas especiales de Cosserat.

Ante esta problemática, se han realizado esfuerzos para unificar o crear entornos de simulación que aborden la mayoría de estos problemas. Uno de estos entornos de simulación es Elastica (Tekinalp et al., 2024) (el paquete en Python es conocido como PyElastica) (Naughton et al., 2020). Este entorno de simulación busca abordar las limitaciones que hay en los simuladores de cuerpo rígido que no consideran efectos debido a la elasticidad de los materiales, al igual que los costosos, pesados y lentos métodos de elementos finitos de alta fidelidad. Al utilizar la teoría de varillas especiales de Cosserat en sus ecuaciones fundamentales, Elastica logra reducir de manera considerable los costos computacionales y la complejidad del sistema. (Naughton et al., 2020).

La necesidad de tener un entorno de simulación confiable, con facilidad de uso y rapidez es sumamente importante para el desarrollo de sistemas como robots continuos o suaves, puesto que permiten observar y comprobar comportamientos de los materiales y/o actuadores utilizados, de manera numérica. Esto ayuda

mucho en economizar el proceso de desarrollo de prototipos, ayuda a validar modelos planteados, al igual que permite corregir errores en el desarrollo de estos. Esto es mucho más cierto hoy en día, donde el área de robótica continua está creciendo y se están desarrollando nuevas aplicaciones constantemente.

Con diversas aplicaciones en mente, naturalmente surgen diversos tipos de robots continuos. Estos se pueden diferenciar por su método de actuación, estructura o tipo de material utilizado para su estructura. Algunos ejemplos incluyen (Zhong et al., 2020): robots de tubo concéntrico, robots accionados por tendones o cables, sistemas hidráulicos o neumáticos, materiales “inteligentes” (SMA, EAP, DEA, etc.), robots continuos de origami, robots continuos magnéticos y/o accionados por magnetismo, robots con rigidez variable por mecanismo, materiales, viscosidad o efectos acústicos, entre otros.

Debido a su relativamente fácil implementación, este trabajo se va a enfocar en el tipo de robot continuo accionado por tendones. El robot consistirá en una varilla flexible con un sistema de vertebras y tendones, acoplado a lo largo de su longitud. Se utilizará el software libre de simulación mencionado, Elastica, para modelar y simular la dinámica del robot a lo largo del tiempo, sujeto a las fuerzas y momentos generados por el sistema de tendones, debido a que esta clase de fuerzas para robots continuos no se ha implementado para Elastica como paquete de simulación. Adicionalmente, se creará una arquitectura de comunicación mediante ROS2 para facilitar su implementación en físico al dividir las tareas asociadas al simulado y control del robot en nodos que pueden funcionar conjunto, debido a que este tipo de arquitectura de comunicación para este tipo de robots no se ha implementado, o al menos no se ha publicado para el uso en la investigación o innovación.

## **1.2 Objetivo General**

El objetivo principal de este trabajo es desarrollar un entorno de nodos en ROS2 que establezcan comunicación entre sí con el propósito de llevar a cabo la simulación y control de un robot flexible tridimensional impulsado por tendones. Este entorno estará diseñado para facilitar tanto el prototipado físico como el control de este tipo de robots, con el fin de fomentar su uso en la industria. Se empleará el paquete de simulación PyElastica para la simulación.

## **1.3 Objetivos Específicos**

Los objetivos específicos se pueden desglosar en:

1. Entender y desarrollar la teoría detrás del paquete de simulación PyElastica, lo que incluye la teoría de varillas de Cosserat, consideraciones y suposiciones tomadas, al igual que diferentes funciones y/o paquetes utilizados en el código. Esto se hará mediante una búsqueda extensa del estado del arte del modelado de robots continuos, específicamente la teoría de varillas de Cosserat y los artículos y documentación relacionados al paquete de simulación PyElastica.
2. Hacer una revisión de trabajos previos acerca de robots continuos accionados por el uso de tendones. Esto se logrará buscando trabajos previos y levantando el estado del arte acerca de este tema, teniendo como objetivo explorar diferentes enfoques utilizados previamente.
3. Desarrollar un modelo de fuerzas y torques para la actuación de tendones, pensado de tal manera que se pueda implementar directamente en el paquete de simulación PyElastica mediante la codificación en Python de la clase “TendonForces” que puede ser integrada en la simulación a través del módulo de forzado de PyElastica. El desarrollo de este modelo de fuerzas y torques se hará teniendo en mente los trabajos previos revisados.

4. Lograr la validación del modelo propuesto para el accionamiento por tendones, esto se logrará al revisar trabajos previos al respecto y también diseñando y realizando experimentos físicos y corroborando los resultados simulados y medidos.
5. Desarrollar un entorno en ROS2 que combine nodos de simulación, control, visualización, interfaz gráfica de usuario, etc., para llevar a cabo la simulación y el control del robot. Esto se logrará mediante código en Python, utilizando ROS2 para crear diferentes nodos y establecer su comunicación a través de tópicos.
6. Llevar a cabo el control del robot flexible simulado a través del entorno creado en ROS2 para que siga una trayectoria en el espacio tridimensional. Esto involucrará todo el sistema creado: el usuario del sistema podrá especificar una trayectoria y el robot simulado será controlado mediante accionamiento de tendones para cumplir con esta.

## 1.4 Delimitaciones

Las delimitaciones de este trabajo se detallarán a continuación:

1. El elemento para analizar tiene que ser flexible. Esto hace referencia a que sus propiedades materiales o geométricas permitan que sufra grandes desplazamientos, y específicamente en el caso de interés de este trabajo, deflexiones grandes, sin sufrir deformaciones plásticas. Esta consideración se toma por dos razones. Primeramente, la premisa de un robot continuo y flexible requiere que la estructura o elemento que lo conforma sea capaz de deformarse y tener grandes desplazamientos ante las cargas de actuación para lograr tener un espacio de trabajo que sea útil, además que lo pueda lograr sin tener que ejercer fuerzas externas muy elevadas, puesto que esto sería inconveniente para su funcionamiento. Segundo, debido a que el paquete de simulación que será utilizado, Elastica, está diseñado para materiales suaves, no es factible utilizarlo para el caso de materiales rígidos (requiere de un tiempo de computación muy grande). Se recomienda utilizar materiales que tengan un módulo de elasticidad en el rango de megapascales o menos.
2. No se considerará la fricción entre el tendón y vértebra para el modelo. Esta consideración es una simplificación que se tomará para el modelo, debido a que considerar este tipo de fricción trae consigo complicaciones en cuanto al desarrollo del modelo de actuación de tendones debido a que la tensión que experimenta el tendón no será la misma en su longitud (que bien podría ser un proyecto a desarrollar a futuro). Además, se considera que, si se utilizan materiales para el tendón y las vértebras que tengan poca fricción, se puede hacer la suposición de omitirla (Rucker & Webster III, 2011). También, el efecto que tiene la fricción en el sistema se puede considerar negligible generalmente debido a que su impacto en la precisión

del modelado es de ordenes de magnitud menores a la actuación (Rao et al., 2021).

3. Las ecuaciones de varilla de Cosserat serán manejadas por PyElastica. Esto se debe a que el paquete de simulación es confiable y validado en varias diferentes ocasiones (Naughton et al., 2020; X. Zhang et al., 2019)
4. No se construirá un prototipo en físico del sistema controlable. Se considera que esto está fuera del alcance del trabajo, es recomendado como un trabajo a futuro.
5. La curvatura de la varilla tendrá como máximo 1 cambio de concavidad (descrito sobre el plano promedio formado por la punta, la base y el promedio de los puntos a lo largo de la curva, lo que permite definir este plano en diferentes orientaciones en el espacio de trabajo). Esta es una consideración que afecta el alcance del control del elemento flexible. Considerar múltiples cambios de concavidad trae consigo muchas complicaciones que no necesariamente son importantes considerar para el control de la punta del robot en su espacio de trabajo. Además, se busca que la configuración que tenga el elemento flexible no sea innecesariamente complicada, por lo que limitar la complejidad de su curvatura es importante.

## 1.5 Estructura del trabajo

Este trabajo se presentará de la siguiente manera: el capítulo II abarcará el contenido respectivo al modelado de la varilla flexible que se utilizará como modelo para el robot continuo que será analizado. Este capítulo se adentra en el estado del arte, al igual que desarrolla la teoría de varillas de Cosserat, hasta llegar a las ecuaciones dinámicas que gobiernan el sistema que será simulado utilizando Elastica. Siguiendo, el capítulo III abarcará el accionamiento del robot continuo mediante el uso de tendones. Esto implica explorar el estado del arte, lograr un planteamiento numérico para que este pueda ser implementado en el software de simulación, al igual que los experimentos físicos y validación necesaria para respaldar el planteamiento numérico de fuerzas y momentos externos causados por los tendones. Después, el capítulo IV se concentrará en el desarrollo de la red de comunicación en ROS2, que comprende los diferentes nodos, sus funciones, sus tópicos y las conexiones que tienen entre sí. Además, se adentrará en los diferentes métodos de integración que fueron utilizados para lograr la correcta interconexión de los nodos y módulos para que funcione el entorno de simulación. Posterior a este desarrollo, se analizarán los resultados obtenidos mediante el presente entorno de simulación. Finalmente, se llevarán a cabo las conclusiones del trabajo y recomendaciones para trabajos a futuro, seguido por referencias bibliográficas y anexos.

## 2 Modelado de la varilla flexible

### 2.1 Estado del arte

El modelado de vigas o estructuras delgadas ha sido un tema de interés en el mundo de la ingeniería debido a su inmensa cantidad de aplicaciones y potencial para resolver muchos problemas en la ingeniería. Es común ver la utilización de la teoría clásica de vigas, también conocida como la teoría Euler-Bernoulli de vigas, para modelar elementos delgados sometidos a cargas transversales. Esta teoría resulta ser válida cuando se consideran deformaciones pequeñas (la variación espacial de la deflexión de la viga se supone que es pequeña, es decir, los ángulos de deformación formados por la línea centroidal de la viga son menores a  $5^\circ$ ), puesto que se pueden hacer simplificaciones en base a esta suposición. Sin embargo, a la hora de describir el comportamiento que puede tener el elemento tipo viga o varilla ante cargas aplicadas tal magnitud que producen grandes deflexiones, se pierde la linealidad de la relación carga-deflexión (Reddy, 2004). En el caso de robots continuos, especialmente aquellos accionados por tendones, es que tengan una “columna vertebral” flexible. Esto quiere decir, que el elemento tipo varilla que conforma su estructura permita incurrir en deflexiones grandes y tener la capacidad de regresar a su configuración original sin sufrir deformaciones plásticas (deflexiones siendo el desplazamiento de los puntos de la línea centroidal desde la configuración inicial a la final, y deformación siendo la taza de cambio del desplazamiento de los puntos del material, que en este caso debería ser igual a cero cuando se deja reposar el elemento tipo viga o varilla para tener una deformación elástica). Por lo tanto, utilizar una teoría de modelado tal como la de Euler-Bernoulli no resulta factible, puesto que está limitada en su planteamiento. Existen diferentes enfoques que se han utilizado para intentar modelar este tipo de elementos flexibles, tales como: teoría de varillas de Cosserat, cinemática de cuerpos pseudo rígidos, formulación de coordenadas nodales absolutas, entre otras (Russo et al., 2023). Es importante

que la formulación escogida para lograr las simulaciones numéricas sea capaz de hacerlo de manera precisa y certera. Sin embargo, muchos de los métodos de simulación actuales para este tipo de aplicación tienden a ser o muy pesados computacionalmente o sobre simplificados para mejorar la rapidez (Naughton et al., 2020). Esto significa que debe haber un balance entre peso computacional y precisión, lo cual llevó a cabo la creación del paquete de simulación Elastica (Naughton et al., 2020), que busca lograr este equilibrio. Las ecuaciones fundamentales que utiliza este software de simulación de código abierto son las ecuaciones de la teoría de varillas de Cosserat. A estas le hacen arreglos para mejorar su precisión y facilitar su adaptación a la simulación numérica.

## **2.2 Teoría de varillas de Cosserat**

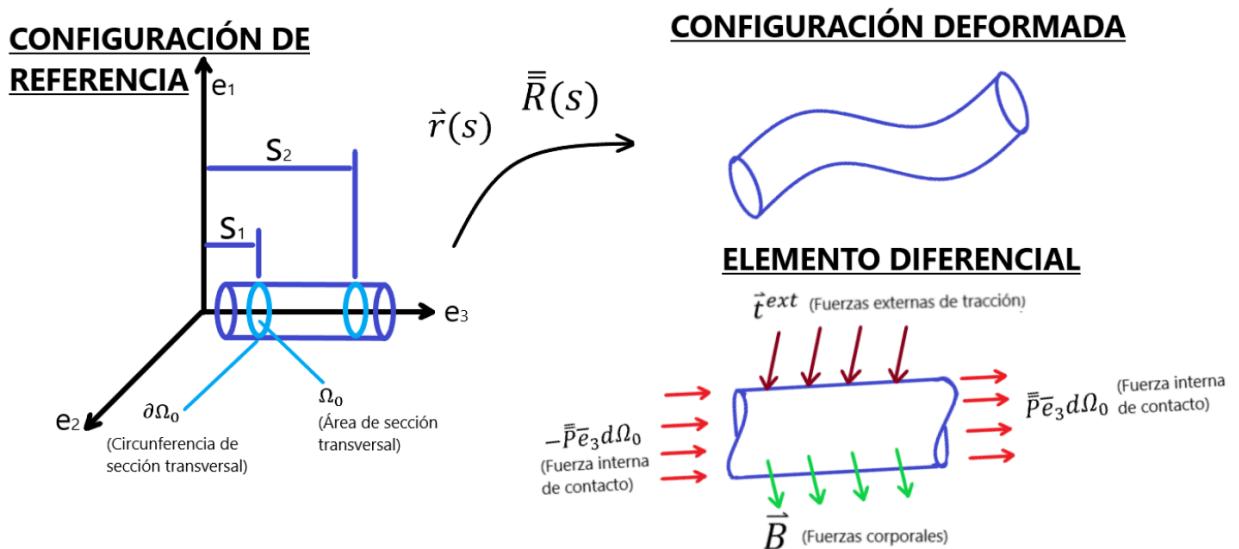
En esencia, la teoría de varillas de Cosserat (Cosserat & Cosserat, 1909) busca modelar la varilla (elemento delgado) como una curva de material que tiene una serie de vectores deformables asociados con cada punto en esa curva (O'Reilly, 2017). La teoría de varillas de Cosserat es conocida como la teoría de vigas geométricamente exacta (Kumar, 2010), debido a que considera todos los modos de deformación posibles en el elemento: flexión, torsión, cortante y estiramiento (Naughton et al., 2020).

Para modelar este tipo de varilla, se toma la suposición que se está trabajando con un elemento delgado, es decir, que su radio es mucho menor a su longitud (en el orden de diez veces o más). Debido a esta delgadez del elemento, se puede hacer la suposición de que, si se conoce la posición de su línea centroidal a lo largo de su longitud, al igual que la orientación que tienen su cara transversal (que vendría siendo la orientación que tiene cada uno de los sistemas de coordenadas asociados a cada punto en la longitud), entonces se puede “reconstruir” la varilla en cualquiera configuración que tenga.

Por lo tanto, se pueden definir dos variables cinemáticas desconocidas,  $\vec{r}(s)$  que vendría siendo la posición de la línea centroidal con respecto al sistema de

coordenadas de referencia y  $\bar{R}(s)$  que vendría siendo la rotación asociada a la sección transversal en la longitud de arco de la configuración inicial,  $s$ . Esto significa que cualquier punto ubicado en  $s$  en el arco original, es decir, no deformado, puede ser mapeado, mediante las funciones  $\bar{R}(s)$  y  $\bar{r}(s)$  a su nueva configuración en la varilla deformada.

Para desarrollar las ecuaciones de la teoría de varillas de Cosserat, primero se comienza con el equilibrio de las fuerzas en un elemento diferencial de varilla:



**Figura 2-1.** El análisis de equilibrio de fuerzas requiere considerar un elemento diferencial aleatorio en la longitud de arco de la varilla.

Primero, se analizan las fuerzas internas externas que recibe el elemento diferencial:

$$\iiint \vec{B} dV + \iint \vec{t}^{ext} dA$$

Donde  $\vec{B}$  son las fuerzas corporales que actúan en la varilla a lo largo de su volumen, y  $\vec{t}^{ext}$  son las fuerzas externas que son aplicadas a la varilla, mediante

una interacción con su superficie exterior. A la ecuación (1), se pueden agrupar los términos al factorizar por la longitud de arco, como se muestra a continuación:

$$\int_{s_1}^{s_2} ds \iint_{\Omega_0} \vec{B} dA + \int_{s_1}^{s_2} ds \int_{\partial\Omega_0} \vec{t}^{ext} dl$$

$$\int_{s_1}^{s_2} ds \left( \iint_{\Omega_0} \vec{B} dA + \int_{\partial\Omega_0} \vec{t}^{ext} dl \right)$$

Donde  $\Omega_0$  hace referencia al área de la sección transversal y  $\partial\Omega_0$  hace referencia a la circunferencia de la sección transversal. Lo que lleva a obtener:

$$\int_{s_1}^{s_2} \hat{n}(s) ds = \int_{s_1}^{s_2} \left( \iint_{\Omega_0} \vec{B} dA + \int_{\partial\Omega_0} \vec{t}^{ext} dl \right) ds \quad (1)$$

Donde  $\hat{n}(s)$  hace referencia a una carga distribuida externa, aplicada a la varilla, que tiene una dirección arbitraria. Se debe notar que las fuerzas corporales se incluyen en este término de carga distribuida.

Ahora, analizando las fuerzas internas que ejerce el elemento tipo varilla:

$$\iint_{\Omega_0} \bar{\bar{P}} \bar{e}_3 d\Omega_0$$

$$\iint_{\Omega_0} \bar{\bar{P}}(X_1, X_2, s) \bar{e}_3 d\Omega_0$$

Donde  $\bar{\bar{P}}$  es el tensor de esfuerzo Piola-Kirchoff, que detalla el estado de esfuerzo a lo largo de la varilla, descrito en términos del arco original y no deformado. Para el análisis que se está haciendo en este desarrollo, es importante saber que esta fuerza interna resultante del esfuerzo interno se debe evaluar en ambas secciones transversales del elemento diferencial. En ese caso, una de las fuerzas resultantes

será negativa, puesto que está apuntando en la dirección  $-e_3$ , como se puede ver en la **Figura 2-1**. Este término puede ser escrito de la siguiente manera, para poder combinarlo con el término de fuerzas externas:

$$\iint_{\Omega_0} \bar{P}(X_1, X_2, s) \bar{e}_3 dA = \vec{n}(s) \quad (2)$$

Entonces,  $\vec{n}(s) = [\vec{n}_1(s), \vec{n}_2(s), \vec{n}_3(s)]^T$  hace referencia a las fuerzas internas de cortante ( $\vec{n}_1(s), \vec{n}_2(s)$ ) y axiales ( $\vec{n}_3$ ) que se presentan en el material del elemento, debido a los esfuerzos internos generados. La suma de las fuerzas resultantes internas en el elemento diferencial sería:

$$\vec{n}(s_2) - \vec{n}(s_1)$$

Entonces, combinando las fuerzas internas (2) y externas (1), se tiene que:

$$\int_{s_1}^{s_2} \hat{n}(s) ds + \vec{n}(s_2) - \vec{n}(s_1) = 0$$

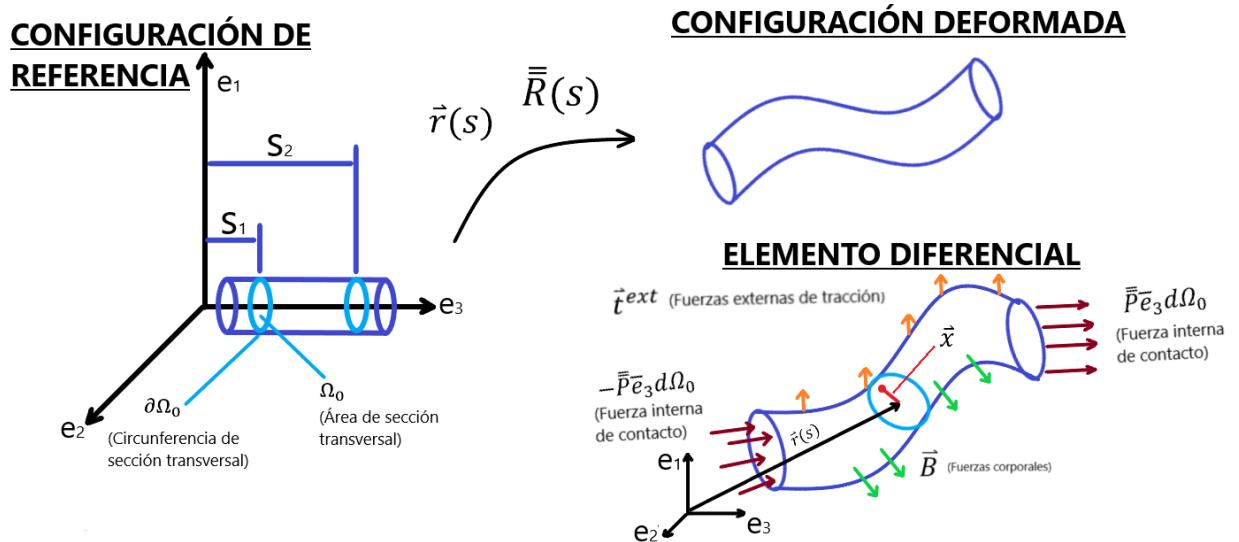
Se destaca que las fuerzas internas en el elemento diferencial se evalúan en ambas secciones transversales, por lo que aparecen los términos de  $\vec{n}(s)$  evaluados en  $s_2$  y  $s_1$ . Ahora, se puede hacer la observación que los límites de integración del término de fuerzas externas son los mismos en los cuales son evaluadas las fuerzas resultantes en las secciones transversales del elemento diferencial. Esto significa que las fuerzas resultantes internas pueden ser agrupadas en la misma integral que el término de fuerzas externas, teniendo solamente que derivar el término de  $\vec{n}(s)$  y evaluarlo en los mismos límites de integración, como se puede ver a continuación:

$$\int_{s_1}^{s_2} (\hat{n}(s) + \vec{n}'(s)) ds = 0$$

Debido a que los límites de integración,  $s_2$  y  $s_1$ , son arbitrarios, significa que el integrando debe ser igual a cero en este caso, por lo que se tiene que la primera ecuación de la teoría de varillas de Cosserat es la siguiente (Kumar et al., 2016):

$$\hat{n}(s) + \vec{n}'(s) = 0 \quad (3)$$

Siguiendo con el análisis, se debe hacer un balance de momentos para obtener la segunda ecuación fundamental de la teoría de varillas de Cosserat. El balance de momentos se hará con respecto al origen del sistema de coordenadas. Entonces, se tiene lo siguiente:



**Figura 2-2.** El análisis de balance de momentos requiere considerar un elemento diferencial aleatorio en la longitud de arco de la varilla.

Se comienza por definir un vector posición arbitrario  $\vec{x}$  que ubica un punto en una sección transversal arbitraria en el elemento deformado:

$$\vec{x} = \vec{r}(s) + (\vec{x} - \vec{r}(s)) \quad (4)$$

Entonces, se comienza con el planteamiento del momento generado debido a cargas externas:

$$\iiint_{V_0} (\vec{x} \times \vec{B}) dV_0 + \iint_{\Omega_0} (\vec{x} \times \vec{t}^{ext}) d\Omega_0 \quad (5)$$

Donde  $V_0$  hace referencia al volumen deformado. Ahora, introduciendo (4) en (5):

$$\begin{aligned} \int_{s_1}^{s_2} \vec{r}(s) ds \times \iint_{\Omega_0} \vec{B} d\Omega_0 + \int_{s_1}^{s_2} ds \iint_{\Omega_0} ((\vec{x} - \vec{r}(s)) \times \vec{B}) d\Omega_0 + \int_{s_1}^{s_2} \vec{r}(s) ds \times \int_{\partial\Omega_0} \vec{t}^{ext} dl \\ + \int_{s_1}^{s_2} ds \int_{\partial\Omega_0} ((\vec{x} - \vec{r}(s)) \times \vec{t}^{ext}) dl \end{aligned}$$

Agrupando los términos:

$$\begin{aligned} \int_{s_1}^{s_2} \vec{r}(s) ds \times \left[ \iint_{\Omega_0} \vec{B} d\Omega_0 + \int_{\partial\Omega_0} \vec{t}^{ext} dl \right] \\ + \int_{s_1}^{s_2} ds \left[ \iint_{\Omega_0} ((\vec{x} - \vec{r}(s)) \times \vec{B}) d\Omega_0 + \int_{\partial\Omega_0} ((\vec{x} - \vec{r}(s)) \times \vec{t}^{ext}) dl \right] \end{aligned}$$

Aquí se pueden identificar algunos términos, de manera que se pueda esclarecer la ecuación. Primero, se identifica que:

$$\iint_{\Omega_0} \vec{B} d\Omega_0 + \int_{\partial\Omega_0} \vec{t}^{ext} dl = \hat{n}(s)$$

Esto se desarrolló en la sección de equilibrio de fuerzas, y se puede aplicar para esta sección también. Ahora, la segunda parte se identifica como:

$$\iint_{\Omega_0} ((\vec{x} - \vec{r}(s)) \times \vec{B}) d\Omega_0 + \int_{\partial\Omega_0} ((\vec{x} - \vec{r}(s)) \times \vec{t}^{ext}) dl = \hat{m}(s)$$

Donde  $\hat{m}(s)$  hace referencia al momento externo distribuido a lo largo de varilla debido a las fuerzas corporales y tracción lateral con respecto a la línea centroidal de la varilla.

Entonces, para el momento debido a cargas externas, se tiene:

$$\int_{s_1}^{s_2} (\vec{r}(s) \times \hat{n}(s)) ds + \int_{s_1}^{s_2} \hat{m}(s) ds \quad (6)$$

Ahora, analizando el momento debido a cargas internas, se utiliza el tensor de esfuerzo Piola-Kirchoff, similar al balance de fuerzas:

$$\iint_{\Omega_{0s2}} (\vec{x} \times \bar{\bar{P}}\vec{e}_3) d\Omega_0 - \iint_{\Omega_{0s1}} (\vec{x} \times \bar{\bar{P}}\vec{e}_3) d\Omega_0$$

Desarrollando esta expresión:

$$\begin{aligned} & \iint_{\Omega_{0s2}} (\vec{r}(s_2) \times \bar{\bar{P}}\vec{e}_3) d\Omega_0 + \iint_{\Omega_{0s2}} ((\vec{x} - \vec{r}(s_2)) \times \bar{\bar{P}}\vec{e}_3) d\Omega_0 \\ & - \iint_{\Omega_{0s1}} (\vec{r}(s_1) \times \bar{\bar{P}}\vec{e}_3) d\Omega_0 - \iint_{\Omega_{0s1}} ((\vec{x} - \vec{r}(s_1)) \times \bar{\bar{P}}\vec{e}_3) d\Omega_0 \end{aligned}$$

Ahora, sacando los términos de  $\vec{r}(s_2)$  y  $\vec{r}(s_1)$  ya que estos son valores constantes:

$$\begin{aligned} & \vec{r}(s_2) \times \iint_{\Omega_{0s2}} \bar{\bar{P}}\vec{e}_3 d\Omega_0 + \iint_{\Omega_{0s2}} ((\vec{x} - \vec{r}(s_2)) \times \bar{\bar{P}}\vec{e}_3) d\Omega_0 - \vec{r}(s_1) \times \iint_{\Omega_{0s1}} \bar{\bar{P}}\vec{e}_3 d\Omega_0 \\ & - \iint_{\Omega_{0s1}} ((\vec{x} - \vec{r}(s_1)) \times \bar{\bar{P}}\vec{e}_3) d\Omega_0 \end{aligned}$$

Del análisis previo hecho para el equilibrio de fuerzas, se conoce que:

$$\iint_{\Omega_0} \bar{\bar{P}}\vec{e}_3 d\Omega_0 = \vec{n}(s)$$

Lo que lleva a tener:

$$\begin{aligned} & \vec{r}(s_2) \times \vec{n}(s_2) + \iint_{\Omega_{0s2}} ((\vec{x} - \vec{r}(s_2)) \times \bar{\bar{P}}\vec{e}_3) d\Omega_0 - \vec{r}(s_1) \times \vec{n}(s_1) \\ & - \iint_{\Omega_{0s1}} ((\vec{x} - \vec{r}(s_1)) \times \bar{\bar{P}}\vec{e}_3) d\Omega_0 \end{aligned}$$

Ahora, se pueden identificar los términos que faltan por simplificar, los cuales tienen la misma forma, pero definidos uno en  $s_1$  y el otro en  $s_2$ . Para ambos términos, se tiene lo siguiente:

$$\iint_{\Omega_{0s2}} \left( (\vec{x} - \vec{r}(s_2)) \times \bar{\vec{P}}\bar{e}_3 \right) d\Omega_0 = \vec{m}(s_2)$$

$$\iint_{\Omega_{0s1}} \left( (\vec{x} - \vec{r}(s_1)) \times \bar{\vec{P}}\bar{e}_3 \right) d\Omega_0 = \vec{m}(s_1)$$

Donde  $\vec{m}(s) = [\vec{m}_1(s), \vec{m}_2(s), \vec{m}_3(s)]^T$  hace referencia a los momentos flectores ( $\vec{m}_1(s), \vec{m}_2(s)$ ) y torsional ( $\vec{m}_3(s)$ ) internos que son un resultado de los esfuerzos internos que sufre el material de la varilla ante las cargas.

Lo que lleva a tener, para el momento debido a cargas internas en el material del elemento:

$$\vec{r}(s_2) \times \vec{n}(s_2) + \vec{m}(s_2) - \vec{r}(s_1) \times \vec{n}(s_1) - \vec{m}(s_1) \quad (7)$$

Entonces, planteando el balance de momentos al sumar el momento externo (6) e interno (7):

$$\int_{s_1}^{s_2} (\vec{r}(s) \times \hat{n}(s) + \hat{m}(s)) ds + \vec{r}(s_2) \times \vec{n}(s_2) + \vec{m}(s_2) - \vec{r}(s_1) \times \vec{n}(s_1) - \vec{m}(s_1) = 0$$

Ahora, agrupando todos los términos para que estén dentro de la misma integral, se tiene lo siguiente:

$$\int_{s_1}^{s_2} \left( \vec{r}(s) \times \hat{n}(s) + \hat{m}(s) + (\vec{r}(s) \times \vec{n}(s))' + \vec{m}'(s) \right) ds = 0$$

Este arreglo es posible porque los términos que antes estaban evaluados en  $s_1$  y en  $s_2$ , pueden ser vistos como la integral de su derivada evaluada en esos mismos límites.

Realizando la derivación implícita del siguiente término:

$$(\vec{r}(s) \times \vec{n}(s))' = \vec{r}'(s) \times \vec{n}(s) + \vec{r}(s) \times \vec{n}'(s)$$

Lo que significa que la ecuación puede agruparse de la siguiente manera:

$$\int_{s_1}^{s_2} (\vec{r}(s) \times (\vec{n}'(s) + \hat{n}(s)) + \hat{m}(s) + \vec{r}'(s) \times \vec{n}(s)' + \vec{m}'(s)) ds = 0$$

Se había obtenido previamente la primera ecuación de la teoría de varillas de Cosserat, la cual se ve que se presenta en la anterior ecuación. Entonces, aplicando la ecuación (3), se tiene:

$$\int_{s_1}^{s_2} (\hat{m}(s) + \vec{r}'(s) \times \vec{n}(s)' + \vec{m}'(s)) ds = 0$$

Debido a que  $s_1$  y  $s_2$  son límites arbitrarios, significa que el integrando tiene que ser igual a cero, por lo que se tiene que la segunda ecuación de la teoría de varillas de Cosserat es (Kumar et al., 2016):

$$\hat{m}(s) + \vec{r}'(s) \times \vec{n}(s)' + \vec{m}'(s) = 0 \quad (8)$$

Teniendo las ecuaciones de la teoría de varillas de Cosserat, (3) y (8), se identifica que se tiene un sistema de seis ecuaciones, donde se conocen las cargas externas,  $\hat{n}(s)$  y  $\hat{m}(s)$ . Sin embargo, como se había planteado inicialmente, el propósito de utilizar la teoría de varillas de Cosserat era aprovechar el uso de un elemento delgado, y por ende poder reconstruir su configuración deformada utilizando las variables cinemáticas: la posición de su línea centroidal,  $\vec{r}(s)$ , y la orientación de las caras transversales y las variables de fuerzas internas,  $\bar{\vec{R}}(s)$ . Las ecuaciones (3) y (8) no están en términos de estas variables cinemáticas, sino que describen el balance de fuerzas y momentos. Esto significa que debe haber una manera para unir las ecuaciones y expresarlas en términos de las variables cinemáticas, lo cual se puede lograr mediante leyes constitutivas.

Para desarrollar estas leyes constitutivas, se puede partir desde la función de energía para una varilla, la cual debe estar expresada en términos de las invariantes, igual a como se ve en la teoría de medios continuos.

Se puede proponer una función de energía  $\Psi$  que tenga la siguiente forma:

$$\vec{v} = \bar{\bar{R}}^T \vec{r}'$$

$$\bar{\bar{k}} = \bar{\bar{R}} \bar{\bar{R}}'$$

$$\Psi(\vec{v}, \bar{\bar{k}})$$

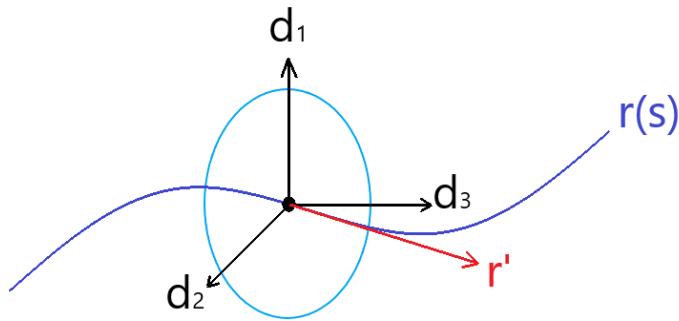
Donde:

$$\vec{r}' = \bar{\bar{R}} \vec{v} = \bar{\bar{R}} v_i e_i = v_i d_i$$

$$v_i = \vec{r}' d_i$$

$$\bar{\bar{k}} = \begin{bmatrix} 0 & -k_3 & k_2 \\ k_3 & 0 & -k_1 \\ -k_2 & k_1 & 0 \end{bmatrix}$$

Aquí, se pueden identificar variables:  $\vec{v} = [v_1 \ v_2 \ v_3] = [\vec{r}' d_1 \ \vec{r}' d_2 \ \vec{r}' d_3]$ , donde  $v_1$  y  $v_2$  hacen referencia a las deformaciones de cortante, y  $v_3$  hace referencia a la deformación de estiramiento axial en la varilla. Por otro lado, la matriz  $\bar{\bar{k}}$  es una matriz antisimétrica que contiene los valores  $\vec{k} = \text{axial}(\bar{\bar{k}}) = [k_1 \ k_2 \ k_3]$ , donde  $k_1$  y  $k_2$  hacen referencia a la curvatura debido a la flexión que tiene la varilla y  $k_3$  hace referencia a la deformación debido a la torsión en la varilla.



**Figura 2-3.** Se muestra el arreglo de los directores locales y también el vector  $\vec{r}'$  que contempla la derivada con respecto a la longitud de arco  $s$  de la línea centroidal de la varilla.

De esta manera, la función de energía de deformación de la varilla está descrita en términos de las invariantes, que vendrían siendo las deformaciones que sufre la varilla. Sin embargo, todavía no se conoce la forma que tiene esta función de energía  $\Psi(\vec{v}, \bar{k})$ .

Si se hace la suposición de que la varilla es isotrópica y de geometría uniforme y el material de la varilla es perfectamente elástico y se caracteriza con una relación de esfuerzo-deformación lineal, entonces la función de energía de deformación de la varilla toma la siguiente forma (Chang et al., 2020):

$$\begin{aligned} \Psi(\vec{v}, \bar{k}) = & \frac{1}{2} (k_T G A (v_1 - v_1^0)^2 + k_T G A (v_2 - v_2^0)^2 + E A (v_3 - v_3^0)^2 + E I (k_1 - k_1^0)^2 \\ & + E I (k_2 - k_2^0)^2 + G J (k_3 - k_3^0)^2) \end{aligned} \quad (9)$$

Donde: el superíndice 0 se refiere al estado original,  $k_T$  es la constante de corrección de cortante del modelo de vigas de Timoshenko,  $G$  es el módulo de rigidez al cortante,  $A$  es el área de la sección transversal,  $E$  es el módulo de elasticidad,  $I$  es el segundo momento de área de la sección transversal,  $J$  es el

momento polar de la sección transversal de la varilla y el superíndice 0 se refiere a la configuración inicial o previa.

Se puede demostrar que, utilizando el principio de minimización de energía potencial, que, para una varilla hiperelástica (Antman, 1995), las ecuaciones constitutivas que gobiernan su comportamiento son las siguientes:

$$\vec{n} = \bar{\bar{R}} \frac{\partial \Psi}{\partial \vec{v}} \quad (10)$$

$$\vec{m} = \bar{\bar{R}} \frac{\partial \Psi}{\partial \vec{k}} \quad (11)$$

Donde, se tiene que:

$$N_1 = \frac{\partial \Psi}{\partial \vec{v}_1}$$

$$N_2 = \frac{\partial \Psi}{\partial \vec{v}_2}$$

$$N_3 = \frac{\partial \Psi}{\partial \vec{v}_3}$$

$$M_1 = \frac{\partial \Psi}{\partial \vec{k}_1}$$

$$M_2 = \frac{\partial \Psi}{\partial \vec{k}_2}$$

$$M_3 = \frac{\partial \Psi}{\partial \vec{k}_3}$$

Y  $N_1, N_2, N_3$  son fuerzas cortantes y la fuerza axial, respectivamente. Al igual que  $M_1, M_2, M_3$  son los momentos flectores y el momento torsional, respectivamente.

Teniendo estas relaciones constitutivas (10) y (11), junto con la función de energía de deformación (9), se puede retornar a las ecuaciones de la teoría de varillas de

Cosserat (3) y (8), puesto que ahora se pueden escribir en términos de las variables cinemáticas de interés:

$$\left( \bar{\bar{R}} \frac{\partial \Psi}{\partial \vec{v}} \right)' + \hat{n} = 0 \quad (12)$$

$$\left( \bar{\bar{R}} \frac{\partial \Psi}{\partial \vec{k}} \right)' + \vec{r}' \times \bar{\bar{R}} \frac{\partial \Psi}{\partial \vec{v}} + \hat{m} = 0 \quad (13)$$

Para obtener las ecuaciones de movimiento que gobiernan el sistema, se hace una consideración similar a aquella hecha para las ecuaciones de equilibrio, solamente que el lado derecho de las ecuaciones no será igual a cero, sino que sería la derivada temporal del momentum lineal y angular, respectivamente. Después de desarrollar aquellas ecuaciones de movimiento, se tendrían las siguientes ecuaciones de movimiento que gobiernan la dinámica del sistema (Kumar, 2010):

$$\left( \bar{\bar{R}} \frac{\partial \Psi}{\partial \vec{v}} \right)' + \hat{n} = \rho A \ddot{r} \quad (14)$$

$$\left( \bar{\bar{R}} \frac{\partial \Psi}{\partial \vec{k}} \right)' + \vec{r}' \times \bar{\bar{R}} \frac{\partial \Psi}{\partial \vec{v}} + \hat{m} = \rho [\bar{\bar{I}} \dot{\vec{\omega}}] \quad (15)$$

Donde  $\rho$  es la densidad por unidad de volumen,  $A$  es el área de la sección transversal en la longitud de arco  $s$ ,  $\bar{\bar{I}}$  es el tensor de segundo momento de área de la sección transversal y  $\vec{\omega}$  es la velocidad angular de la sección transversal.

## **2.3 Desarrollo del paquete de simulación Elastica**

Esta sección resaltará los arreglos y consideraciones que se tomaron en el trabajo de (Gazzola et al., 2018) para formar el paquete de simulación numérica Elastica

(Tekinalp et al., 2024), que utiliza las ecuaciones de la teoría de varillas de Cosserat para sus ecuaciones fundamentales.

Lo primero que buscan es transformar el sistema directamente a coordenadas lagrangianas (locales), desde las coordenadas globales. Debido a la ortonormalidad de los directores de los marcos de referencia locales, las derivadas temporales y espaciales de los marcos de referencia locales,  $\bar{R}(s, t)$ , están asociadas a la velocidad angular  $\vec{\omega}$  y la curvatura  $\vec{k}$ . Se tiene que:

$$\frac{\partial d_j}{\partial t} = \frac{\partial(\bar{R}^T e_j)}{\partial t} = \frac{\partial \bar{R}^T}{\partial t} e_j = \frac{\partial \bar{R}^T}{\partial t} \bar{R} d_j = \vec{\omega} \times d_j \quad (16)$$

$$\frac{\partial d_j}{\partial s} = \frac{\partial(\bar{R} e_j)}{\partial s} = \frac{\partial \bar{R}^T}{\partial s} e_j = \frac{\partial \bar{R}^T}{\partial s} \bar{R} d_j = \vec{k} \times d_j \quad (17)$$

Además, se tiene que la derivada en el tiempo y en el espacio para la línea centroidal de la varilla  $\vec{r}(s, t)$  son las siguientes:

$$\dot{\vec{r}} = \frac{\partial \vec{r}}{\partial t} \quad (18)$$

$$\vec{r}' = \frac{\partial \vec{r}}{\partial s} \quad (19)$$

Lo que lleva a que se combinen las ecuaciones (16) y (18) se combinen con las ecuaciones (14) y (15), para obtener las ecuaciones de gobierno para la varilla de Cosserat (Gazzola et al., 2018).

Es de conveniencia tener todas las ecuaciones del sistema con respecto al marco de referencia local, puesto que la resultante de fuerza y momento interno dependen de propiedades geométricas y materiales, lo que se puede ver en las leyes constitutivas. Esto hace que sea más natural y conveniente que estén

expresadas en términos de los marcos de referencia locales puesto que necesitan ser invariantes, y en este caso serían constantes si se analizan localmente.

Después de realizar arreglos y transformaciones, se obtiene el sistema expresado con variables definidas localmente.

Ahora, continuando con el desarrollo, se añade un factor de dilatación a las ecuaciones de gobierno del sistema, puesto que se quiere capturar los efectos de deformaciones de cortante y axiales en cada sección transversal de la varilla. Esto significa que ahora  $\frac{\partial \vec{r}}{\partial s} \neq d_3$ . Es decir, la derivada del vector posición  $\vec{r}(s, t)$  con respecto a la longitud de arco inicial  $s$ , no necesariamente es igual al tercer vector director en el marco de referencia local en las secciones transversales en la varilla. Esto se puede apreciar mejor en la **Figura 2-3**.

Entonces, se introduce un factor de dilatación local  $e(\hat{s}, t) = \frac{ds}{d\hat{s}}$ , donde  $\hat{s}$  es la configuración de referencia original, que afecta las propiedades geométricas del sistema, las cuales ahora dependen de este factor de dilatación. Estas propiedades geométricas se aplican al sistema de ecuaciones.

Una vez tienen el nuevo sistema con el factor de dilatación local incorporado, se necesita encontrar una manera de resolver el sistema numéricamente para poder lograr las simulaciones. Lo primero que se hace es escoger una manera de representar las rotaciones en la simulación. El método escogido es el de la matriz exponencial  $R = e^{\theta u}: \mathbb{R}^3 \rightarrow \mathbb{R}^{3 \times 3}$ . Este método permite la computación eficiente de transformaciones geométricas no lineales, de manera rápida, ya que puede ser fácilmente calculada mediante la fórmula de Rodrigues (Gazzola et al., 2018; Rodrigues, n.d.).

Para la simulación de las ecuaciones de gobierno, se busca discretizar la varilla en una serie discreta de vértices y marcos de referencia locales. Cada vértice tiene asociado a sí los siguientes valores: vector de posición, velocidad del vector de posición, masa puntual concentrada, y fuerza externa. Luego, cada marco de

referencia local tiene asociado a sí los siguientes valores: longitud entre vértices, longitud de referencia entre vértices, factor de dilatación local, vector tangente a la línea centroidal de la varilla, vector de deformación cortante/axial, vector de velocidad angular, área de sección transversal de referencia, segundo momento de inercia de masa, matriz de flexión/torsión, matriz de cortante/estiramiento y momento externo.

En un sistema continuo, todas las variables se pueden definir de manera puntual y discreta, sin embargo, en el caso de un sistema discreto, algunas variables, especialmente aquellas como la curvatura de la varilla, deben ser descritas en una forma integrada a lo largo de un dominio (Gazzola et al., 2018). Entonces, lo que hicieron fue designar el dominio como una región Voronoi, puesto que cualquiera variable definida en el dominio, dividida por el correspondiente dominio de integración, da el promedio puntual.

Entonces, los dominios Voronoi se definen para todos los vértices interiores (aquellas que no están en los extremos de la varilla). A los vértices interiores también se le asocian los siguientes valores: longitud de dominio Voronoi original y factor de dilatación de la región Voronoi. También, la matriz de rigidez de flexión/torsión se modifica para que sea consistente con la representación Voronoi, al igual que las curvaturas de la varilla.

Al introducir estos cambios al sistema, se tiene el sistema final, discretizado en el espacio. Lo que queda por hacer es utilizar algún tipo de método para resolver estas ecuaciones, o integrarlas en el tiempo para observar su evolución.

Para el paquete de simulación Elastica, se escogió optar por una discretización temporal, de manera que se evolucionen las ecuaciones de gobierno a través del tiempo. Los autores (Gazzola et al., 2018) escogen utilizar un integrador temporal que pueda conservar energía, por lo tanto, se escogió utilizar un esquema de segundo orden simpléctico de Verlet.

El funcionamiento de este integrador es el siguiente: primero se actualizan las posiciones lineales y angulares, luego se evalúan las aceleraciones lineales y angulares locales, y luego se vuelven a actualizar las posiciones lineales y angulares nuevamente.

Los esquemas de integración simplécticos son muy importantes para sistemas que consideran conservar energía a lo largo de un gran tiempo de simulación (Dixon & Reich, 2004). Este integrador escogido logra un balance entre costos computacionales, precisión numérica e implementación modular. El uso de un integrador implícito permite incorporar varios efectos físicos y restricciones suaves adicionales, aunque pueden elevar el costo computacional.

Es importante destacar que el sistema puede volverse “stiff” o “rígido” cuando se tienen módulos de elasticidad muy grandes, o varillas demasiado delgadas. Los autores recomiendan seguir una relación empírica para prevenir inestabilidades numéricas al escoger la discretización temporal adecuadamente (Gazzola et al., 2018):

$$dt = a dl$$

Donde  $dt$  hace referencia al “step” temporal que se le indica al simulador para que discretice el tiempo total en intervalos de esa magnitud,  $a = \sim 10^{-2} \frac{s}{m}$  y  $dl$  es la longitud de cada intervalo de discretización espacial.

Para la validación del paquete de simulación, se realizaron varios diferentes ejemplos con soluciones conocidas o experimentales. En (Gazzola et al., 2018), los autores validan el software con tres casos conocidos con soluciones analíticas. Logran la validación para el caso de inestabilidad debido a “helical buckling”, oscilaciones axiales de una varilla empotrada verticalmente, al igual que la deflexión de una viga empotrada con un extremo libre. En (X. Zhang et al., 2019), los autores validaron el paquete de simulación Elastica con los siguientes casos de estudio: una articulación de codo humana, robots bio-híbridos de tamaño milimétrico cuya función es caminar o nadar, locomoción de una serpiente

tomando en cuenta su musculatura compleja al igual que la replicación de alas de plumas de un ave de tamaño real.

## 3 Modelo de fuerzas y momentos para accionamiento de tipo tendón

### 3.1 Estado del arte

El problema de modelar el accionamiento a una varilla debido a las fuerzas y momentos ejercidos por tendones es un problema que ha sido evaluado previamente. Se trata de plantear cuáles fuerzas y/o momentos están presentes cuando se aplica tensión en un sistema tipo vértebra-tendón. Este tipo de sistema consiste en una serie de “vertebras” que se refiere a elementos espaciados a lo largo de una longitud, cuya estructura permite atravesar un hilo denominado “tendón” a través de ellas, manteniendo el hilo a la misma altura normalmente. Es necesario que, en la última vértebra del sistema, haya algún tipo de fijación del tendón a la vértebra, permitiendo que se pueda tensionar el hilo desde la base y este no se desprenderá del sistema. Este sistema permite aplicar fuerzas y/o momentos distribuidos en las vértebras a lo largo de la longitud de la varilla, partiendo de tensionar el hilo en la base. Esto resulta particularmente atractivo cuando se tiene que controlar la configuración de la varilla flexible en espacios de trabajo no accesibles, o que requieren buena maniobrabilidad.

Revisando trabajos previos, se encuentra que (Li et al., 2018) utilizaron un modelo de cinemática basada en curvatura constante, por lo que definieron los parámetros del arco formado por el robot y mapean las longitudes de los tendones a los parámetros del arco, para luego obtener la configuración final del robot. Aunque puede resultar rápida la solución, es una simplificación y no permite predecir configuraciones que no incluyan curvatura constante. El trabajo de (Chaiopoulos et al., 2019) desarrolla un modelo de viga tipo Euler-Bernoulli, que captura comportamientos de grandes deformaciones utilizando el tensor de deformación de Green-Lagrange al igual que la variación en el módulo de elasticidad debido a grandes deformaciones, en el caso de un accionamiento por tendones en el plano.

Los autores plantean una serie de ocho ecuaciones diferenciales que resuelven con un integrador de ecuaciones diferenciales ordinarias para encontrar ocho variables a lo largo del eje centroidal discretizado hasta llegar a la convergencia, teniendo como entrada dos tensiones que corresponden a dos tendones antagonistas. En el trabajo de (Gao et al., 2017), utilizan un modelo de fuerzas de tendones basado en los ángulos formados por las vértebras. Las fuerzas contempladas incluyen las fuerzas paralelas a la vértebra (fuerza normal) y la fuerza perpendicular a la vértebra (fuerza causada por fricción). El ángulo que utilizan para describir estas fuerzas es el ángulo que tiene una fuerza resultante con la siguiente. Usando esta descripción para las fuerzas de los tendones, utilizan un modelo de varillas de Cosserat para modelar la columna elástica del robot y resolver el sistema de ecuaciones diferenciales ordinarias utilizando condiciones de frontera que incluyen las cargas debido a los tendones en el lado proximal. Para resolver las ecuaciones, utilizaron el método de diferencias finitas con la fórmula de Lobatto IIIa implementada en la función bvp4c en Matlab R2013b. Otro de los aportes es el trabajo de (Kato et al., 2015), donde usan una aproximación de curvatura constante por trozos para formular un mapeo de cinemática directa, que toma como entrada las tensiones en los tendones y como resultado da la configuración del robot, tomando en cuenta la fricción dada en las vértebras y tendón. Para las fuerzas ejercidas por el tendón, utilizan el mismo balance de fuerzas de (Gao et al., 2017), pero usan una ley de Hooke para un resorte rotacional para calcular el ángulo de rotación (usado en el balance de fuerzas) y la curvatura del segmento. Luego concatenan las curvaturas de todos los segmentos para obtener la configuración del robot. El enfoque de (Rucker & Webster III, 2011) es de utilizar la teoría de varillas de Cosserat, utilizando leyes constitutivas lineales elásticas, para modelar la columna elástica del robot. También, modelan el tendón como un hilo extensible utilizando la teoría de Cosserat y derivan las cargas distribuidas que el tendón aplica a la columna elástica, a través de las variables cinemáticas que se pueden incorporar a la varilla

que modela la columna elástica. Por otro lado, (Rao et al., 2021) desarrollan varias teorías y métodos de modelaje, ambos para el robot y para el accionamiento por tendones. Los autores proponen un sistema sin fricción, lo que significa que la componente de fuerza perpendicular a las vértebras es nula. En el caso de curvatura constante, buscan mapear las longitudes de los tendones a los parámetros del arco formado para calcular la curvatura del robot. En otro modelo, encuentran el modelo estático al calcular la fuerza y momento neto entre el tendón y las fuerzas externas, para encontrar las fuerzas y momentos resultantes internos producidos. Estos luego se acoplan a la parametrización de la columna elástica. Es similar al enfoque hecho por (Rucker & Webster III, 2011), pero utilizan un tendón que no es extensible.

## **3.2 Implementación en PyElastica**

### ***3.2.1 Funcionamiento de PyElastica***

Debido a que el enfoque de este trabajo es acoplar las fuerzas de tendones al paquete de simulación PyElastica, se debe explorar cómo funciona el paquete de simulación y cuáles son los datos que se pueden extraer de las simulaciones. Lo que se busca hacer es un enfoque similar a aquel realizado por (Rucker & Webster III, 2011), en el que consideran la fuerza interna en el tendón y luego la aplican a las vértebras. Para hacer esto, calculan el vector unitario que describe la orientación del vector de posición relativa entre vértebras, y este vector lo escalan con la tensión del tendón. Luego continúan su análisis y transforman el sistema al introducir estas ecuaciones arregladas a las ecuaciones de varilla. Este enfoque se implementará para las simulaciones en PyElastica, solamente que se deberían utilizar los datos computados por la simulación y usarlos para calcular las fuerzas, las cuales se aplicarían como fuerzas externas.

Comenzando por analizar el funcionamiento del paquete de simulación PyElastica antes de desarrollar el módulo de forzado por tendones, se hace la observación que PyElastica funciona de manera modular.

Como se puede observar en su documentación (Gazzola Lab, 2024), lo primero que se debe hacer en el código es crear un objeto simulador. Este simulador se va a encargar de recibir todos los parámetros de simulación y todas las varillas a ser simuladas. Cualquiera nueva varilla que se quiera añadir al sistema, se acopla a esta instancia del objeto de simulador. Este objeto se va a encargar de manejar las ecuaciones de gabinete previamente desarrolladas en la sección **2.3 Desarrollo del paquete de simulación Elastica**, al igual que manejar todos los datos asociados a las simulaciones, los cuales van a ser útiles más adelante para desarrollar el módulo de forzado por accionamiento de tendones.

Es importante destacar que todo lo que se añade al simulador, se hace a través de módulos. Es decir, se importan de la librería de Elastica los módulos que se quieren utilizar y estos se aplican mediante métodos que tiene el simulador, tal como: “SIMULATOR.constrain(ROD).using()”, o SIMULATOR.add\_forcing\_to(ROD).using().” Donde se añadirían los módulos como argumentos en “using()”, al igual que cualquier otro parámetros que se requiera.

Luego, se deben crear los objetos de varillas, los cuales serán posteriormente acoplados al simulador. Estos objetos toman como argumento el tipo de varilla (por ahora puede ser una varilla Cosserat derecha, o en forma de aro), al igual que todos los parámetros materiales y geométricos. Los parámetros utilizados para este trabajo se verán a mayor detalle en la sección **3.3 Experimentos y validación**. Las varillas portan una gran cantidad de datos que son calculados conforme a las iteraciones de la simulación que se van computando. Para efectos de este trabajo, los datos de interés son: “position\_collection” y “director\_collection”. El primero de estos dos es una colección de los datos de posición global de cada uno de los nodos en la varilla, capturada en el instante que se inspecciona. Esto quiere decir que en cada intervalo de tiempo en el que se realiza una iteración, se calcula una nueva colección de posiciones globales para los nodos. Estas posiciones se pueden agrupar para obtener la dinámica del

sistema y observar visualmente cómo evolucionó la configuración de la varilla a lo largo del tiempo. Por otro lado, “director\_collection” es una colección de los datos de los vectores directores de los marcos de referencia locales. Estos directores pueden ser agrupados en forma de matriz 3x3, de manera que representen la orientación del marco local de su respectivo nodo.

Siguiendo, se definen las condiciones de frontera. En este trabajo, no se considerarán condiciones de frontera complejas; el caso de interés es de una varilla empotrada, sujeta a accionamiento por tendones, por lo que las condiciones de frontera necesarias son las de una frontera fija y la otra frontera libre. Esto se hace al aplicar una restricción o “constrain” directamente a la varilla que fue creada en el paso anterior. PyElastica cuenta con varias condiciones de frontera predefinidas, y la que se escoge para este trabajo es “OneEndFixedBC”, o condición de frontera fija en un lado.

Después, se aplica un elemento de disipación de energía. Esto no es necesario, sin embargo, ayuda con la estabilidad de las simulaciones y ayuda a acercarse mejor al comportamiento real del material utilizado. De igual manera, PyElastica cuenta con varios tipos de amortiguamiento, y en este caso se utiliza “AnalyticalLinealDamper”, o amortiguador lineal analítico.

También, se puede especificar las uniones que puede haber entre varillas. Sin embargo, en este trabajo el enfoque es con solamente una varilla, por lo que esta parte del flujo de trabajo se ignora.

Posteriormente, se especifica la parte más importante para la simulación: el forzado. Como se explicó previamente, el simulador de Elastica funciona mediante módulos que son importados y posteriormente aplicados a las varillas. Lo útil de que se utilicen módulos es que pueden ser fácilmente acoplados al sistema usando la misma sintaxis para diferentes módulos. Esto quiere decir que el usuario del paquete de simulación Elastica puede utilizar la cantidad de módulos que desee, al igual que crear sus propios módulos para utilizar. En el caso de este

trabajo, se codificó un módulo de forzado que funciona para aplicar las fuerzas de accionamiento por tendones a lo largo de la varilla. Este módulo será visto con mayor detalle la sección **3.2.2 Planteamiento y desarrollo del módulo de accionamiento por tendones**. Además del módulo de forzado de tendones, se debe considerar también el forzado distribuido debido a la gravedad. PyElastica cuenta con varios módulos de forzado preestablecidos, y este es uno de ellos, por lo que se utiliza “GravityForces”, o fuerzas de gravedad, como uno de los módulos importados para utilizar en la simulación y se acopla a la varilla.

Además de los módulos y parámetros para la simulación, se debe configurar la función de “MyCallback”, que funciona como una función que se ejecuta mientras la simulación está andando. Esta función es crucial para el funcionamiento del sistema, puesto que se necesitan datos entre iteraciones de las simulaciones, y esta función de “MyCallback” podrá extraer esos datos y enviarlos a donde se necesiten. Si no se configura esta función de “MyCallback”, la simulación se ejecutará como un proceso, y no es hasta que esta termine que se podrán ejecutar otros procesos, por lo que se emplea el uso de la función “MyCallback” para que realice procesos durante la simulación. Además, esta función es la que se encargará de recopilar los datos a lo largo de la simulación, para tener la colección final de datos de posición y poder lograr una visualización de la evolución del sistema, por lo que es indispensable en este trabajo.

De último, se realiza la finalización del simulador. Esto se hace mediante el método “finalize()”, que se encarga de coleccionar todos los datos, condiciones, fuerzas, parámetros, y prepara el sistema para la simulación. Luego se ejecuta la función “integrate()”, que se encarga de evolucionar el sistema a través del tiempo. Se están desarrollando diferentes integradores temporales (Gazzola Lab, 2024), sin embargo el recomendado es el que se discutió en la sección **2.3 Desarrollo del paquete de simulación Elastica**, el integrador de segundo orden simpléctico Verlet.

Posterior a terminar la simulación, se debe hacer el postprocesado de los datos. Para efectos de este trabajo, el interés es observar la evolución que tuvo el sistema en el tiempo. Lo que se hace con los datos computados es generar un video mediante la librería moviepy (Zulko, n.d.), tomando como dato la colección de posición de cada uno de los nodos en la varilla, a través del tiempo.

### ***3.2.2 Planteamiento y desarrollo del módulo de accionamiento por tendones***

Como se mencionó en la sección **3.1 Estado del arte**, este problema de accionamiento por tendones se ha trabajado en trabajos previos. El problema yace en la naturaleza del sistema de robótica continuo: la tensión aplicada a los tendones genera fuerzas y momentos a lo largo de la longitud del robot (en este caso la varilla flexible), y la deflexión que este sufre afecta directamente estas fuerzas y momentos. Las fuerzas y momentos se ven afectados de manera no lineal, y varían para cada una de las vértebras a lo largo del robot, lo que lo vuelve un problema cuya descripción no es necesariamente trivial.

Trabajos previos atacan estos problemas con diferentes modelos de forzado, sin embargo, estos modelos son puramente matemáticos y a menudo se enfocan en el caso del robot accionado en el plano, en lugar de un espacio tridimensional. Dado que uno de los enfoques de este trabajo es utilizar el paquete de simulación PyElastica, se debe crear un nuevo enfoque, utilizando los datos que se computan en las simulaciones realizadas para actualizar las fuerzas que actúan en las vértebras. Como se explicó en la sección **3.2.1 Funcionamiento de PyElastica**, las fuerzas externas se aplican mediante módulos de forzados que se importan al simulador. El paquete de simulación no cuenta con un módulo de forzado por accionamiento de tendones, por lo que este trabajo intentará llenar ese vacío. La idea sería de utilizar los datos de posición y orientación de los nodos, en especial los de las vértebras, e ir ajustando los valores de fuerzas y momentos aplicados

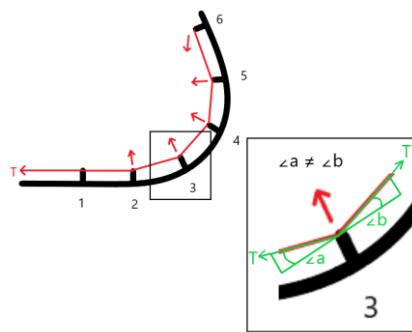
en las vértebras, para que de esta manera estén acoplados directamente a la deflexión que tiene el robot (o varilla flexible) a lo largo de la evolución del sistema.

Antes de comenzar con el planteamiento y subsecuente desarrollo del módulo de forzado externo por accionamiento de tendones, es importante tomar en cuenta algunas consideraciones.

- Primero, se considera que no hay fricción actuando en el medio de tendón-vértebra. Esta es una consideración aceptable cuando se utilizan materiales que tienen bajo coeficiente de fricción (Rucker & Webster III, 2011), puesto que el orden de impacto de la fricción es mucho menos aquella de la actuación (Rao et al., 2021).
- Segundo, se considera que el material de la varilla flexible que forma parte de la columna vertebral del robot continuo es flexible. Esto hace referencia a que puede ocurrir en grandes desplazamientos y deflexiones, sin sufrir deformaciones plásticas, esto sin tener que ser forzada enormemente. Esto es importante porque la premisa de un robot continuo flexible requiere que el robot pueda contorsionarse con grandes curvaturas y que pueda regresar a su configuración original sin tener que ejercer tanta fuerza. Además, como se mencionó en la sección **2.3 Desarrollo del paquete de simulación Elastica**, el software no es compatible con materiales que tienen un módulo de elasticidad muy elevado.
- Tercero, se considera que todas las vértebras utilizadas a lo largo del robot son del mismo tamaño, y se colocan a lo largo del robot en una línea recta a partir de su configuración original. Además, su centro de masa se debe alinear con el centroide de la varilla flexible o columna del robot continuo. Esto se hace por simplicidad y eficacia para la integración al entorno de simulación.

Para comenzar con el planteamiento, se debe entender cómo aparecen las fuerzas y momentos de los tendones físicamente en las vértebras del robot continuo. Como se puede observar en la **Figura 3-1**, la tensión aplicada en el

tendón causa que la varilla flexible se flexione, lo cual logra hacer mediante la interacción entre tendón y vértebra. Aunque la **Figura 3-1** esté dibujada en un plano bidimensional por motivos de simplicidad, este trabajo considera un sistema tridimensional, lo que lleva a contemplar un tercer vector componente a las fuerzas y momentos, lo que completa el vector 3D resultante actuando en las vértebras.



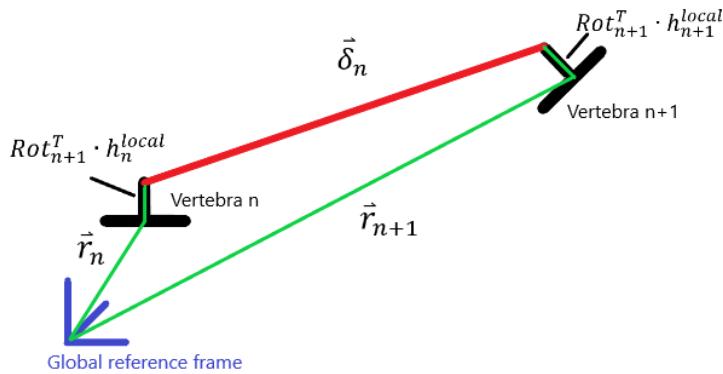
**Figura 3-1.** Ángulos formados entre tendón y vértebra son diferentes en ambos lados de la vértebra.

Comenzando con el análisis de las fuerzas en las vértebras internas (todas excepto la última vértebra en el extremo libre de la varilla), existe una fuerza que es una resultante de ambos vectores de fuerza en cada lado de la vértebra. Estos vectores de fuerza tienen componentes que dependen del ángulo formado entre la vértebra misma y el tendón. Estos ángulos no necesariamente son los mismos en cada lado de las vértebras. Para calcular las fuerzas resultantes en cada una de las vértebras, es muy importante poder describir las componentes de las fuerzas correctamente, independiente de la configuración que tenga la varilla. De manera similar, la última vértebra en el extremo libre debe ser sujetada a la misma lógica, solamente que tendría un solo vector de fuerza actuando sobre ella, puesto que no hay otra vértebra después de esta, que en torno se convierte en el vector de fuerza resultante para esta vértebra.

Se pueden considerar varios enfoques para lograr esta meta. En este trabajo se consideraron tres enfoques, dos de los cuales fueron descartados. El primer enfoque consiste en utilizar trigonometría para calcular continuamente los ángulos

formados entre vértebra y tendón, y de esta manera calcular las componentes de los vectores de fuerza con funciones trigonométricas. Este enfoque resultó ser factible y estable en el caso que se consideren fuerzas en el plano bidimensional; cuando se tiene alguna fuerza que sale del plano, la simulación se tiende a desestabilizar o da resultados que no están alineados con la realidad. El segundo enfoque consiste en formar las matrices de rotación para cada vértebra y encontrar las matrices de rotación relativas entre vértebras, de esta manera apuntando a construir el vector de fuerza con la orientación relativa entre vértebras. Sin embargo, utilizar las matrices de rotación solamente, no es suficiente para capturar la interacción vértebra-tendón, puesto que las rotaciones relativas entre vértebras no necesariamente son las mismas que hay entre tendón y vértebra, especialmente cuando se trata de tensiones elevadas en el tendón que causan grandes deflexiones en la varillas.

El tercer enfoque, que fue el que se escogió para este trabajo utiliza la información de posición que se calcula en las simulaciones hechas por el software, específicamente “position\_collection”, como se mencionó en la sección **3.2.1 Funcionamiento de PyElastica**. Debido a la suposición de fricción negligible en el sistema, la tensión en el tendón permanece constante, por lo tanto, si se encuentra la orientación del tendón entre vértebras, también se encuentran todos los vectores de fuerza aplicados en cada vértebra. Entonces, se utiliza el dato de posición de cada vértebra en el sistema,  $\vec{r}_n$ , al igual que la altura de la vértebra,  $h_n^{local}$ , un vector se puede construir que describe el cambio de posición desde la altura de una vértebra hasta la próxima, como se puede ver en la **Figura 3-2**:



**Figura 3-2.** Obteniendo vector de posición relativa entre vértebras,  $\vec{\delta}_n$ , mediante suma vectorial.

El vector de altura de la vértebra,  $\vec{h}_n^{local}$  se expresa en términos de las coordenadas locales de la vértebra en cuestión. Dado que el vector posición  $\vec{r}_n$  está expresado en el marco de referencia global, el vector de altura de la vértebra debe ser transformada al marco de referencia global antes de proceder con la suma vectorial, esto se hace mediante las matrices de rotación que se obtienen mediante los datos de “director\_collection” que se calculan en cada iteración de la simulación. La matriz de rotación construida por estos directores es una matriz de rotación que describe la orientación del marco de referencia con respecto al marco de referencia local, denotada  $Rot_n$ . En la siguiente fórmula,  $Rot_n$  es traspuesta para lograr una correcta rotación del vector  $h_n^{local}$ , del marco local al global:

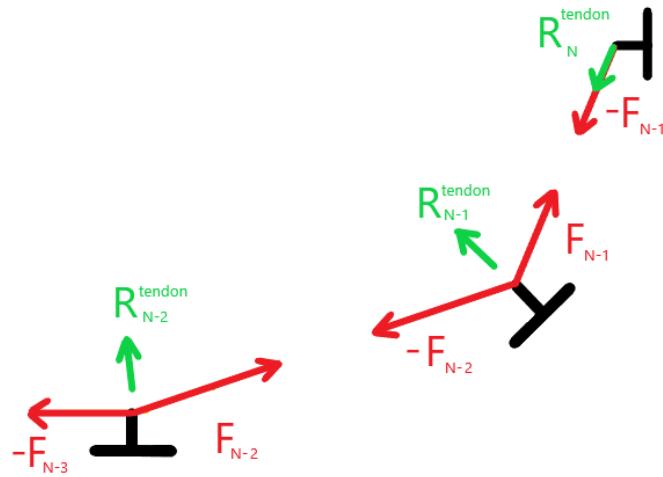
$$\vec{\delta}_n = (\vec{r}_{n+1} + Rot_{n+1}^T \cdot \vec{h}_{n+1}^{local}) - (\vec{r}_n + Rot_{n+1}^T \cdot \vec{h}_n^{local}) \quad (20)$$

Donde  $\vec{\delta}_n$  es el vector de posición relativa entre una punta de vértebra hasta la siguiente punta, como se puede ver en la **Figura 3-2**. Una vez que se calcule el vector  $\vec{\delta}_n$ , se debe calcular su vector con norma unitaria, para que luego este sea escalado por la magnitud de la tensión aplicada al tendón. Para hacer esto, se aplica la siguiente operación:

$$\vec{\zeta}_n = \frac{\vec{\delta}_n}{\|\vec{\delta}_n\|}$$

$$\vec{F}_n = T \cdot \vec{\zeta}_n \quad (21)$$

Donde  $\vec{\zeta}_n$  es el vector con norma unitaria,  $\vec{F}_n$  es el vector de fuerza que actúa en la vértebra  $n$ , apuntando hacia la siguiente vértebra, puesto que se utilizó  $\vec{\delta}_n$  para describir su orientación, y  $T$  es la tensión aplicada al tendón. Luego, para encontrar la fuerza resultante, se deben sumar los dos vectores de fuerza que actúan en ambos lados de la vértebra  $n$ , puesto que ambos vectores actúan sobre la punta de la vértebra a la misma vez, como se puede ver en la **Figura 3-3**. Se nota que, en cada vértebra, hay un vector de fuerza positivo y otro negativo, esto es porque uno de estos vectores se describe con  $\vec{\delta}_n$  y el otro con  $-\vec{\delta}_{n-1}$ .



**Figura 3-3.** Esquema que muestra cómo se presentan los vectores de fuerzas de tendones resultantes.

Entonces, para calcular la fuerza resultante de tendón en cada vértebra,  $\vec{R}_n^{tendon}$ , se tiene lo siguiente:

$$\vec{R}_n^{tendon} = \vec{F}_n - \vec{F}_{n-1} \quad (22)$$

Existe una excepción, que es la última vértebra del extremo libre. Debido a que no existen ninguna vértebra que le sigue a esta, su vector  $\vec{\delta}_n$  es nulo, lo que hace que su vector de norma unitaria no esté definido. En este caso, se obvia el cálculo del vector  $\vec{\delta}_n$  y su vector de norma unitaria se hace un vector cero, para que su vector de fuerza en esa dirección,  $\vec{F}_N$  sea igual a cero. Esto se puede ver en la **Figura 3-3.**

Para tomar en cuenta el peso que tienen las vértebras, se añade otra fuerza, que es simplemente un vector con el peso de la vértebra entera:

$$W_n = [0.0 \quad 0.0 \quad m_{vertebra} * g] \quad (23)$$

La consideración de que el centro de masa de las vértebras debe alinearse con la línea centroidal de la varilla resulta particularmente útil en esta parte del desarrollo, puesto que hace que no se generen momentos debido a una excentricidad de centro de masa de las vértebras. El peso se coloca en la tercera componente del vector, lo que referencia al eje  $z$  del marco de referencia global. Entonces, el vector de fuerza resultante en cada nodo de vértebra es:

$$\vec{R}_n = \vec{F}_n - \vec{F}_{n-1} + W_n \quad (24)$$

Ahora, es necesario analizar los momentos que se generan a lo largo del robot. Debido a que hay fuerzas en el espacio que se aplican a una distancia de la línea centroidal (se aplican a la altura de la vértebra), se generan momentos en estas vértebras y se aplican respectivamente. Por lo tanto, es igual de importante considerar el efecto de estos momentos. Para calcular el momento generado en

cada vértebra debido a las fuerzas ejercidas por los tendones, se debe calcular el producto cruz entre el vector de altura de vértebra,  $\vec{h}_n^{local}$ , y la fuerza resultante por tendón en cada vértebra,  $\vec{R}_n^{tendon}$ . Antes de realizar esta operación vectorial, se debe transformar el vector  $\vec{R}_n^{tendon}$  al marco de referencia local, para obtener  $\vec{R}_n^{local}$ , puesto que está definido en el marco global. Esto es para ayudar con el manejo de momentos en PyElastica, puesto que el software maneja fuerzas en el marco de referencia global y momentos en el marco de referencia local. Para lograr esta rotación, se premultiplica el vector  $\vec{R}_n^{tendon}$  por la matriz de rotación asociada a la vértebra,  $Rot_n$ . Es importante destacar que no se considera la resultante final que considera el peso de la vértebra, puesto que esta no genera momentos en la línea centroidal de la varilla. Entonces, realizando las operaciones:

$$\vec{R}_n^{local} = Rot_n \cdot \vec{R}_n^{tendon} \quad (25)$$

Ahora calculando el momento,  $\vec{M}_n$ , en cada vértebra:

$$\vec{M}_n = h_n^{local} \times \vec{R}_n \quad (26)$$

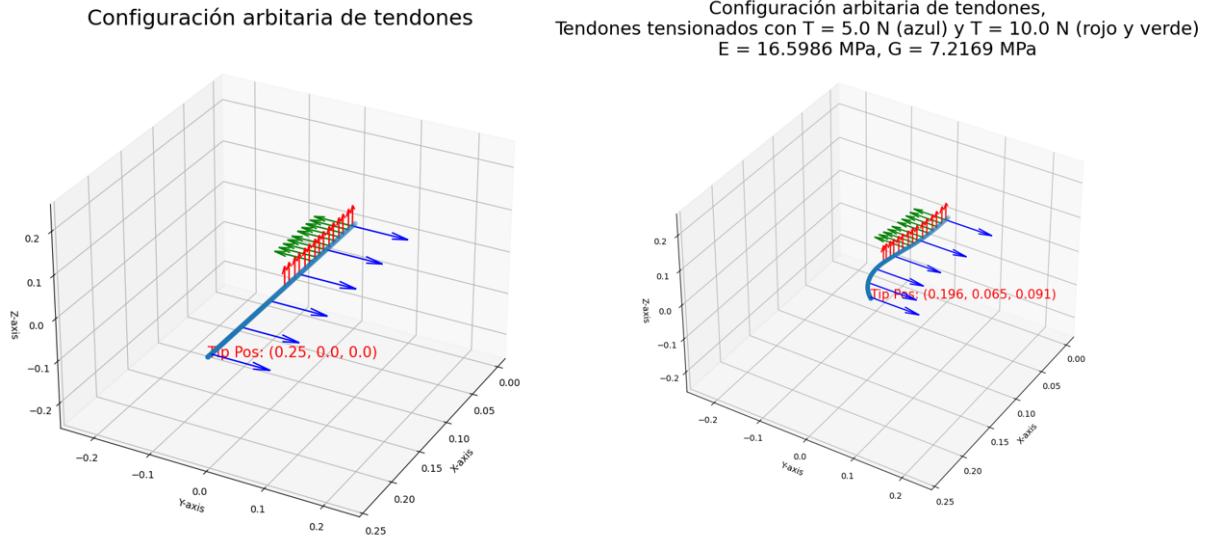
El vector  $\vec{M}_n$  se aplica localmente a cada vértebra respectivamente.

### **3.2.3 Estructura como módulo de forzado externo para el accionamiento por tendones**

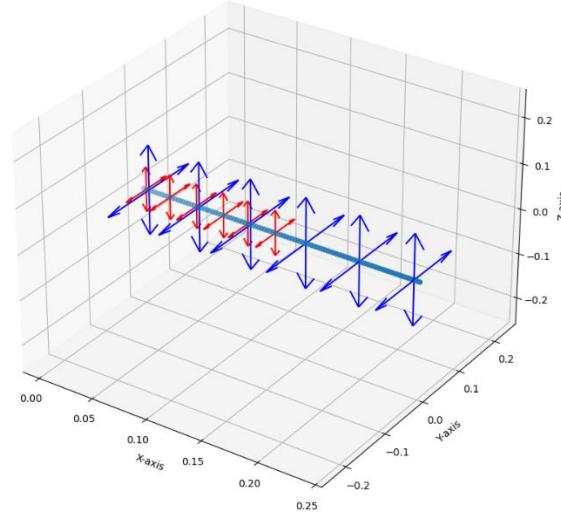
Para integrar el módulo de accionamiento por tendones en el software de simulación PyElastica, se deben seguir las guías propuestas en la documentación de este. Comenzando, se indica (Gazzola Lab, 2024) que todas las clases de forzado externo que sean creadas, deben heredar la clase default, llamada “NoForces”. Esta clase tiene la estructura y propiedades necesarias para que sea

posible ser utilizada por el simulador de PyElastica a la hora de aplicar las fuerzas y momentos necesarios.

Para el caso del módulo de forzado externo para el accionamiento por tendones, se debe destacar que se programaron dos módulos: “TendonForces” y “QuadTendonForces”. El propósito de tener dos módulos es para que se puedan utilizar en dos casos: “TendonForces” está previsto para el caso en el que se quiera hacer una simulación con tendones personalizados, con un accionamiento de tipo abierto. Es decir, se especifican los tendones, sus respectivas configuraciones y tensiones, y se ejecuta la simulación sin tener ningún tipo de control de posición. Por otro lado, “QuadTendonForces” está previsto para el caso en el que se quiera controlar la posición del robot continuo. El nombre indica que se tienen cuatro tendones (uno arriba, uno abajo, uno a la izquierda y otro a la derecha del robot) y, además, se tiene otro conjunto de tendones antagonistas que deberían ser más cortos (se menciona que deberían serlo, pero esto queda a disposición del usuario. Son tendones antagonistas porque se accionan en orientaciones contrarias a los tendones largos, para lograr un mayor rango de posiciones posibles del robot). Una representación visual de los dos tipos de arreglo para los módulos de forzado por tendones se puede ver a continuación en la **Figura 3-4** y **Figura 3-5**:



**Figura 3-4.** Representación visual de un arreglo formado por TendonForces, donde se tienen 3 hileras de tendones de diferentes tamaños y longitudes. En el caso del sistema manual, el usuario tiene la libertad de escoger cualquiera configuración de tendones deseada.



**Figura 3-5.** Representación visual del arreglo formado por QuadTendonForces, donde se tienen dos conjuntos de vértebras: 4 hileras de vértebras largas (color azul) y 4 hileras de vértebras cortas (color rojo). Este se utiliza en el caso del

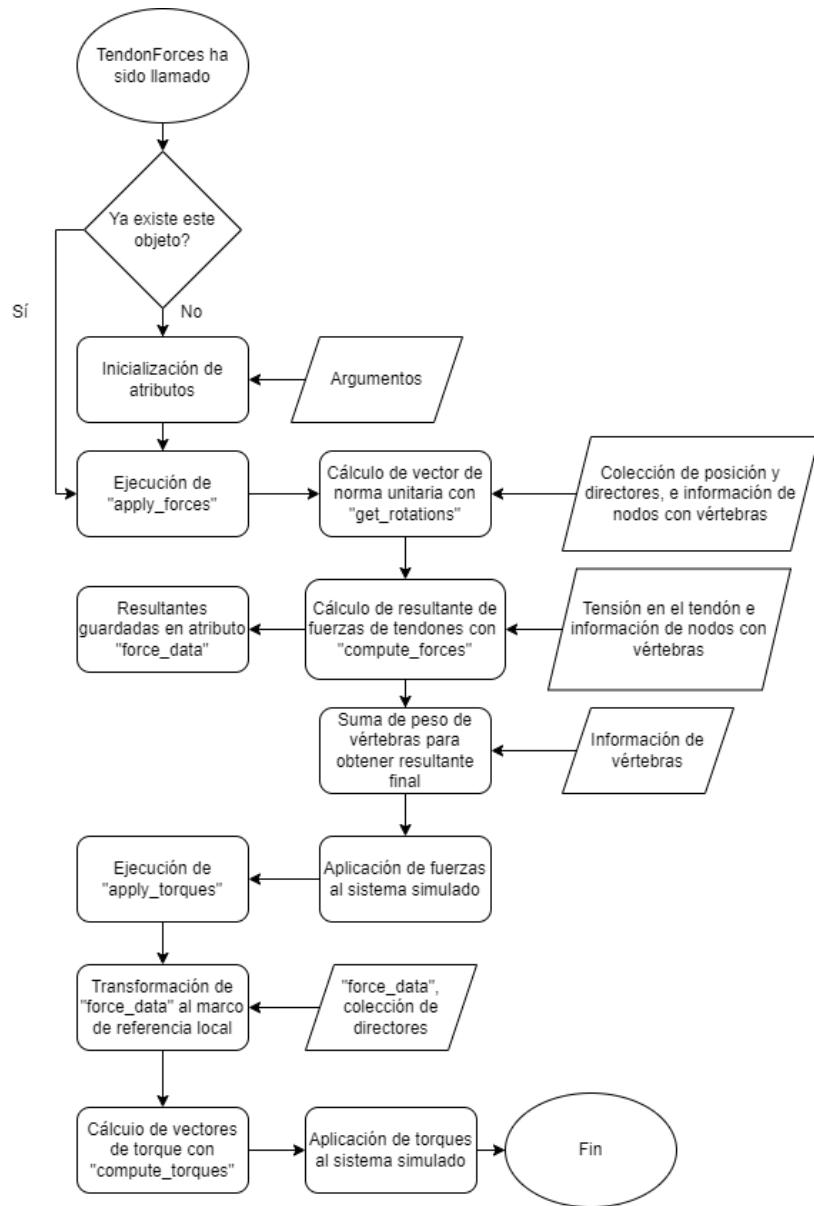
sistema de control, donde no se puede cambiar la configuración del robot, más sí sus parámetros.

### 3.2.3.1 *Forzado manual y personalizado con TendonForces*

La clase creada de “TendonForces” crea un objeto en el simulador que hace referencia a una sola hilera de vértebras con su respectivo tendón. Si se desean utilizar múltiples tendones, solamente se debe aplicar múltiples veces este módulo de forzado, con sus parámetros respectivos. El objeto creado al crear una instancia de esta clase se inicializa tomando como argumento los siguientes valores:

- “vertebra\_height”: altura de vértebras
- “num\_vertebrae”: cantidad de vértebras
- “first\_vertebra\_node”: nodo en la varilla donde se tiene la primera vértebra
- “final\_vertebra\_node”: nodo en la varilla donde se tienen la última vértebra
- “vertebra\_mass”: la masa que tiene cada vértebra completa
- “tension”: la tensión que será aplicada al tendón
- “vertebra\_height\_orientation”: vector unitario expresado localmente que describe la orientación que tiene la vértebra
- “n\_elements”: la cantidad de nodos que hay en la varilla

Se puede apreciar el funcionamiento de este módulo de forzado con el diagrama a continuación, seguido de una descripción de su funcionamiento:



**Figura 3-6.** Diagrama de flujo para el funcionamiento general de TendonForces.

La inicialización de “TendonForces” consiste en tomar como atributos del objeto creado, varios de los valores introducidos como argumentos. Además, se hacen algunos cálculos con estos valores obtenidos de los argumentos para definir otros atributos del objeto. Estos atributos de inicialización son muy importantes para el funcionamiento del módulo de forzado.

Siguiendo, como es especificado por la documentación de PyElastica (Gazzola Lab, 2024), se deben definir dos métodos: “apply\_forces” y “apply\_torques”. Estos métodos serán ejecutados por el simulador de PyElastica en cada iteración de la simulación, y se encargarán de proveer los valores de forzado al simulador. Para mejorar la velocidad a la cual se realizan los cálculos necesarios para proveer los valores de forzado, los métodos “apply\_forces” y “apply\_torques” no realizan computaciones como tal, sino que delegan la responsabilidad a otros métodos del objeto que están decorados con el decorador njit de Numba (Anaconda Inc., n.d.). La razón por la cual se escogió hacer esto es porque el decorador njit (Numba Just-In-Time) convierte el método decorado en lenguaje de máquina. Esto agiliza enormemente las computaciones realizadas, por lo que conviene utilizar el decorador en lugar de que se hagan en Python.

Entonces, el método de “apply\_forces” ejecuta el método de “get\_rotations”, el cual está decorado por njit, y cuya función es de calcular el vector con norma unitaria para cada tramo de tendón entre cada una de las vértebras, para luego retornar un arreglo conteniendo esta información que será utilizada en el método “apply\_forces”. Para lograr estos cálculos, toma como argumento la colección de datos de posición de los nodos de la varilla, “position\_collection”, la colección de datos de los directores locales de los nodos de la varilla, “director\_collection”, los nodos que tienen vértebras, “vertebra\_nodes”, y el vector local que describe la orientación y altura que tienen las vértebras, “vertebra\_height\_vector”. Los cálculos realizados son los de la ecuación (20).

Siguiendo, el método de “apply\_forces” ahora ejecuta el método de “compute\_forces”, el cual también está decorado por njit, y cuya función es tomar el arreglo de vectores unitarios obtenidos en “get\_rotations”, al igual que la tensión que será aplicada al tendón y los nodos que tienen vértebras, “vertebra\_nodes”, para luego escalar los vectores unitarios por la tensión aplicada y realizar la suma vectorial de los dos vectores de fuerza en cada vértebra, retornando el resultado

de esta suma, que se guarda como atributo del objeto bajo el nombre “force\_data”. Los cálculos realizados son los de las ecuaciones (21) y (22).

Antes de aplicar las fuerzas al sistema, se le suma a “force\_data” la fuerza de la gravedad a cada nodo que tiene una vértebra, como se hace en las ecuaciones (23) y (24). Después de realizar este paso, se aplica finalmente el arreglo de fuerzas al sistema.

Ahora, siguiendo con la parte de los momentos, el método “apply\_torques” se encarga de aplicar los momentos generados en cada vértebra. Debido a que PyElastica maneja los momentos y torques en el marco de referencia local, se debe transformar las fuerzas de los tendones calculadas previamente con (22), desde el marco de referencia global al local. Para hacer esto se emplea la ecuación (25): se multiplica la matriz de rotación asociada a cada vértebra, que describe el marco de referencia local con respecto al global, y esta matriz se posmultiplica por las respectivas fuerzas obtenidas previamente, almacenadas en el atributo “force\_data”. Cabe destacar que “force\_data” no incluye la fuerza debido a gravedad del peso de la vértebras, porque esta no genera un momento. Las fuerzas en el marco local se almacenan en la variable local “transformed\_force\_data”.

Teniendo las fuerzas debido a los tendones transformadas al marco de referencia local, se procede con ejecutar el método “compute\_torques”, que se encarga de calcular los torques que serán aplicados a los nodos con vértebras debido a las fuerzas de los tendones en ellas, cuyos valores son calculados con (26). Este método está decorado con njit por las razones mencionadas previamente. Como argumento, el método recibe “vertebra\_height\_vector”, “vertebra\_nodes”, “transformed\_force\_data”, “n\_elements” y “system.external\_torques”. El último argumento es propio del sistema siendo simulado por PyElastica, y se le pasa a “compute\_torques” para que aplique los torques directamente dentro del método decorado por njit. Esto no se hizo en “apply\_forces” porque era necesario retornar las fuerzas de los tendones y almacenarlas en un atributo, “force\_data” que se

utiliza en “compute\_torques”, lo cual no se puede realizar si el método está decorado por njit.

Habiendo aplicado ambas las fuerzas y momentos a los nodos que tienen vértebras, se completa la función del módulo de forzado. Este módulo de forzado está acoplado con la interfaz gráfica de usuario manual, que se detalla en la sección **4.1 Nodo de interfaz gráfica (GUI)**. A continuación, se hablará del segundo módulo de fuerzas, “QuadTendonForces” y cómo se diferencia a “TendonForces”.

### ***3.2.3.2 Forzado retroalimentado con un controlador usando QuadTendonForces***

El módulo de forzado que será descrito en esta sección, “QuadTendonForces”, es muy similar al previo “TendonForces”. Fundamentalmente, difieren en el hecho de que “QuadTendonForces” no está diseñado para que se pueda modificar la configuración de los tendones: siempre habrá un total de ocho tendones en el sistema. Los parámetros de estos tendones y vértebras sí pueden ser modificados (un conjunto de parámetros para los tendones largos, y un conjunto de parámetros para los tendones cortos), sin embargo, la tensión que se aplica a los tendones se dictará mediante el nodo controlador, el cual se verá a detalle en la sección **4.4 Nodo de control**. Es muy importante destacar que, por razones de simplicidad y compactación del código, solamente habrá cuatro tendones activados como máximo en un instante dado: uno vertical largo, uno horizontal largo, uno vertical corto y uno horizontal corto. Cuáles tendones se utilicen también es función del nodo controlador decidir, que, dependiendo del cuadrante que se encuentre el punto deseado, modifica los tendones activos.

El funcionamiento de “QuadTendonForces” inicia similar a “TendonForces”. Primero, PyElastica crea una instancia del objeto usando como argumento los siguientes valores:

- “vertebra\_height\_long”: altura de vértebras de los tendones largos
- “num\_vertebrae\_long”: cantidad de vértebras del tendón largo

- “first\_vertebra\_node\_long”: nodo en la varilla donde se tiene la primera vértebra de los tendones largos
- “final\_vertebra\_node\_long”: nodo en la varilla donde se tienen la última vértebra de los tendones largos
- “vertebra\_mass\_long”: la masa que tiene cada vértebra completa de los tendones largos
- “vertebra\_height\_short”: altura de vértebras de los tendones cortos
- “num\_vertebrae\_short”: cantidad de vértebras de los tendones cortos
- “first\_vertebra\_node\_short”: nodo en la varilla donde se tiene la primera vértebra de los tendones cortos
- “final\_vertebra\_node\_short”: nodo en la varilla donde se tienen la última vértebra de los tendones cortos
- “vertebra\_mass\_short”: la masa que tiene cada vértebra completa de los tendones cortos
- “n\_elements”: la cantidad de nodos que hay en la varilla

De igual manera, se inicializan los atributos del objeto con los mismos valores de los argumentos para ser utilizados luego en el código. También se calculan valores tales como “vertebra\_weight\_vector\_long” y “vertebra\_weight\_vector\_short” que son los vectores que contienen la información del vector de fuerza debido a gravedad de cada vértebra y “vertebra\_nodes\_long” y “vertebra\_nodes\_short” que contienen la información de los nodos que tienen vértebras a lo largo de la varilla.

De manera similar a “TendonForces”, los métodos que realizan la mayoría de los cálculos se decoran con njit para mejorar la rapidez de computación. En “apply\_forces”, se calcula de igual manera el vector de norma unitaria que contiene la información de la orientación de las fuerzas de tendones para los cuatro tendones activos en ese momento. Luego, se repite el mismo procedimiento que se utilizó para obtener “force\_data” en TendonForces: se utilizan los datos respectivos de los tendones y se envían individualmente al método “compute\_forces”, que calcula entonces los siguientes valores que se guardan como atributos del objeto: “force\_data\_vertical\_long”,

“force\_data\_horizontal\_long”, “force\_data\_vertical\_short” y “force\_data\_horizontal\_short”. Ahora, las fuerzas de los tendones largos se suman junto con “vertebra\_weight\_vector\_long” y este nuevo arreglo aplica al sistema. También se hace lo mismo con los tendones cortos y su respectivo “vertebra\_weight\_vector\_short”. La razón por la cual se suman las fuerzas separadamente en base a si son de tendón corto o largo es porque está la posibilidad de que las vértebras de los tendones cortos y los largos no coincidan, y por ende deben ser sumadas separadas y aplicadas al nodo correspondiente.

Ahora, se sigue con “apply\_torques”. De igual manera a “TendonForces”, se requiere transformar las fuerzas de los tendones al marco de referencia local. Se aplica el mismo procedimiento de posmultiplicar la matriz de rotación asociada a cada vértebra por las fuerzas asociadas a cada vértebra. Esto se hace de manera separada, para que las fuerzas transformadas estén diferenciadas por orientación (vertical o horizontal) y longitud (largas o cortas). Para calcular los vectores de torque para las vértebras, se utiliza el método “compute\_torques”, decorado por njit. En el caso de “QuadTendonForces”, el método “compute\_torques” está modificado para acomodar la mayor cantidad de tendones. Este método toma como argumento el vector que detalla la altura y orientación de las vértebras verticales y horizontales, el arreglo que detalla cuáles nodos tienen las vértebras, la data de los vectores de fuerza transformados al marco local para las vértebras verticales y horizontales y también la cantidad de nodos que hay en el sistema, “n\_elements”. El método “compute\_torques” se ejecuta dos veces: una vez para los tendones cortos y una vez para los tendones largos, con sus argumentos correspondientes. Los resultados retornados, “apply\_torque\_long” y “apply\_torque\_short” se suman para obtener el momento final que se aplica al sistema.

Una de las grandes diferencias de “QuadTendonForces” es que contiene un método escrito especialmente para permitir que el nodo controlador pueda influir en la configuración de tendones escogidos y la tensión aplicada a estos. Este es

el método de “update\_tendon\_tension”. Lo que hace este método es actualizar los datos de la configuración de los tendones, es decir, escoge cuáles tendones son los que están actualmente activos. Además, actualiza los valores de las tensiones que se aplican a estos tendones activos. Este método es parte del objeto creado por PyElastica cuando inicia la simulación, por lo que puede ser accedido si se conoce su ubicación en la memoria. Mayor detalle de este funcionamiento se puede encontrar en la sección **4.2 Nodo de simulación**.

### **3.3 Experimentos y validación**

Debido a que el modelo de fuerzas y momentos para accionamiento mediante tendones desarrollado en este trabajo no ha sido comprobado físicamente para observar su similitud con un sistema real, es necesario realizar experimentos para lograr llegar a un consenso acerca de su validez como modelo, y su precisión en la simulación de un robot accionado por tendones.

Para llevar a cabo esta validación, se diseñaron tres experimentos. Los primeros dos experimentos se encargan de calibrar el sistema físico con el simulado, para asegurar que los parámetros y condiciones de la simulación son los mismos, o muy cercanos, del sistema físico. El tercer experimento compara los resultados de posición en el espacio tridimensional obtenidos con diferentes cargas de tensión en los tendones, para el caso físico y el caso simulado. En base a los resultados obtenidos del tercer experimento, se concluye acerca de la validez del modelo desarrollado en este trabajo.

De manera general, los experimentos tienen las siguientes metas: el primer experimento busca calibrar el módulo de elasticidad de la varilla en la simulación, de manera que la deflexión que esta tenga bajo su propio peso sea la misma que la observada en el sistema real. El segundo experimento busca calibrar el módulo de rigidez al cortante de la varilla en la simulación, de manera que la deflexión que esta tenga bajo su propio peso y la fuerza ejercida por uno de los tendones laterales (que causa una ligera torsión en la varilla) sea la misma que la observada

en el sistema real. El tercer experimento busca validar el modelo de fuerzas y momentos para accionamiento mediante tendones desarrollado en la sección

**3.2.2 Planteamiento y desarrollo del módulo de accionamiento por tendones** e implementado como visto la sección **3.2.3 Estructura como módulo de forzado externo para el accionamiento por tendones**, al comparar las poses que tiene el sistema real con el sistema simulado, ante el propio peso del sistema y la acción de dos tendones perpendiculares con cargas de tensión variadas.

Antes de proceder, se deben hacer algunas observaciones. Es muy importante que la simulación esté correctamente calibrada con los experimentos físicos, por lo que las condiciones de simulación deben ser lo más cercanas posibles a los experimentos. Dado que los experimentos No. 2 y No. 3 requieren utilizar tendones y vértebras, es necesario que la calibración del módulo de elasticidad de la varilla (realizado en el experimento No. 1) se haga contemplando las vértebras ya pegadas al cilindro de silicona, puesto que esto le añade rigidez al sistema. También, por naturaleza del polímero utilizado (cilindro de silicona), este exhibe comportamiento de “creep”, por lo cual es muy importante estar atento de este fenómeno y permitir que el material de la varilla se relaje sin esfuerzos antes de proceder con cualquier forzado y consecuente medición.

Además, por temas de imperfección en la confección del sistema para realizar los experimentos, existen desfases en las mediciones tomadas: para el eje X (dirección axial) hay un desfase de -0.3 cm, para el eje Y (dirección horizontal) hay un desfase de +2.0 cm y para el eje Z (dirección vertical) hay un desfase de +0.3 cm. Estos desfases se toman en cuenta para el análisis de los resultados.

### **3.3.1 Objetivos de los experimentos**

El experimento No. 1 consiste en determinar correctamente el módulo de elasticidad (young's modulus), de manera que las deflexiones del sistema físico ante su propio peso coincidan con las simulaciones realizadas en PyElastica con las mismas condiciones. De esta manera se coincide con los parámetros del

experimento y los que se definen en el software de simulación para proceder con el experimento No. 2 y No. 3, que se enfocan en el comportamiento de la varilla ante fuerzas de tendón.

El experimento No. 2 consiste en determinar correctamente el módulo de rigidez al cortante (shear modulus), de manera que coincida con las simulaciones realizadas en PyElastica. En este experimento, se calibrará el módulo de rigidez al cortante para que la deflexión de la varilla ante su propio peso y el forzado de un tendón lateral horizontal sea la misma que la que se computa en la simulación de PyElastica bajo las mismas condiciones. De esta manera se coincide con los parámetros del experimento y los que se definen en el software de simulación para proceder con el experimento No. 3.

El experimento No. 3 tiene como propósito proveer datos reales de la deflexión que tiene el cilindro de silicona ante las fuerzas y momentos ejercidos por dos tendones y el propio peso del cilindro de silicona. Esto funcionará como respaldo para verificar el planteamiento del modelo de fuerzas y momentos de una actuación tipo tendón en una varilla suave como la es el cilindro de silicona. Esto permitirá validar los resultados obtenidos a futuro mediante el software de simulación PyElastica, usando el modelo de fuerzas y momentos planteados para una entrada tipo tendón.

### **3.3.2 Calibración de la simulación**

Antes de llevar a cabo el procedimiento de los experimentos, es importante entender cómo funcionaría la calibración del modelo simulado. El funcionamiento es el siguiente:

1. Se define la función que describe el sistema, en este caso la función escogida es la simulación de PyElastica. PyElastica recibe valores de entrada y computa las simulaciones, en este caso se tomará el valor final de la deflexión de la punta de la varilla simulada como valor de salida.

2. Se estructura una función aparte que se encarga de ejecutar varias simulaciones hasta llegar a los valores deseados. Esto se hará mediante la función `fsolve` de SciPy (Virtanen et al., 2020) que se enfoca en encontrar raíces para funciones no lineales.
3. Como entrada, `fsolve` recibirá un valor inicial para el módulo de elasticidad (para el experimento No. 1) o para el módulo de rigidez al cortante (para el experimento No. 2). Después de haber ejecutado una simulación mediante PyElastica, se tomará el valor final de la deflexión de la punta y se le restará el valor deseado de la deflexión de la punta, en este caso el valor que fue medido en los experimentos corregido para tomar en cuenta su desfase.
4. El paso 3 se repite automáticamente debido al funcionamiento interno de `fsolve`, ajustando el parámetro de entrada respectivo hasta llegar a una raíz en la función proporcionada (en este caso hasta llegar a la deflexión deseada).
5. El parámetro obtenido se imprime y se anota para ser utilizado en los experimentos subsecuentes.

### **3.3.3 Materiales**

A continuación, se listan los materiales necesarios para llevar a cabo los tres experimentos:

- 2 cilindros de silicona
  - 27.0 cm de largo
  - 1.1 cm diámetro
- Material para impresión 3D PLA
- Hilo de nylon monofilamento
- Hoja cuadriculada en centímetros

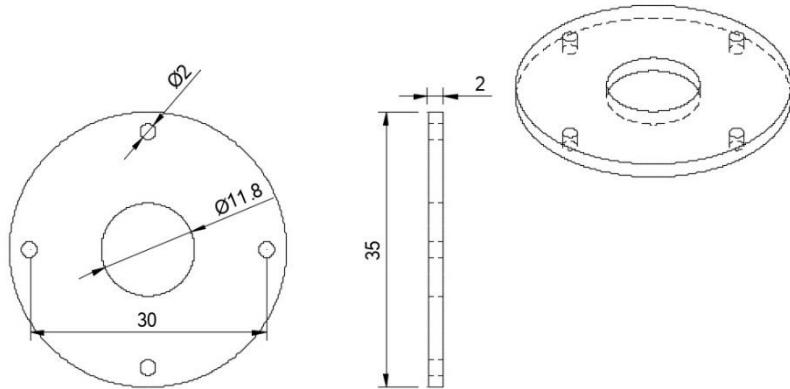
- Pesos calibrados
- 1.6L de agua en un recipiente
- Jeringa con capacidad de 50-60 mL
- Tornillos para madera / clavos
- Bloques y/o tablas de madera
- 2 rodamientos de diámetro interno de 8mm y diámetro exterior de 22mm
  - Código: 608 ZZ
- 2 trípode para cámara
- Balanza de precisión
- Goma de adhesión rápida
- Apuntador láser de baja potencia
- Nivel
- Escuadra

### **3.3.4 Experimento No. 1**

El procedimiento del experimento No. 1 que fue llevado a cabo se detalla a continuación:

1. Se calculó el volumen del cilindro de silicona usando los datos de diámetro y longitud.
2. Se calculó la densidad del cilindro de silicona utilizando los datos obtenidos de su respectiva masa y volumen.
3. Usando una tabla de madera y tornillos de máquina, se fijó 2cm de un cilindro de silicona, de manera que quedaran 25cm de cilindro de silicona libre. Se aseguró que, por el momento, el cilindro quedara apoyado para que no tenga deflexión.

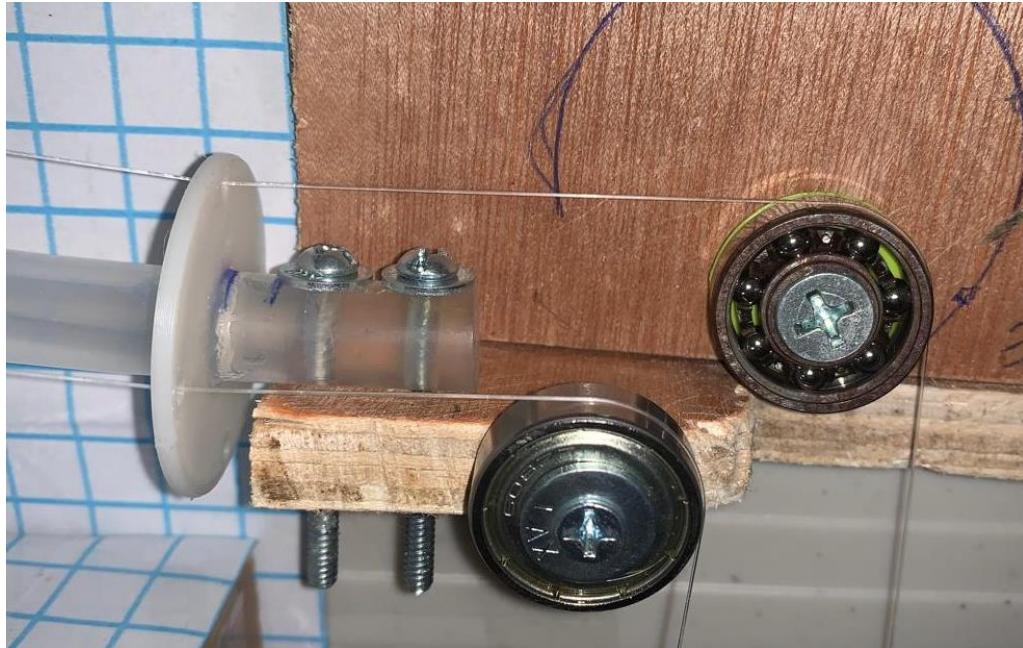
4. Se diseñaron 6 discos para ser utilizados como vértebras. Estos toman en cuenta el diámetro de la varilla y siguen el diseño visto en la **Figura 3-7**.



**Figura 3-7.** Diseño básico de las vértebras utilizadas para los experimentos presentados.

5. Con esta fijación, se tienen 25cm de cilindro libre. Se marcaron 0.5cm desde el extremo empotrado y desde el extremo libre, para que la longitud útil del cilindro sea de 24cm.
6. Teniendo la longitud útil del cilindro de silicona, 24.0 cm, se dividió esta entre cinco, lo cual dio 4.8 cm. Entonces, comenzando desde un extremo de la longitud útil de la varilla hacia el otro, se colocaron los 6 discos a lo largo del cilindro de silicona, espaciados cada 4.8cm. Para asegurar su fijación al cilindro, se colocó un poco de adhesivo en el punto de contacto entre cilindro y disco.
7. Se pasó hilo monofilamento de nylon a través de los agujeros (sólo los horizontales apuntando hacia afuera y los verticales apuntando hacia arriba) de los discos a lo largo del cilindro de silicona, asegurando de atar un nudo en el agujero del último disco, ubicado en la punta del extremo libre del cilindro de silicona. Se aseguró dejar hilo suelto en el otro extremo, donde está el empotramiento del cilindro de silicona.

8. Se colocaron los dos rodamientos de manera muy cuidadosa a la fijación del sistema, asegurando que cuando el hilo monofilamento de nylon reposara sobre el rodamiento, este quedara con un ángulo cercano o igual a cero con respecto al eje horizontal. Esto se hizo para ambos hilos de monofilamento de nylon que fueron pasados por los discos, como se puede notar en la **Figura 3-8**.

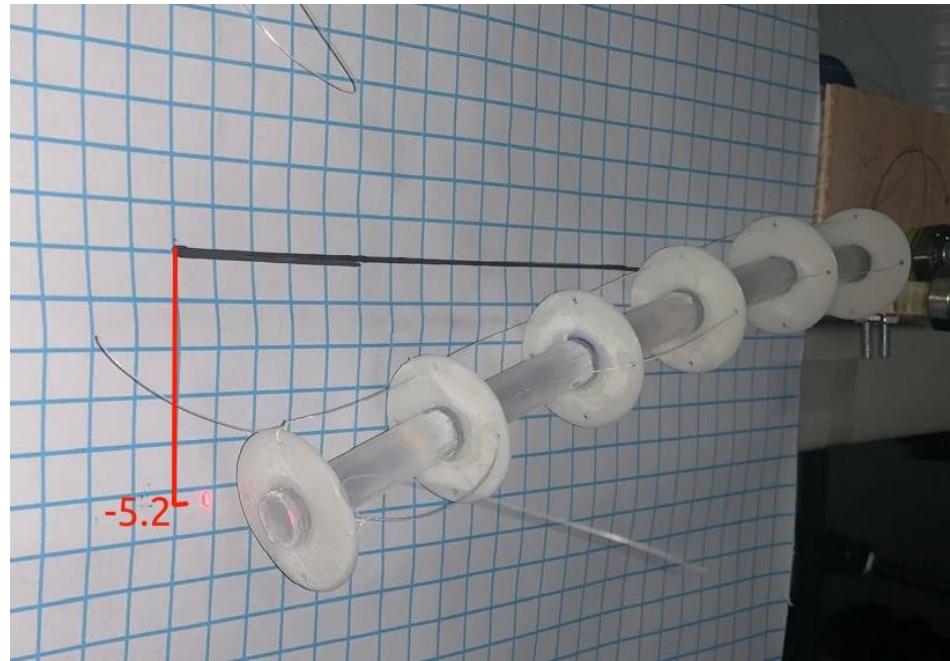


**Figura 3-8.** El arreglo de sujeción para la varilla mediante tornillos de máquina, al igual que la correcta alineación de los rodamientos para asegurar un ángulo cercano a cero con la horizontal para el hilo monofilamento de nylon.

9. En una superficie plana y vertical, se colocó la hoja cuadriculada.
10. Se colocó la fijación con el cilindro de silicona lo más cercano posible a la hoja cuadriculada, sin lograr contacto con esta. También asegurando que todo el cilindro estuviera dentro de la cuadrícula.
11. Se ajustó el trípode con la cámara para asegurar una buena toma de imágenes, de manera que se pueda observar todo el cilindro y la hoja cuadriculada.

12. Se ajustó el trípode con el láser de manera que fuera fácil reajustar su posición cuando se soltara el cilindro de silicona y que este se flexionara. También, se verificó que este estuviera nivelado y recto con respecto a la hoja cuadriculada.
13. Se soltó el apoyo que tenía el cilindro de silicona.
14. Cuando se estabilizó la deflexión de la varilla, se ajustó el trípode con el láser de manera que el láser apuntara directamente a la punta del cilindro de silicona, se tomó una imagen del sistema con la cámara asegurando que se capturara el sistema completo.
15. Ahora, se volvió a apoyar el cilindro de silicona de manera que no tuviera deflexión, se esperaron 5 minutos.
16. Se repitió el paso 13 un total de 5 veces, y se notó que los resultados fueron los mismos.
17. Se anotó la deflexión en Z (dirección vertical) de la punta de la varilla y esta fue utilizada para calibrar el módulo de elasticidad de la simulación.
18. Se volvió a apoyar el cilindro de silicona de manera que no tuviera deflexión.

Como se mencionó en el procedimiento, se repitieron las mediciones cinco veces, las cuales dieron resultados idénticos, lo que incrementa la confiabilidad en ellas. A continuación, en la **Figura 3-9**, se puede ver la imagen que muestra el sistema en deflexión por su propio peso, junto con el láser apuntando a la punta del cilindro de silicona, con la medición anotada en la hoja cuadriculada.



**Figura 3-9.** Medición realizada mediante el uso del láser para el experimento No. 1. Se nota que los tendones no están tensados y que el láser está en el centro de la punta del cilindro, con una medición de -5.2 centímetros.

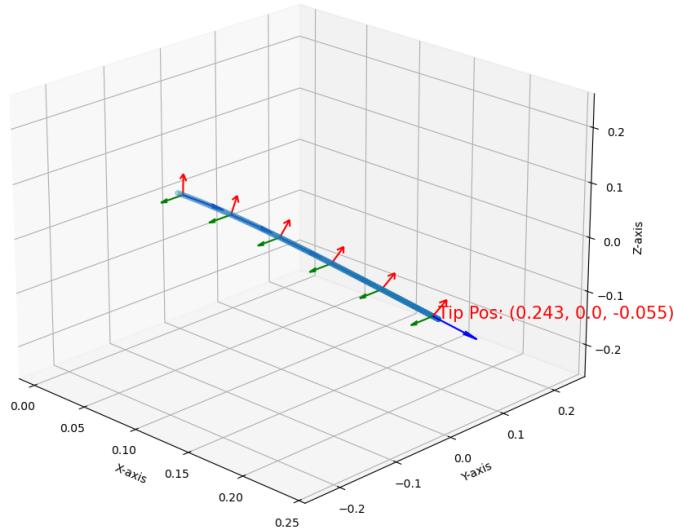
Siguiendo el procedimiento detallado en la sección **3.3.2 Calibración de la simulación**, se obtuvo el siguiente resultado:

Experimento No. 1		
Z medido (cm)	Z corregido (cm)	Resultado
		Mód. Elasticidad (Pa)
-5.2	-5.4	1.6598637E+07

**Tabla 3-1.** Tabla de resultado obtenido para la calibración de la simulación realizada en el experimento No. 1.

El resultado de la simulación se puede ver a continuación en la **Figura 3-10**:

Pose final para el experimento No. 1,  
 Deflexión bajo el propio peso del sistema,  
 $E = 16.5986 \text{ MPa}$ ,  $G = 7.2169 \text{ MPa}$



**Figura 3-10.** Pose final de la simulación después de que la varilla se flexionara bajo su propio peso y el de las vértebras solamente.

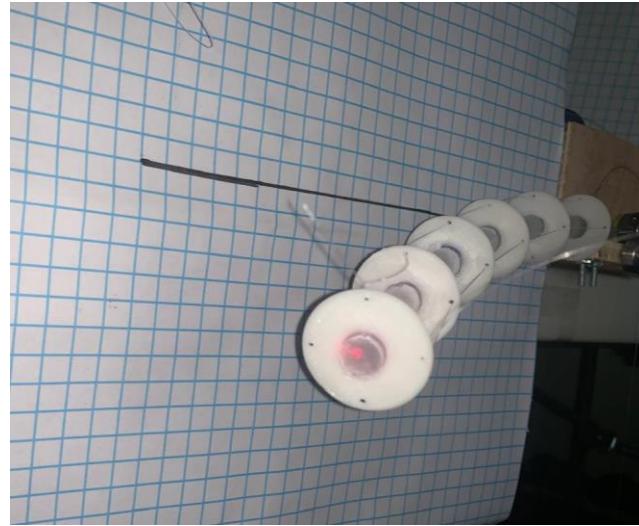
### 3.3.5 Experimento No. 2

El procedimiento del experimento No. 2 que fue llevado a cabo se detalla a continuación:

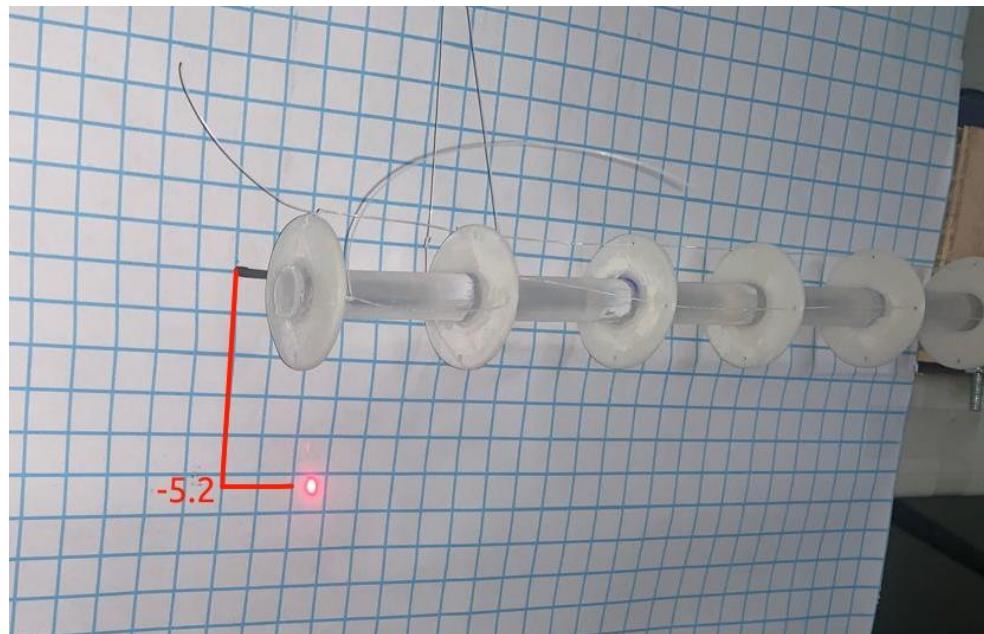
1. Se preparó el trípode con la cámara para que tuviera una buena toma del sistema.
2. Se preparó el trípode con el láser, asegurando que este estuviera nivelado y recto con respecto a la hoja cuadriculada.
3. Se ató un nudo, en el extremo del empotramiento con el hilo monofilamento suelto que corresponde al tendón en el plano horizontal del disco, a uno de los pesos calibrados.

4. El hilo monofilamento de nylon fue colocado encima de su rodamiento respectivo y el peso calibrado fue soltado poco a poco, hasta que la tensión en el hilo soportara el peso completo del peso calibrado.
5. Se ajustó el trípode con el láser de manera que el láser apuntara directamente a la punta del cilindro flexionado.
6. Se tomó una foto del sistema flexionado con el láser apuntando a la punta.
7. Se apoyó el peso calibrado y se apoyó el cilindro de silicona de manera que este no tuviera deflexión.
8. Se activó el láser nuevamente y se tomó una imagen del sistema, capturando dónde exactamente en la hoja cuadriculada marcó el láser.
9. Se midió la deflexión en Z (dirección vertical) que tuvo la punta del cilindro de silicona. Se repitieron los pasos 8-12 cinco veces, esperando 5 minutos entre ensayo. Se notó que los resultados eran los mismos. Este resultado entonces se utilizó para calibrar el módulo de rigidez al cortante de la simulación en PyElastica.

Como se mencionó en el procedimiento, se repitieron las mediciones cinco veces, las cuales dieron resultados idénticos, lo que incrementa la confiabilidad en ellas. A continuación, en la **Figura 3-11** se puede ver la imagen que muestra el sistema en deflexión por su propio peso y la acción del tendón horizontal, junto con el láser apuntando a la punta del cilindro de silicona y posteriormente se apoya el cilindro y se muestra con la medición anotada en la hoja cuadriculada en la **Figura 3-12**.



**Figura 3-11.** El láser está correctamente nivelado y está apuntando al centro de la punta del cilindro de silicona flexionado.



**Figura 3-12.** La medición después de apoyar la varilla y mantener el láser en el mismo punto que en la **Figura 3-11**.

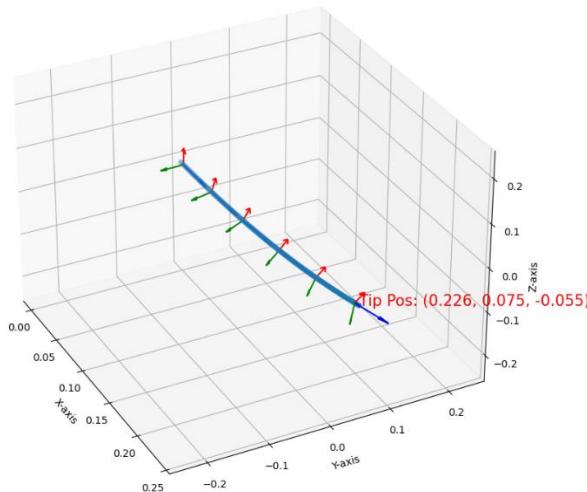
Siguiendo el procedimiento detallado en la sección **3.3.2 Calibración de la simulación**, se obtuvo el siguiente resultado:

Experimento No. 2		
Z medido (cm)	Z corregido (cm)	Resultado
		Mód. Rigidez Cortante (Pa)
-5.2	-5.4	7.2168800E+06

**Tabla 3-2.** Tabla para el resultado obtenido de la calibración de la simulación en el experimento No. 2.

El resultado de la simulación se puede ver a continuación en la **Figura 3-13**:

Posé final para el experimento No. 2,  
 Deflexión bajo el propio peso y Tendón horizontal tensionado con 2.0139917105 N  
 $E = 16.5986 \text{ MPa}$ ,  $G = 7.2169 \text{ MPa}$



**Figura 3-13.** Simulación de la varilla ante las cargas de su propio peso, el peso de las vértebras y una tensión de 2.0139917105 N en el tendón ubicado en la hilera apuntando en la dirección +Y.

### 3.3.6 Experimento No. 3

El procedimiento para este experimento se describe a continuación:

1. Utilizando el mismo sistema del experimento No. 2, se ataron dos pesos calibrados a los tendones por el lado de empotramiento, que tenían hilo de nylon suelto. Estas se mantuvieron apoyados por el momento.

2. Se colocó una hoja en una superficie horizontal y plana, perpendicular a la hoja milimétrica vertical, de manera que esta estuviera a una altura de alrededor de -3.0 cm con respecto al eje cero marcado en la hoja cuadriculada vertical.
3. Se ajustó y preparó el trípode con la cámara al igual que el trípode con el láser.
4. Al peso calibrado unido al tendón en el plano vertical, se le añadió 50 g.
5. Al peso calibrado unido al tendón en el plano horizontal, se le añadió 25 g.
6. Se soltaron suavemente los pesos calibrados para que la tensión no fuera aplicada tan bruscamente al tendón.
7. Se ajustó el trípode con el láser de manera que el láser apuntara directamente a la punta del cilindro de silicona flexionado.
8. Se tomó una foto del sistema con el láser activado, posteriormente desactivándolo.
9. Utilizando la escuadra, se colocó perpendicular a la superficie horizontal con la hoja cuadriculada, y se alineó con el centro del cilindro de silicona flexionado. Esto fue para medir el desplazamiento que tuvo la punta en el eje Y (dirección horizontal), cuya medición fue anotada.
10. Ahora, se apoyaron los dos pesos calibrados al igual que el cilindro de silicona.
11. Se activó el láser y se tomó una foto para capturar exactamente dónde en la cuadrícula marcó la posición de la punta el láser, esta medición funciona para el eje X (dirección axial) y para el eje Z (dirección vertical), ambas de las cuales se anotaron.
12. Los pasos 5-12 se repitieron un total de 10 veces, hasta tener 10 mediciones, 500g añadidos al tendón en el plano vertical y 250g añadidos al tendón en el plano horizontal.

13. Los resultados obtenidos en el paso 13 se compararon con las simulaciones realizadas en PyElastica, utilizando los mismos parámetros y condiciones que en los experimentos respectivos. Las simulaciones fueron realizadas utilizando el interfaz gráfico manual para realizar simulaciones de lazo abierto, que se puede ver a mayor detalle en el capítulo **4 Implementación en ROS2**. Estos resultados fueron analizados y se concluyó acerca de la validación del modelo de accionamiento mediante tendones.

Los resultados obtenidos para este experimento se pueden ver a continuación. Primero, se detallan los resultados de las mediciones realizadas en el experimento físico, con sus correcciones respectivas debido a los desfases:

	Masa (g)		(cm)	+0.3 (cm)	(cm)	-2.0 (cm)	(cm)	-0.2 (cm)
Ensayo	Tendón vertical	Tendón horizontal	X Medido	X Corregido	Y Medido	Y Corregido	Z Medido	Z Corregido
1	105.37	80.37	24.2	24.5	5.2	3.45	-1.3	-1.5
2	155.37	105.37	24	24.3	6.3	4.3	0.8	0.6
3	205.37	130.37	23.6	23.9	7	5	2.7	2.5
4	255.37	155.37	22.9	23.2	8	6	4.7	4.5
5	305.37	180.37	21.6	21.9	9.8	7.8	6.7	6.5
6	355.37	205.37	19.9	20.2	10.5	8.5	8.9	8.7
7	405.37	230.37	18.5	18.8	12	10	10.1	9.9
8	455.37	255.37	16.2	16.5	12	10	11.8	11.6
9	505.37	280.37	13.9	14.2	12.7	10.7	12.7	12.5
10	555.37	305.37	11.1	11.4	13.6	11.6	13.8	13.6

**Tabla 3-3.** Resultados de las mediciones realizadas, junto con sus correcciones, para los 10 ensayos del experimento No. 3.

Se nota que las mediciones fueron corregidas al revertir el desfase asociado a ellas. Luego, se detallan los resultados de las simulaciones y se comparan estos resultados con los resultados obtenidos del experimento físico, lo cual se puede observar en las columnas de diferencia en la **Tabla 3-4**:

Ensayo	(cm)					
	X simulado	Y simulado	Z simulado	Diferencia X	Diferencia Y	Diferencia Z
1	24.6	3.1	-1.7	0.1	-0.35	-0.2
2	24.4	4.2	0.2	0.1	-0.1	-0.4
3	23.9	5.2	2.2	0	0.2	-0.3
4	23.1	6.3	4.2	-0.1	0.3	-0.3
5	22	7.3	6.1	0.1	-0.5	-0.4
6	20.5	8.2	8	0.3	-0.3	-0.7
7	18.7	9	9.8	-0.1	-1	-0.1
8	16.5	9.7	11.3	0	-0.3	-0.3
9	14.1	10.2	12.6	-0.1	-0.5	0.1
10	11.4	10.4	13.5	0	-1.2	-0.1

**Tabla 3-4.** Tabla que muestra los resultados de las diferencias entre las posiciones XYZ simuladas y las medidas en el experimento No. 3.

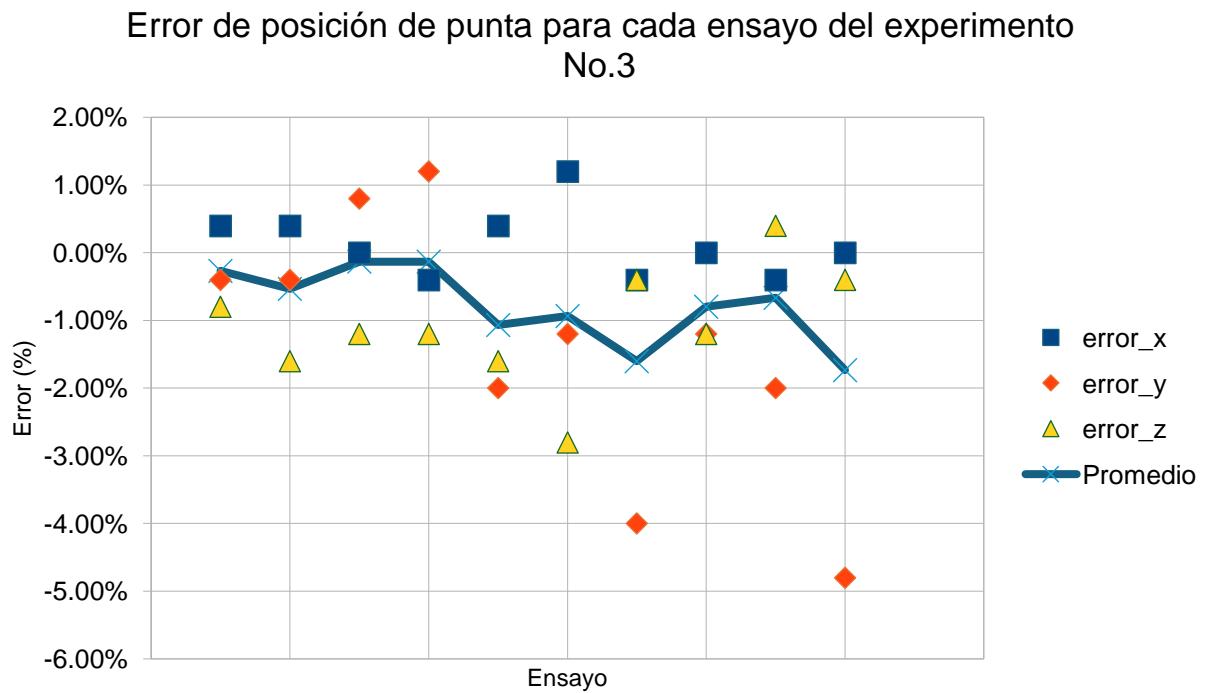
Para poder entender mejor los resultados, se calculó una tercera tabla que muestra el porcentaje de error de posición obtenido para cada uno de los ensayos, la cual se puede observar a continuación:

Ensayo	Diferencia / Longitud de robot * 100%			Promedio error
	Error X	Error Y	Error Z	
1	0.40%	-1.40%	-0.80%	-0.60%
2	0.40%	-0.40%	-1.60%	-0.53%
3	0.00%	0.80%	-1.20%	-0.13%
4	-0.40%	1.20%	-1.20%	-0.13%
5	0.40%	-2.00%	-1.60%	-1.07%
6	1.20%	-1.20%	-2.80%	-0.93%
7	-0.40%	-4.00%	-0.40%	-1.60%
8	0.00%	-1.20%	-1.20%	-0.80%
9	-0.40%	-2.00%	0.40%	-0.67%
10	0.00%	-4.80%	-0.40%	-1.73%

**Tabla 3-5.** Resultados de error relativo entre los datos de posición simulados y los medidos en el experimento No. 3.

Se nota que el cálculo del error se hace al considerar la longitud total de la varilla del robot. Como se detalla en la parte superior de la **Tabla 3-5**, el cálculo de error se hace al dividir la diferencia medida entre la longitud total del robot, la cual es de 25cm.

Para visualizar los resultados, se muestra la siguiente gráfica que ilustra los valores de error calculados en la **Tabla 3-5**:



**Figura 3-14.** Gráfica de errores para el experimento No. 3, donde se observa que el error máximo fue en el eje Y para el último ensayo. Se nota también que el error promedio de cada uno de los ensayos se mantiene menor a una magnitud de 2%.

En base a los resultados obtenidos, se concluye que el margen de error se encuentra en un promedio aceptable de por debajo de  $\pm 2\%$ . Esto significa que el

modelo de fuerzas y momentos para el accionamiento mediante tendones está validado y se demuestra en los resultados obtenidos en las simulaciones de PyElastica.

Las imágenes de las mediciones realizadas, junto con las respectivas simulaciones se pueden observar en la sección **6.1 Mediciones y simulaciones del experimento No. 3** del anexo.

## 4 Implementación en ROS2

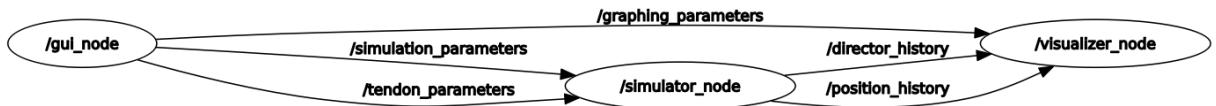
En este capítulo, se detallará la arquitectura de la red de comunicación en ROS2, además del funcionamiento de cada uno de los nodos y cómo están integrados en la red. Es importante destacar que esta red desarrollada para este trabajo sirve como ejemplo para trabajos futuros que deseen aprovechar de la modularidad del entorno en ROS2 para añadir nuevos nodos con diferentes funcionalidades a la red.

En este capítulo se van a presentar dos modos de simulación: simulación de lazo abierto donde manualmente se añaden todos los parámetros, incluyendo las tensiones de los tendones, utilizando el módulo de forzado TendonForces desarrollado en la sección **3.2.3.1 Forzado manual y personalizado con TendonForces**. También simulación de lazo cerrado, en la cual se llevará a cabo el control de la posición de la punta del robot, y esta utilizará el módulo de forzado QuadTendonForces, desarrollado en la sección **3.2.3.2 Forzado retroalimentado con un controlador usando QuadTendonForces**. Representaciones visuales de estos dos tipos de modos de simulación se pueden ver en la **Figura 3-4** y **Figura 3-5**, respectivamente, de la sección **3.2.3 Estructura como módulo de forzado externo para el accionamiento por tendones**.

De manera general, el funcionamiento del sistema de lazo abierto, o manual, es el siguiente:

1. Primero se encienden los nodos de simulación, visualización y de interfaz gráfica de usuario manual.
2. En la interfaz gráfica de usuario, se especifican todos los parámetros y condiciones que se desean para la simulación, incluyendo todos los parámetros y tensiones para los tendones que se quieran añadir por el usuario.
3. Se envían los datos, que se reciben en los nodos de simulación y visualización a la misma vez.
4. El nodo de simulación inicia la simulación y cuando termina, envía los datos de historial de posición y orientación de los nodos de la varilla al nodo de visualización.
5. El nodo de visualización se encarga de crear un video que muestra la evolución del sistema a lo largo del tiempo.

El entorno completo en ROS2 para el sistema de lazo abierto se puede visualizar en su gráfica RQT:



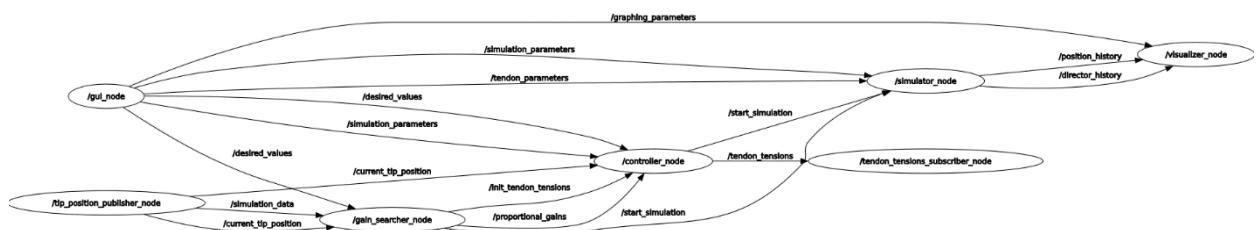
**Figura 4-4-1.** Gráfico RQT que muestra los nodos (óvalos) y los tópicos que los conectan para el sistema de lazo abierto, o manual.

De manera general, el funcionamiento del sistema de lazo cerrado, o de control, es el siguiente:

1. Primero se encienden los nodos de simulación, visualización, controlador, buscador de ganancias y de interfaz gráfica de usuario para control.
2. En la interfaz gráfica de usuario, se especifican todos los parámetros y condiciones que se desean para la simulación, incluyendo aquellos relacionados al control del robot. También, se especifica si se desea o no realizar un ajuste de ganancias proporcionales.

3. Se envían los datos, los que se reciben en diferentes nodos a la misma vez.
4. El nodo de simulación comienza la simulación, y simultáneamente le envía los datos de posición al controlador, que le responde con tensiones en los tendones del robot, intentando llegar a la posición deseada, con las ganancias que se le dieron.
5. Una vez que termina el tiempo de simulación, el nodo de simulación publica la historia de posición de todos los nodos en el robot hacia el nodo de visualización, que se encarga de crear un video que muestra la evolución del sistema a lo largo del tiempo.
6. En el caso de que se escoja la opción de ajuste de ganancias, el nodo de ajuste de ganancias publicará nuevas ganancias, en base a los resultados obtenidos, hacia el nodo controlador, y se comienza una nueva simulación en el nodo de simulación. Esto hasta que se cumplan los criterios de tiempo y convergencia.

El entorno completo en ROS2 para el sistema de lazo cerrado se puede visualizar en su gráfica RQT:



**Figura 4-4-2.** Gráfico RQT que muestra los nodos (óvalos) y los tópicos que los conectan para el sistema de lazo cerrado, o de control.

## 4.1 Nodo de interfaz gráfica (GUI)

Este nodo se encarga de proveer una interfaz entre el usuario y el entorno de simulación interconectado en la red de comunicación en ROS2. La importancia de tener este nodo de interfaz gráfica es que permite que el usuario interactúe directamente con el entorno de simulación sin tener que adentrar en el código que yace detrás del entorno de simulación, puesto que, para un usuario que no está familiar con el código, puede resultar complicado. Además, todos los nodos dentro de la red de comunicación en ROS2 para este entorno de simulación responden a señales enviadas por el nodo interfaz gráfica, de manera directa e indirecta.

### 4.1.1 *Integración en el sistema*

Para esta sección de integración en la red de comunicación en ROS2, se destaca que para ambas interfaces gráficas (manual y de control), la integración es la misma. La única diferencia en cuanto a la integración de las interfaces son algunos publicadores que se omiten en la interfaz gráfica para el caso manual.

Para comenzar, se importa la librería de tkinter, proveniente de la librería estándar de Python (Python Software Foundation, n.d.), la librería CustomTkinter (Schimansky, n.d.) y un módulo de PIL (Clark, n.d.) llamado Image.

Este nodo requiere que esté andando el nodo GUI que publicará los parámetros y valores que desee introducir el usuario, al igual que la ventana de interfaz gráfica como tal. Esto requiere que se tenga que realizar un arreglo para que ambos procesos puedan funcionar sin problemas. De manera general, se crea una clase que contiene la aplicación que funcionará como la ventana de GUI. Esta tiene todas las funcionalidades, botones, campos para llenar, imágenes, etc., que tiene la ventana GUI cuando se ejecuta. También se crea la clase que contiene la información para el nodo como tal, que en este caso solamente crea los publicadores.

Entonces, para unir la ventana de GUI con el nodo, primero se crea una instancia de objeto del nodo, y esta instancia se usa como argumento para crear el objeto de la clase GUI. Luego se mantiene en bucle el objeto de GUI y se usa la función de ROS2 “spin()” para el nodo. Ahora, cuando se desee publicar alguna información desde la interfaz gráfica, sólo se debe acceder a los publicadores que el objeto del nodo tiene como atributo, los cuales son accesibles por el objeto de GUI ya que el objeto de nodo fue pasado como argumento.

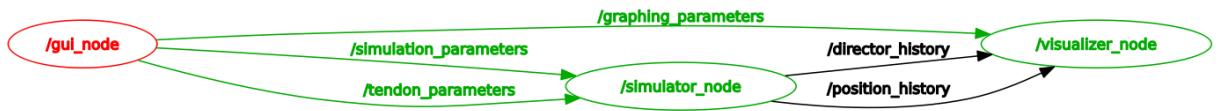
Los publicadores que son utilizados para este nodo son los siguientes, en el caso del GUI manual:

- “sim\_parameters\_publisher”: se encarga de enviar los parámetros de la simulación al tópico ‘/simulation\_parameters’.
- “tendon\_parameters\_publisher”: se encarga de enviar los parámetros de los tendones al tópico ‘/tendon\_parameters’.
- “graphing\_parameters\_publisher”: se encarga de enviar los parámetros de visualización al tópico ‘/graphing\_parameters’.

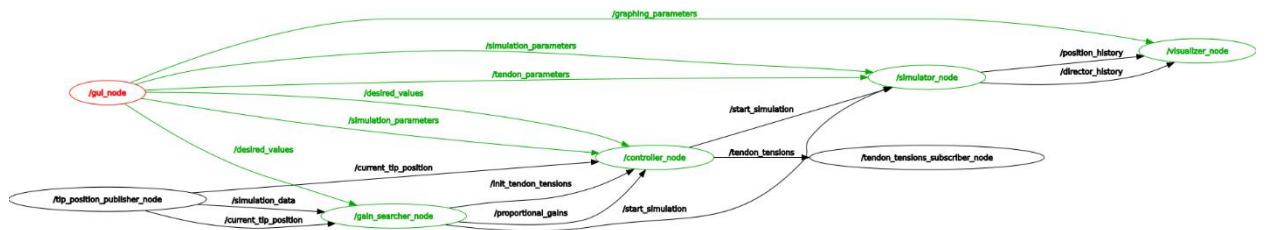
Los publicadores que son utilizados para el caso del GUI de control son los mismos, solamente que se añaden los siguientes:

- “desired\_values\_publisher”: se encarga de enviar los valores de posición deseados, las ganancias del controlador y el tiempo de llegada deseado a través del tópico ‘/desired\_values’.
- “initialize\_tendon\_tensions\_publisher”: se encarga de enviar una señal al tópico ‘/tendon\_tensions\_init’.

La interconexión del nodo de GUI con el resto de la red de comunicación en ROS2 se puede ver en las figuras continuación, para ambos casos (manual y de control):



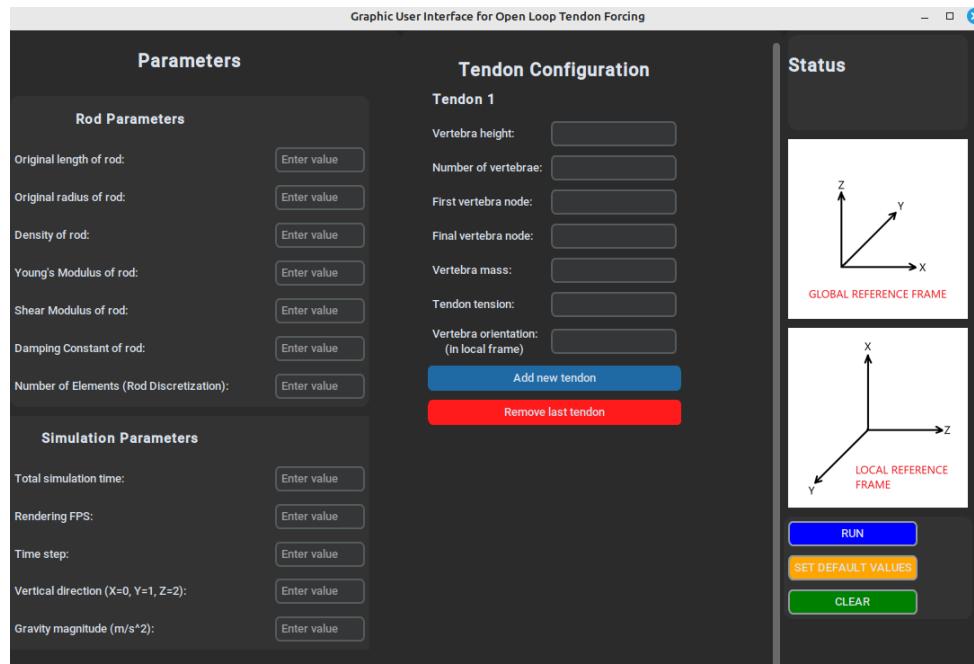
**Figura 4-3.** Gráfico RQT del sistema manual. El nodo de interfaz gráfico está señalado en rojo y los tópicos a los que publica están en verde.



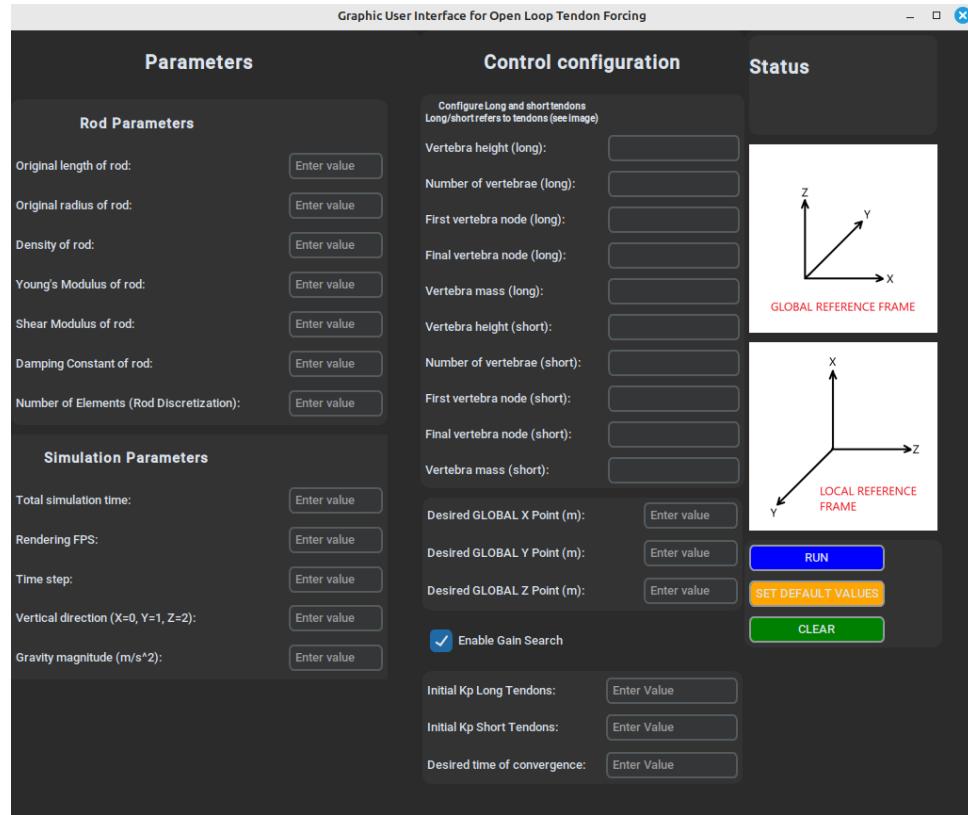
**Figura 4-4.** Gráfico RQT del sistema de control. El nodo de interfaz gráfico está señalado en rojo y los tópicos a los que publica están en verde.

#### 4.1.2 Funcionamiento

Ambas interfaces gráficas se presentan a continuación:



**Figura 4-5.** Interfaz gráfica de usuario (GUI) para el caso manual.



**Figura 4-6.** Interfaz gráfica de usuario (GUI) para el caso de control.

El funcionamiento de las dos interfaces gráficas de usuario es relativamente similar; solamente hay cambios de algunos botones o campos. Primeramente, la ventana de los GUI crea tres secciones con las cuales puede interactuar el usuario. Las secciones se dividen en columnas de la ventana creada: la primera columna izquierda es la sección de parámetros para ambos GUI, luego la segunda sección es la sección de tendones para el GUI manual y la sección de control para el GUI de control, finalmente la tercera sección es la sección de información para ambos GUI. Dependiendo de cuál interfaz decida utilizar el usuario, se presentarán diferentes funcionalidades a la interfaz y al entorno de simulación en sí.

Comenzando con la primera sección, la sección de parámetros, se tienen dos subsecciones dentro de esta sección. Estas subsecciones son iguales para ambos GUI. Las secciones y las variables que le piden llenar al usuario son las siguientes:

- Subsección de parámetros de varilla:
  - o Longitud original de varilla (m)
  - o Radio original de la varilla (m)
  - o Densidad de la varilla ( $\text{kg/m}^3$ )
  - o Módulo de elasticidad de la varilla (Pa)
  - o Módulo de rigidez al cortante de la varilla (Pa)
  - o Constante de amortiguamiento de la varilla
  - o Cantidad de elementos (discretización de la varilla)
- Subsección de parámetros de simulación:
  - o Tiempo total de simulación (s)
  - o Imágenes por segundo para el renderizado (FPS)
  - o Paso temporal (s) (time step para la discretización temporal)
  - o Dirección vertical (X=0, Y=1, Z=2) (La gravedad se aplica en esta dirección)
  - o Magnitud de gravedad ( $\text{m/s}^2$ )

Ahora, en el caso del GUI manual, la segunda sección es la sección de tendones. Esta sección se encarga de recibir los parámetros para cada hilera individual de vértebras y tendones que el usuario quiera colocarle a la varilla, para lograr una simulación a lazo abierto. Esta sección cuenta con dos botones que ayudan a su funcionalidad. Uno de los botones, “Add new tendon”, permite al usuario a añadir un nuevo tendon al sistema y llenar sus parámetros. Por default, solamente hay un tendon que el usuario debe llenar de parámetros, entonces el botón para añadir nuevos tendones resulta útil para simulaciones con múltiples tendones. El otro botón, “Remove last tendon” permite al usuario remover el último de los tendones que se añadió al sistema. Esto permite remover tendones que se añadieron por error, o incluso realizar una simulación sin ningún tendon configurado en el robot.

En el caso del GUI de control, la segunda sección es la sección de control. Esta sección se encarga de configurar los parámetros de los tendones largos y cortos que serán utilizados para el arreglo de “QuadTendonForces” en la subsección de tendones. Luego, se tiene la subsección de posición deseada, donde el usuario puede llenar los campos para la posición en XYZ en el espacio a la cual desea que llegue la punta del robot al realizar el control. Después, está la sección de ajuste, en la que lo primero que se tiene es una opción para realizar un ajuste de ganancias. Si esta opción tiene no está activada, entonces aparecen solamente dos campos a llenar: las ganancias que serán utilizadas para los tendones largos y los cortos, respectivamente. Si la opción está activada, entonces aparecen tres campos a llenar por el usuario: el tiempo deseado de llegada a la posición especificada, al igual que las ganancias iniciales de los tendones largos y cortos, respectivamente.

Luego, para ambas interfaces gráficas, se tiene la misma tercera sección, la cual es la sección de información. Lo primero que se tiene en esta sección es una subsección donde se muestra un mensaje de estatus del simulador, para poder saber cuáles errores se han presentado o si está realizando computaciones en un momento dado. Seguido, se tienen dos imágenes que muestran las orientaciones default de los marcos de referencia globales y locales, lo cual es útil para el usuario a la hora de introducir los datos en la interfaz gráfica, puesto que algunos datos se especifican en diferentes marcos de referencia. Seguido, hay una subsección que contiene tres botones con funciones. El primer botón, “RUN”, es el botón que envía todos los parámetros y valores a la red de comunicación en ROS2 para comenzar con los procesos de los nodos. En el caso de que haya información mal escrita o que haga falta información en los campos a llenar, apretar el botón “RUN” ocasionará que se presente un mensaje de error en la subsección de estatus. El segundo botón, “SET DEFAULT VALUES”, se encarga de configurar todos los datos en la interfaz gráfica de manera automática, utilizando valores preestablecidos. Esto es de utilidad para ver exactamente cómo se deben llenar

todos los campos, al igual que funciona de ejemplo para realizar simulaciones iniciales. De último, el botón “CLEAR” se encarga de borrar toda la información que haya sido llenada en los campos de la interfaz gráfica, para facilitar el uso por el usuario y que no tenga que borrar los campos uno por uno.

## **4.2 Nodo de simulación**

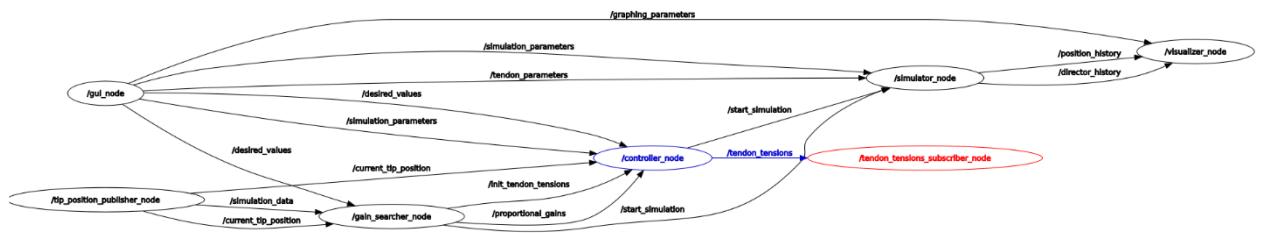
Este nodo se encarga de realizar las simulaciones utilizando el software de simulación PyElastica. Es uno de los más complejos en cuanto a la integración de los sistemas en el entorno de ROS2, por la manera que funciona el software PyElastica. Naturalmente, este nodo requiere el uso del paquete de simulación PyElastica (Gazzola Lab, 2024).

### ***4.2.1 Integración en el sistema***

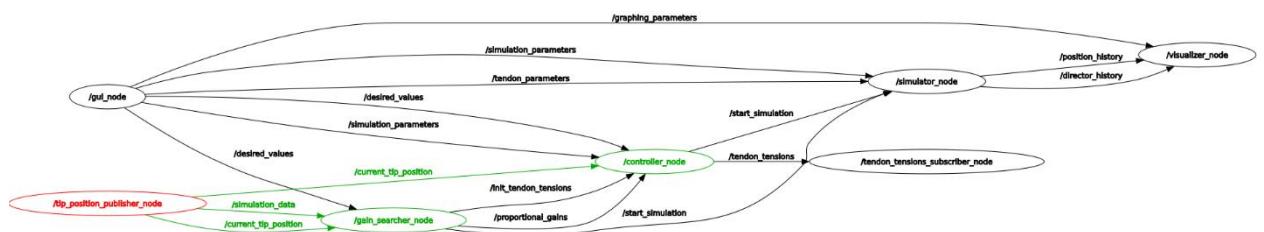
Como se mencionó en la sección **3.2.1 Funcionamiento de PyElastica**, el software utiliza módulos que importa y utiliza directamente. El problema, cuando se quiere hacer el control del sistema mediante un nodo que está externo al módulo de forzado, es que no se puede afectar el software de simulación y el módulo de forzado. Esto es porque PyElastica, a la hora de aplicar el forzado externo, crea una instancia del objeto que vendría siendo la clase de forzado en cuestión. En otras palabras, PyElastica crea un objeto que no puede ser afectado por un nodo externo de ROS2. Esto es un problema, puesto que se tenía previsto tener un nodo de control que se encargara solamente de modificar las tensiones en el sistema para lograr que el robot se estabilice en un punto deseado en el espacio.

Además de esto, la simulación del sistema por PyElastica se da como un solo proceso, lo que significa que cualquiera operación o publicación de datos que se quiera hacer, tendrá que esperar hasta que termine la simulación. Esto es un problema también, puesto que para controlar el robot es necesario publicar datos de posición hacia el nodo controlador que los procesará.

Para resolver estos problemas, lo que se hizo fue crear dos nodos en el mismo documento de código que el nodo de simulación. Uno de los nodos, “tip\_position\_publisher\_node” se enfocaría en publicar los datos de posición actuales de la punta del robot continuo al igual que el tiempo de simulación actual, los cuales se envían al nodo del controlador. El otro nodo, “tendon\_tensions\_subscriber\_node” se encargaría de recibir mediante una suscripción, las tensiones de los tendones que serían enviadas por el nodo controlador que deberían directamente afectar la simulación. Las interacciones que tienen estos dos nodos adicionales se pueden visualizar en los siguientes gráficos RQT con los nodos y tópicos resaltados:



**Figura 4-7.** Gráfico RQT para el sistema de control, donde se señala “tendon\_tensions\_subscriber\_node” en rojo y el tópico al cual se subscribe en azul.



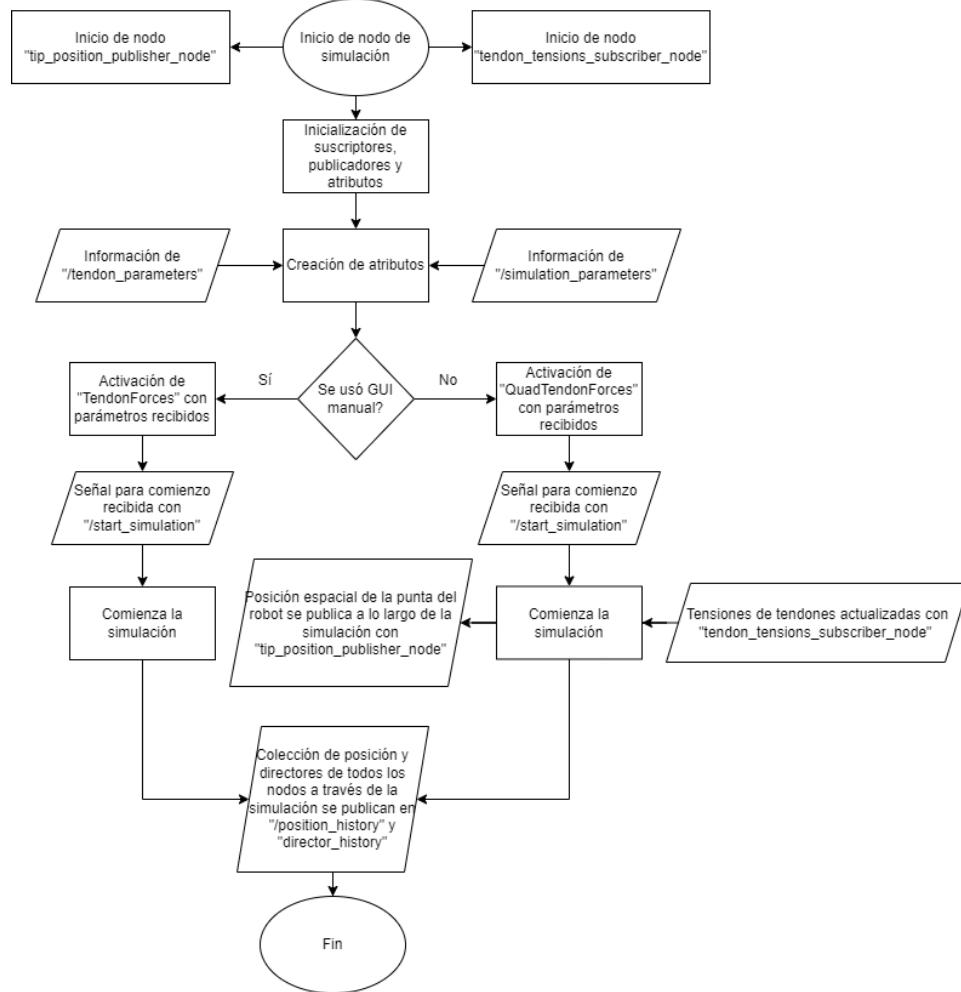
**Figura 4-8.** Gráfico RQT para el sistema de control, donde se señala “tip\_position\_publisher\_node” en rojo y los tópicos a los cuales publica en verde.

Estos dos nodos, para ser utilizados, se deben crear como objetos y ser utilizados en el mismo simulador para que sean útiles. Para resolver el problema de que el simulador ocurre en un solo proceso, estos dos nodos fueron llamados en la inicialización de la función de recopilación de datos “MyCallback” que se utiliza en el simulador de PyElastica, mencionado en la sección **3.2.1 Funcionamiento de PyElastica**. Entonces, al tener las dos instancias de los nodos inicializadas, se pueden utilizar sus métodos de publicación y suscripción para enviar y recibir datos durante la simulación, cada vez que se llame la función de “MyCallback”.

Luego, para resolver el problema de no poder acceder al módulo de forzado que está siendo utilizado por PyElastica, para de esta manera afectar las tensiones de los tendones y lograr controlar el robot, se tuvo que adentrar en el funcionamiento del software. Como se mencionó previamente en esta sección, PyElastica crea una instancia del objeto definido por la clase de forzado externo que se desea utilizar, y usa este objeto para aplicarle las fuerzas y torques a la varilla a lo largo de la simulación. Este objeto se guarda en la memoria y puede ser accedido si se conoce su ubicación. Entonces, debido a que el nodo de “tendon\_tensions\_subscriber\_node” se instancia dentro de la clase de “MyCallback” en el simulador, es necesario importar como argumento la instancia del nodo simulador dentro del “MyCallback”, para que pueda ser accedido en los métodos de “MyCallback”. Entonces, dentro de esta instancia del simulador, se accede a la varilla, luego a las fuerzas externas y torques de esta, se escoge el objeto de forzado externo en cuestión (en este caso “QuadTendonForces”), y a este objeto se le ejecuta un método que fue programado previamente, como se vio en la sección **3.2.3 Estructura como módulo de forzado externo para el accionamiento por tendones**, tomando como argumento los datos de tensiones que fueron recibidos por el nodo “tendon\_tensions\_subscriber\_node”. Esto entonces hace que se actualicen las tensiones con aquellas enviadas por el nodo de control, durante la simulación que hace PyElastica.

### 4.2.2 Funcionamiento

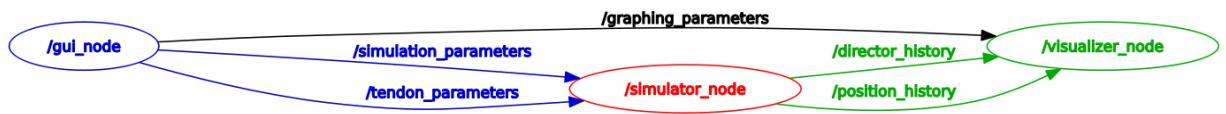
Se puede apreciar el funcionamiento de este nodo de simulación con el diagrama a continuación, seguido de una descripción de su funcionamiento:



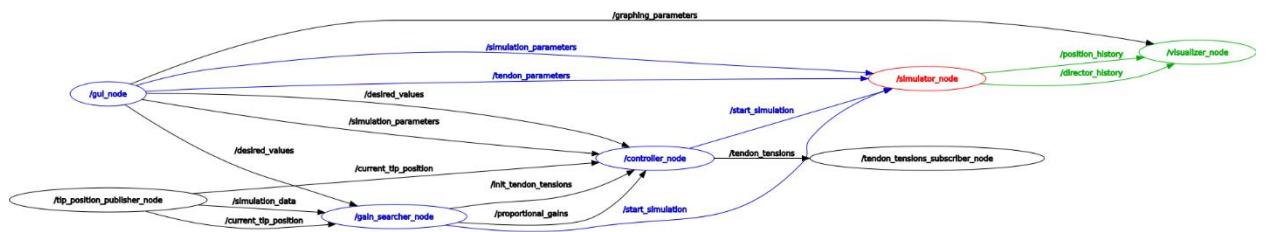
**Figura 4-9.** Diagrama de flujo para el funcionamiento del nodo de simulación.

El funcionamiento del nodo de simulación es relativamente simple, aparte del problema de integración visto en la sección previa, **4.2.1 Integración en el sistema**. El nodo de simulación le responde a la interfaz gráfica de usuario, recibiendo los parámetros introducidos por el usuario mediante el tópico '/simulation\_parameters'. Estos se reciben y se utilizan para preparar el simulador. También, se reciben los parámetros de los tendones mediante el tópico

'/tendon\_parameters'. Estos parámetros se utilizan para preparar inicialización del objeto de forzado mediante tendones. Cabe destacar que el tamaño del mensaje enviado por '/tendon\_parameters' determina si se utilizará el forzado externo de "TendonForces" o "QuadTendonForces" (ambos vistos en la sección 3.2.3 **Estructura como módulo de forzado externo para el accionamiento por tendones**), esto es para el caso que se utilice la interfaz gráfica para la simulación manual del robot o aquella controlada por el nodo de control. Las interconexiones para el nodo de simulación en el sistema manual y el de control se pueden ver a continuación:



**Figura 4-10.** Gráfico RQT para el sistema manual donde se señala al nodo de simulación en rojo, los tópicos a los cuales se suscribe en azul y los tópicos a los cuales publica en verde.



**Figura 4-11.** Gráfico RQT para el sistema de control donde se señala al nodo de simulación en rojo, los tópicos a los cuales se suscribe en azul y los tópicos a los cuales publica en verde.

En cualquiera de los casos que se haya escogido (manual o controlado), el nodo del simulador va a recibir un mensaje a través del tópico '/start\_simulation', lo que ejecutará un método que comienza la simulación con los datos que se hayan recibido hasta ese momento. La razón por la cual se usa esta metodología es para poder realizar múltiples simulaciones consecutivas, para el caso de ajuste de las

mejores ganancias. Además, el proceso de la simulación como tal de PyElastica se debe ejecutar como un método asincrónico dentro de la instancia del objeto del nodo simulador, de manera que a la misma vez que la simulación esté andando, los nodos de “tip\_position\_publisher\_node” y también “tendon\_tensions\_subscriber\_node” puedan estar andando y enviando o recibiendo datos. Esto es importante porque estos dos nodos están inicializados en la misma terminal de Python, y como la simulación realizada por PyElastica no permite procesos simultáneos, esta debe ser ejecutada asincrónicamente. Esta ejecución asincrónica se hace mediante la librería de asyncio, que proviene de la librería estándar de Python (Python Software Foundation, n.d.).

Al finalizar la simulación, y todos los datos de posición de los nodos y los directores locales de los nodos a través del tiempo hayan sido recolectados mediante el “MyCallback” del simulador, se preparan los datos y se envían hacia el nodo de visualización mediante los tópicos ‘/position\_history’ y ‘/director\_history’. Estos datos están listos para ser recibidos y arreglados por el nodo visualizador para crear el video de la evolución del sistema a través del tiempo.

Debido a que hay varios nodos que deben estar andando a la misma vez, se utiliza la funcionalidad de ROS2 conocida como MultiThreadExecutor, donde se añade el nodo simulador y el suscriptor de las tensiones, al igual que el bucle asincrónico de la simulación en PyElastica y se ejecuta.

### **4.3 Nodo de visualización**

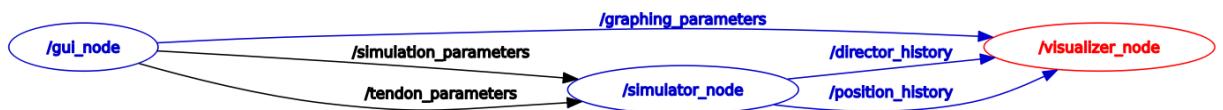
El nodo de visualización es de los menos complejos en la red de comunicación en ROS2 puesto que su función es relativamente simple. De manera general, lo que hace es recibir la colección de posiciones a lo largo del tiempo de todos los nodos de la varilla del sistema, y los plasma en un video que puede ser guardado y reproducido por el usuario.

### 4.3.1 Integración en el sistema

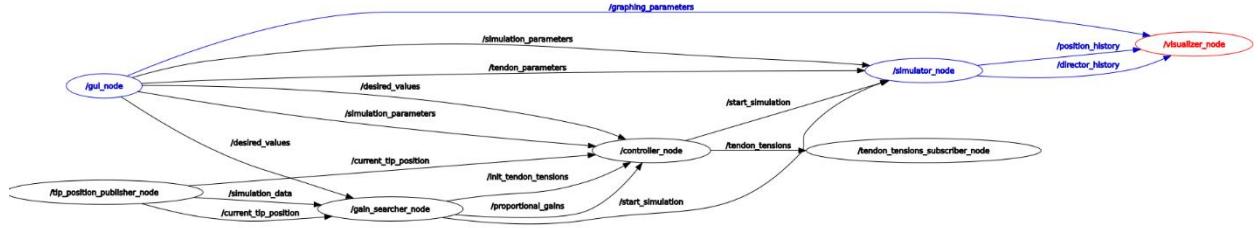
La integración del nodo de visualización en la red de comunicación de ROS2 es simple. Este nodo solamente recibe información a través de tres tópicos: '/graphing\_parameters' que contiene los parámetros deseados por el usuario en cuanto a la visualización del sistema en el video, '/position\_history' que contiene toda la data de posición de los nodos de la varilla a lo largo del tiempo y '/director\_history' que contiene toda la data de los directores locales de los nodos de la varilla a lo largo del tiempo. Este primer tópico envía información publicada por el nodo de interfaz gráfica de usuario, y el segundo y tercer tópico envía información publicada por el nodo de simulación. Cabe destacar que las conexiones que tiene en la red de comunicación permanecen iguales si se utiliza la interfaz gráfica de usuario manual y aquella para controlar el robot, por lo que este nodo no se ve afectado si se escoge cualquiera de las dos interfaces gráficas.

Para hacer funcionar este nodo, simplemente se crea un objeto de nodo usando la clase que se codificó, y este nodo se enciende y permanece “escuchando” a los tópicos a los cuales está suscrito, mediante la función de ROS2 conocida como “spin()”.

La interconexión que tiene este nodo con el resto de la red se puede ver a continuación en las siguientes figuras:



**Figura 4-12.** Gráfico RQT para el sistema manual, donde se señala el nodo de visualización en rojo y los tópicos a los cuales se suscribe en azul.



**Figura 4-13.** Gráfico RQT para el sistema de control, donde se señala el nodo de visualización en rojo y los tópicos a los cuales se suscribe en azul.

#### 4.3.2 Funcionamiento

El nodo de visualización comienza por importar módulos de matplotlib (The Matplotlib development team, n.d.), moviepy (Zulko, n.d.) y numpy (NumPy Developers, n.d.), los cuales serán útiles para la creación del video de simulación.

Al inicializar el nodo, solamente se crean los tres suscriptores cuyos tópicos se mencionaron previamente, “graphing\_parameters\_subscriber”, “position\_history\_subscriber” y “director\_history\_subscriber”. El método de callback que se le asigna a “graphing\_parameters\_subscriber” es el método “graphing\_parameters\_callback”. Este método se encarga de recibir los parámetros definidos en la interfaz gráfica de usuario para la visualización, y los utiliza para definir los atributos necesarios para el correcto funcionamiento del nodo visualizador. Los atributos que se definen son los siguientes:

- “base\_length”: la longitud de la varilla
- “array\_size”: la longitud del arreglo para X, Y, Z (es el mismo valor que la cantidad de nodos en el sistema más uno)
- “final\_time”: el tiempo final de la simulación
- “rendering\_fps”: la cantidad de imágenes por segundo que tendrá el video final
- “zmin”: el valor mínimo del eje Z en el gráfico del video
- “zmax”: el valor máximo del eje Z en el gráfico del video

- “ymin”: el valor mínimo del eje Y en el gráfico del video
- “ymax”: el valor máximo del eje Y en el gráfico del video
- “num\_frames”: valor calculado de cantidad total de imágenes en el video
- “num\_arrays”: valor calculado de cantidad total de arreglos de posición (X, Y, Z).

Lógicamente, “graphing\_parameters\_callback” se activa primero, puesto que la información recibida del interfaz gráfico de usuario es la primera información enviada por el usuario. Después que termina de realizar la simulación, el nodo de simulación envía la colección de todas las posiciones a lo largo del tiempo de todos los nodos, y se activa “position\_history\_callback”. Este método se ocupa de preparar la información recibida para que sea plasmada en un video. Es importante preparar la información, puesto que los tópicos de ROS2 no permiten enviar los arreglos de posición tal como se obtienen del nodo de simulación, sino que deben ser “aplastados” antes de ser enviados. Esto significa concatenar los arreglos de X, Y y Z en un solo arreglo y enviarlo a través del tópico, puesto que el tipo de datos que maneja ROS2, FloatMultiArray, no puede manejar arreglos multidimensionales. Entonces, al recibir la información “aplastada” en el “position\_history\_callback”, se debe arreglar de manera que vuelva a tener la forma ordenada que tenía antes, esto se logra con la función de numpy conocida como “reshape()”.

De manera similar, al recibir la data de los directores locales de los nodos en la varilla mediante su tópico respectivo, se activa “director\_history\_callback”, cuya función es solamente de recibir los datos y convertir el arreglo “aplastado” en la forma original que debería tener, mediante la función de numpy “reshape()”. Este nuevo arreglo se guarda como atributo y se utiliza para crear los marcos de referencia locales para las vértebras en el video final.

Al tener la data ordenada, se puede proceder con la creación de las imágenes del video. De manera general, lo que se busca hacer es crear un gráfico tridimensional de las posiciones de todos los nodos junto con los marcos de referencia local para

cada vértebra, para un instante, guardar ese gráfico y luego crear un nuevo gráfico, de manera que se grafiquen todos los datos que se tengan y que al final se tengan todos los gráficos guardados y pegados en un video.

La creación de cada imagen debe hacerse a través de un método. Entonces, se creó el método “make\_frame” que se encarga de crear una imagen utilizando la funcionalidad de gráfico tridimensional de matplotlib. Usando los atributos previamente definidos, se crea el gráfico. A este método se le puede añadir cualquiera otra funcionalidad que se desee, por ejemplo, la posición de la punta en todo momento, al igual que el marco de referencia local de cada vértebra en todo momento.

Entonces, se ejecuta la función “VideoClip()” de moviepy, llamando al método “make\_frame” y la duración final del video “final\_time”. Al finalizar la creación del video, le da un nombre al archivo mp4 y lo guarda en la carpeta designada.

Después de crear el video, el nodo de visualización permanece encendido y listo para recibir más información y crear nuevos videos. Esto es particularmente útil si se tiene la opción de ajuste de ganancias activada, puesto que creará nuevos videos en cada iteración de ganancias escogidas por el buscador, esto se verá a mayor detalle en la sección **4.5 Nodo de ajuste de ganancias**.

## **4.4 Nodo de control**

El nodo de control cumple una función muy importante en la red de comunicación de ROS2, cuando se quiere lograr el control de la posición de la punta de la varilla. Este nodo no tiene función si se desea realizar una simulación de lazo abierto utilizando la interfaz gráfica de usuario manual, y, por ende, “TendonForces” como módulo de forzado. Esto es porque en un lazo abierto, no hay retroalimentación del error que tiene el robot en su posición, por lo que no se recibe ni se envía información al nodo de control para que actualice las tensiones en el robot.

De manera general, lo que hace este nodo es recibir parámetros y también la posición deseada, definida a través del interfaz gráfico de usuario. Usando la posición deseada, el nodo controlador escoge cuáles tendones activar. En el caso de que se esté realizando un ajuste de ganancias, el nodo controlador también recibe las ganancias determinadas por el nodo de ajuste de ganancias. Entonces, usando las ganancias proporcionadas, calcula el error en la posición de la punta de la varilla y va ajustando las tensiones en los tendones.

#### ***4.4.1 Integración en el sistema***

La integración en la red de comunicación de ROS2 de este nodo es relativamente simple, en el sentido de que no se necesitan aplicar pasos adicionales para que pueda funcionar adecuadamente el nodo, sino que con definir publicadores y suscriptores es suficiente.

La inicialización del nodo crea los siguientes publicadores:

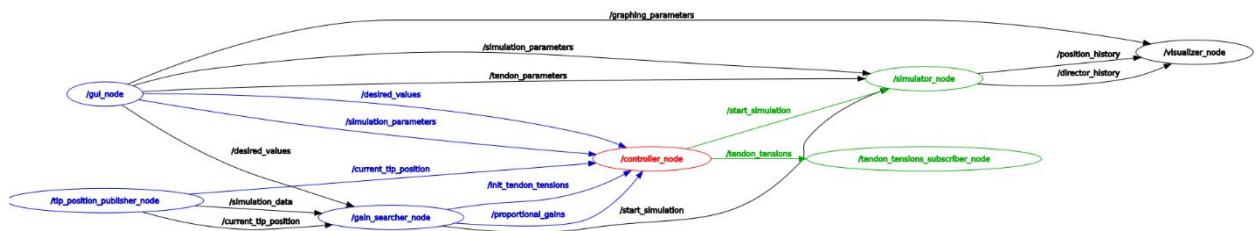
- “tendon\_tensions\_publisher”: se encarga de publicar en un mensaje los tendones que se activarán, además de las tensiones aplicadas a estos tendones. El mensaje se publica a través del tópico ‘/tendon\_tensions’.
- “start\_simulation\_publisher”: se encarga de publicar un número entero a través del tópico ‘/start\_simulation’, cuyo propósito es solamente iniciar la simulación de PyElastica en el nodo de simulación.

También crea los siguientes suscriptores:

- “desired\_values\_subscriber”: se encarga de recibir la posición XYZ deseada para la punta de la varilla, al igual que las ganancias a utilizar (en el caso de que se esté realizando un ajuste de ganancias, estas ganancias serían las iniciales) y también un valor booleano que determina si se va a realizar el ajuste de ganancias. El mensaje se recibe a través del tópico ‘/desired\_values’.

- “simulation\_parameters\_subscriber”: se encarga de recibir los mismos datos de simulación que se reciben en el nodo de simulación. El mensaje se recibe a través de ‘/simulation\_parameters’.
- “current\_tip\_pos\_subscriber”: recibe la información de la posición actual de la punta de la varilla a través del tópico ‘/current\_tip\_position’.
- “initialize\_tendon\_tensions\_subscriber”: se encarga de inicializar las tensiones en los tendones a cero. Es útil cuando se hace un ajuste de ganancia y se requiere que las tensiones regresen a cero para comenzar una nueva simulación. El mensaje se recibe a través del tópico ‘/init\_tendon\_tensions’.
- “proportional\_gains\_subscriber”: recibe las ganancias proporcionales que se enviaron por el nodo de ajuste de ganancias. El mensaje se recibe a través del tópico ‘/proportional\_gains’.

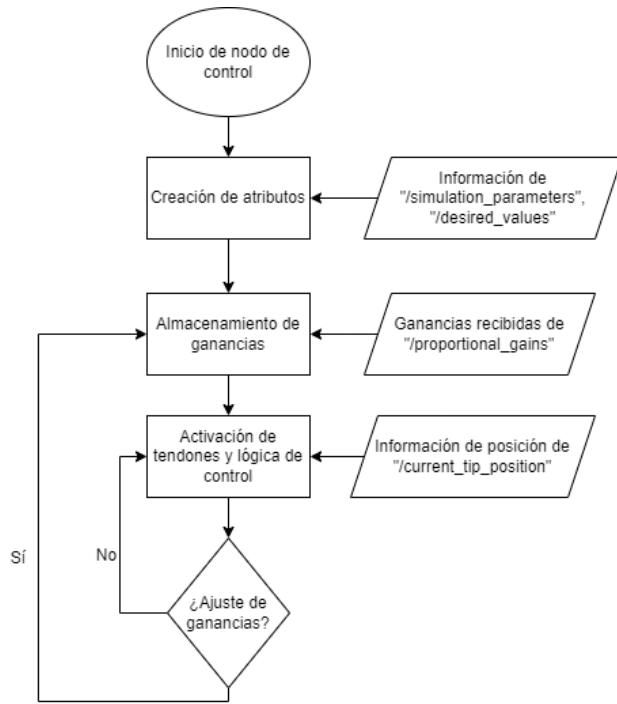
La conexión que tiene este nodo de control con el resto de la red de comunicación de ROS2 se puede ver en la siguiente **Figura 4-14**.



**Figura 4-14.** Gráfico RQT para el sistema de control, donde se señala el nodo de control en rojo, los tópicos a los cuales se suscribe en azul y a los que publica en verde.

#### 4.4.2 Funcionamiento

Se puede apreciar el funcionamiento de este nodo de control con el diagrama a continuación, seguido de una descripción de su funcionamiento:



**Figura 4-15.** Diagrama de flujo para el nodo de control.

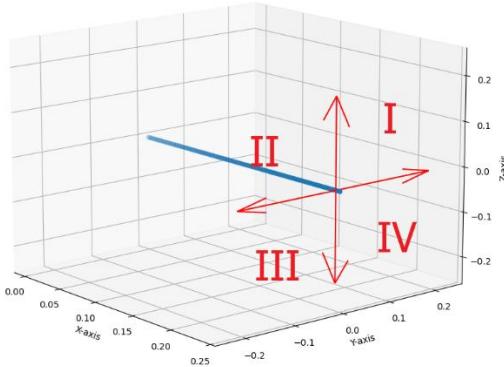
El funcionamiento del nodo de control hace posible que la punta de la varilla pueda llegar de manera confiable a la posición deseada por el usuario. Lo primero que se hace es importar la librería de numpy (NumPy Developers, n.d.). Luego, se comienza con la inicialización del nodo. Esta inicialización crea los publicadores y suscriptores mencionados en la sección previa, además que se establecen algunos atributos del objeto de nodo que serán de utilidad próximamente.

Lo primero que ocurre después de que el usuario introduzca y envíe todos los valores correspondientes en la interfaz gráfica de usuario es que se activa el método callback asociado al suscriptor “simulation\_parameters\_subscriber”, el cual tiene nombre “parameters\_callback”, que se encarga de recibir los parámetros de simulación y los guarda como atributos del objeto de nodo. Después, se activa el método callback asociado al suscriptor de nombre “initialize\_tendon\_tensions\_subscriber”, llamado “initialize\_tendon\_tensions”, que se encarga hacer cero los valores de las tensiones en los tendones, al igual que

reinicializa los tendones activados, de manera que no se tenga ningún tendon escogido por el momento. Siguiendo, se activa el método callback asociado al suscriptor “desired\_values\_subscriber”, llamado “desired\_values\_callback”, que se encarga de guardar como atributo del objeto de nodo los valores deseados por el usuario. En el caso que se esté realizando un ajuste de ganancias, se activará el método callback de “proportional\_gains\_subscriber”, llamado “proportional\_gains\_callback”, cada vez que se vaya a realizar una nueva simulación, cuya función es de guardar como atributo del objeto de nodo los valores de las ganancias proporcionadas.

Ahora, el nodo de control está preparado para controlar el robot. Lo que sigue es que el nodo de simulación envíe la posición actual de la punta de la varilla del robot, lo que activará el método callback asociado al suscriptor “current\_tip\_pos\_subscriber”, llamado “controller\_callback”, cuya función es realizar los cálculos y luego enviar la señal del control de regreso.

El funcionamiento del método “controller\_callback” es el siguiente: primero se determina en cuál cuadrante del espacio se encuentra la posición deseada por el usuario. Esto afectará cuáles tendones se escogen para activar y lograr el control ya que se estará utilizando la configuración de QuadTendonForces como se explicó en la sección **3.2.3 Estructura como módulo de forzado externo para el accionamiento por tendones**. La lógica para escoger cuáles tendones se activa se basa en escoger los dos tendones largos que corresponden al cuadrante en el que se encuentra la posición deseada, y luego se escogen los dos tendones cortos que son antagonistas a estos, es decir, en orientación opuesta. Una visualización de esta lógica se puede ver en la **Figura 4-16**.



**Figura 4-16.** Se demuestran los cuatro cuadrantes donde se puede ubicar el punto en el espacio deseado por el usuario.

La lógica para escoger los tendones es la siguiente: los tendones largos tendrán el signo correspondiente a las coordenadas deseadas respectivas, y los tendones cortos tendrán el signo opuesto a este. Por ejemplo, si el punto deseado es (0.22, -0.05, 0.09), entonces se escogerá el tendon largo horizontal ubicado en la dirección -Y, y el tendon largo vertical ubicado en la dirección +Z, debido al signo que tienen las coordenadas del punto deseado XYZ.

Después de escoger los tendones que serán activados, estos valores se almacenan en forma de un arreglo de punto flotante cuyos valores son de  $\pm 1.0$  para indicar la orientación de los tendones: vertical largo, horizontal largo, vertical corto y horizontal corto. En el caso de que el punto deseado se encuentre encima de uno de los ejes Y o Z, los valores de los tendones ubicados en este punto serían igual a cero.

Siguiendo, se procede a calcular las tensiones en los tendones que serán enviadas al nodo de simulación. Primero, se calculan las diferencias entre los valores actuales de la posición de la punta del robot (denotadas con el subíndice *actual*), y las deseadas (denotadas con el subíndice *deseada*), como se muestra a continuación:

$$dx = x_{actual} - x_{deseada}$$

$$dy = (y_{actual} - y_{deseada}) * \text{sign}(y_{deseada})$$

$$dz = (z_{actual} - z_{deseada}) * \text{sign}(z_{deseada})$$

Es importante incluir la multiplicación del signo de las coordenadas deseadas para  $y$  y  $z$ , puesto que esto ayuda a mantener la veracidad del cálculo de tensiones sin importar el cuadrante en el que se encuentre el punto deseado por el usuario. Esto se implementó porque surge un problema cuando se tienen puntos deseados con valores negativos, dependiendo del cuadrante en el que se quiera posicionar la punta del robot. Esto causa que los signos cambien y pueden producir tensiones negativas en el cálculo de tensiones, lo cual no es posible en este sistema.

Antes de proceder con el cálculo de las tensiones en los tendones actualizadas, se debe comentar acerca del método llamado “positive\_or\_zero”, que es parte del objeto de nodo. Este método lo único que hace es que verifica si el argumento que se le da es positivo. Si es positivo, entonces devuelve el mismo argumento como resultado. Si el argumento es negativo, entonces retorna un valor cero como resultado. El propósito de esto es que físicamente no se pueden tener tensiones negativas en un tendón, por lo que este método permite que numéricamente solamente se presenten tensiones positivas.

Ahora, procediendo con la actualización de las tensiones en los tendones, se calcula lo siguiente:

$$\text{tension vertical long} = \text{tension vertical long} - dz * K_{long}$$

$$\text{tension horizontal long} = \text{tension horizontal long} - dy * K_{long}$$

$$\text{tension vertical short} = \text{tension vertical long} + (dx + dz) * K_{short}$$

$$\text{tension horizontal short} = \text{tension horizontal short} + (dx + dy) * K_{short}$$

Donde  $K_{long}$  es la ganancia para los tendones largos,  $K_{short}$  es la ganancia para los tendones cortos, “tension vertical long” es la tensión para el tendón vertical largo, “tension horizontal long” es la tensión para el tendón horizontal largo,

“tension vertical short” es la tensión para el tendón vertical corto y “tension horizontal short” es la tensión para el tendón horizontal corto.

Una vez calculadas las nuevas tensiones, estas se almacenan y se preparan para ser publicadas por “tendon\_tensions\_publisher”. Después de enviar las tensiones actualizadas, el nodo queda en espera de nuevas posiciones actuales de la punta de la varilla del robot para aplicar el mismo control, hasta que terminen las simulaciones del nodo de simulación.

## **4.5 Nodo de ajuste de ganancias**

El nodo de ajuste de ganancias tiene como meta facilitar al usuario encontrar las mejores ganancias para los criterios de convergencia que se desean. Este nodo no tiene función si se desea realizar una simulación de lazo abierto utilizando la interfaz gráfica de usuario manual, y, por ende, “TendonForces” como módulo de forzado. Esto es porque la premisa de buscar ganancias implica el control del robot. Por ende, utilizar el interfaz gráfico manual, que solamente se ocupa de una simulación de lazo abierto, no es compatible con la función de este nodo. Esto no quiere decir que el sistema no permitirá realizar simulaciones, sino que este nodo de ajuste de ganancias no tendrá función en ese caso.

De manera general, este nodo se encarga de encontrar ganancias para los tendones que permitan lograr un control que se ajuste a los criterios del usuario. Para este trabajo, solamente se toman en consideración dos criterios: el tiempo de llegada de la punta de la varilla a la posición deseada, y también el sobrepaso máximo que experimenta el sistema después que llega por primera vez a la posición deseada. En base a los datos recopilados en la simulación realizada, se ajustan los valores de las ganancias y se vuelve a realizar la simulación, utilizando las nuevas ganancias en el nodo de control. Cuando se llegue cercano a los criterios, dentro de una tolerancia, se detiene el ajuste de ganancias y se imprimen las ganancias finales.

#### **4.5.1 Integración en el sistema**

La integración en la red de comunicación de ROS2 para este nodo requiere pasos adicionales además de establecer los publicadores y suscriptores. Esto es porque se requiere tener más de un proceso andando a la misma vez, similar al nodo de simulación. En el caso de este nodo de ajuste, es necesario tener el proceso de ajuste que inicializa las simulaciones y actualiza las ganancias, activo constantemente. A la misma vez, se necesita tener otro proceso andando que recibe la posición de la varilla y la procesa para saber si se llegó o no al punto deseado, al igual otro proceso que recibe el tiempo de simulación y lo guarda como atributo del objeto de nodo.

Por estas razones, se le debe dedicar un subprocesso al ajuste de ganancias, para que pueda funcionar de manera separada a todas las otras funciones del nodo. Esto se hace mediante el módulo de threading que proviene de la librería estándar de Python (Python Software Foundation, n.d.), que permite asignarle un “thread” o subprocesso a una función, en este caso el método “gain\_searcher” del objeto de nodo.

La inicialización del nodo crea los siguientes publicadores:

- “proportional\_gains\_publisher”: se encarga de enviar las ganancias proporcionales actualizadas a través del tópico ‘/proportional\_gains’.
- “initialize\_tendon\_tensions\_publisher”: se encarga de señalarle al nodo de control que inicialice las tensiones de los tendones y las haga igual a cero, a través del tópico ‘/init\_tendon\_tensions’.
- “start\_simulation\_publisher”: se encarga de señalarle al nodo de simulación que inicie una nueva simulación, a través del tópico ‘/start\_simulation’.

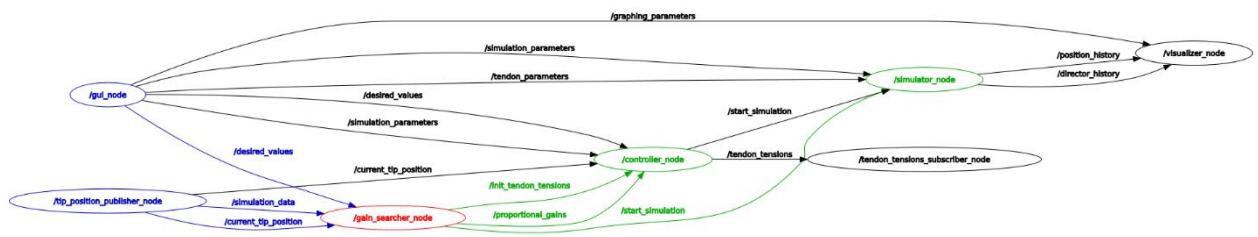
También, se crean los siguientes suscriptores:

- “desired\_values\_subscriber”: recibe los valores de posición XYZ deseados por el usuario, al igual que las ganancias iniciales y un valor booleano que

determina si se va a realizar el ajuste de ganancias o no. Este mensaje se recibe a través del tópico ‘/desired\_values’.

- “current\_tip\_pos\_subscriber”: recibe posición XYZ actuales de la punta del robot a través del tópico ‘/current\_tip\_position’.
- “simulation\_data\_subscriber”: recibe el tiempo actual de la simulación.

La conexión de este nodo con el resto de la red de comunicación en ROS2, se puede ver a continuación en la **Figura 4-17**.

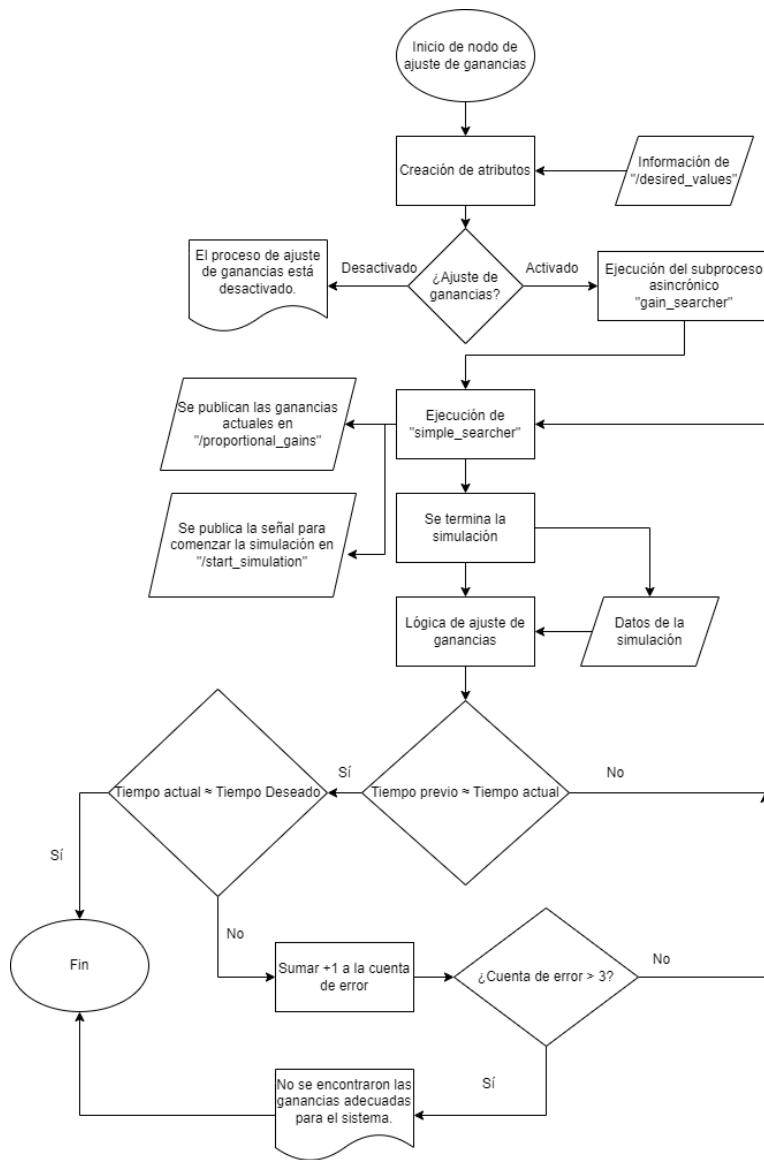


**Figura 4-17.** Gráfico RQT para el sistema de control con el nodo de ajuste de ganancias señalado en rojo, con los tópicos a los cuales se suscribe en azul y a los que publica en verde.

#### 4.5.2 Funcionamiento

El funcionamiento de este nodo hace posible que el usuario encuentre las ganancias que mejor se ajustan al sistema para que se cumplan los criterios establecidos, sin necesidad del usuario de incurrir a un método de ensayo y error manual, cambiando simulación tras simulación las ganancias del nodo de control.

Se puede apreciar el funcionamiento de este nodo de ajuste de ganancias con el diagrama a continuación, seguido de una descripción de su funcionamiento:



**Figura 4-18.** Diagrama de flujo para el nodo de ajuste de ganancias.

La inicialización del nodo crea los publicadores y suscriptores que se mencionaron en la sección previa, además que se crea el subproceso que será utilizado para el proceso de ajuste de ganancias, como se mencionó en la sección previa.

Lo que ocurre después de esto es que el usuario introduce todos los valores y parámetros en la interfaz gráfica de usuario para el control y que estos se envíen. Al ser enviados, el nodo de ajuste de ganancias activa el método callback

asociado al suscriptor “desired\_values\_subscriber”, el cual tiene nombre “desired\_values\_callback”, que se encarga de recibir los valores deseados, las ganancias iniciales, y el valor booleano que determina si se va a realizar un ajuste de ganancias, y guardas estos valores como atributos del objeto de nodo. También, este método comienza a andar el subprocesso designado para el ajuste de ganancias, lo que ejecuta el método asociado al subprocesso, llamado “gain\_searcher”.

Dependiendo del valor booleano guardado del paso anterior, el método “gain\_searcher” va a comenzar el ajuste. Si el valor booleano es “False”, entonces solamente se imprimirá un mensaje que lo indica. Si el valor booleano es “True”, entonces se comienza el ajuste de las ganancias. El método “gain\_searcher” solamente cumple la función de separar el proceso de ajuste de los otros procesos andantes, al designarle un subprocesso separado. La lógica y el funcionamiento del ajuste de ganancias se da en otro método que se referencia dentro de “gain\_searcher”, cuyo nombre es “simple\_searcher”.

El método “simple\_searcher” toma como argumento la función que inicializa el sistema y envía el mensaje para comenzar una nueva simulación. Dicha función es un método llamado “start\_simulation”. El método “start\_simulation” reinicializa varios atributos que deben ser redefinidos para que se pueda dar una nueva simulación. Esto se hace para asegurar que cada simulación que se realice tenga variables que no tengan influencia por simulaciones previas. Después de esta reinicialización, se envía un mensaje utilizando el publicador “initialize\_tendon\_tensions\_publisher”, para que se inicialicen en cero las tensiones en el nodo de control. Después, se envían las ganancias actuales en un mensaje utilizando el publicador “proportional\_gains\_publisher”, que le llega al controlador para que actualice las ganancias que utilizará en la simulación actual. Después, se envía un mensaje al nodo de simulación usando el publicador “start\_simulation\_publisher” para que se comience una nueva simulación de PyElastica. Posterior a esto, se va a esperar a que se termine la simulación, lo

cual se hace a través de un bucle cuyo propósito es seguir esperando hasta que un atributo booleano llamado “end\_simulation” se vuelva “True”. Si se cumple esta condición, entonces el método termina y retorna los valores de sobrepasso máximo y también la diferencia entre el tiempo deseado por el usuario y el tiempo que tomó para llegar a la posición por primera vez, valores que se calculan en el método “tip\_position\_callback”. Estos valores retornados se procesan en el método “simple\_searcher”.

Mientras se realiza la simulación en el nodo de simulación, el nodo de ajuste recibe información de la simulación al ejecutar el método callback asociado a “simulation\_data\_subscriber”, llamado “simulation\_data\_callback”. Este método callback se encarga de recibir el tiempo de simulación y lo guarda como atributo, al igual que controla cuándo el atributo booleano “end\_simulation” cambia de “False” a “True”.

También, se recibe la información de la posición XYZ actual del robot en la simulación, lo que activa el método callback asociado a “current\_tip\_pos\_subscriber”, llamado “tip\_position\_callback”. Este método callback tiene una de las funciones más importantes de este nodo: va guardando el tiempo en un atributo separado al que lo guarda “simulation\_data\_callback”, y lo hace hasta que la posición deseada XYZ para la punta de la varilla se logra por primera vez (esto es calculado mediante una norma de errores), dentro de una tolerancia especificada dentro del método. Si se cumple la llegada de la punta de la varilla a la posición deseada, se deja de buscar el tiempo y se busca el sobrepasso máximo que experimenta el sistema. Este paso consiste en calcular la norma del error tras cada iteración y comparar las normas de manera consecutiva, escogiendo siempre la mayor de las dos.

Cabe destacar que el cálculo de la norma se hace mediante un método llamado “norm\_calculator”, el cual está decorado con njit, y lo que hace es tomar como argumento la posición XYZ actual y deseada, calcula la diferencia entre cada

coordenada respectivamente, y luego utiliza la función de numpy “linalg.norm()” para calcular la norma del vector de error, retornando el resultado.

Regresando al método “simple\_searcher”, primero se inicializan algunas variables que se utilizarán posteriormente. Después se llama al método “start\_simulation” por primera vez, y se guardan los valores de sobrepaso máximo y diferencia de tiempo de llegada. Luego se comienza el bucle que llevará a cabo el ajuste de las ganancias del sistema. Este bucle tiene como condición que se debe cumplir un error menor a la tolerancia especificada en el método o que se tengan más que cinco cuentas de error (cada cuenta se suma al contador si la iteración no hizo progreso significativo). Entonces, el funcionamiento del bucle es el siguiente:

$$K_{new,long} = K_{prev,long} - t_{error} * K_{prev,long} - O_{error} * K_{prev,long}$$

$$K_{new,short} = K_{new,short} - t_{error} * K_{new,short} - O_{error} * K_{new,short}$$

Donde  $K_{new,long}$  es la nueva ganancia para los tendones largos,  $K_{new,short}$  es la nueva ganancia para los tendones cortos,  $K_{prev,long}$  es la ganancia previa para los tendones largos,  $K_{prev,short}$  es la ganancia previa para los tendones cortos,  $t_{error}$  es la diferencia entre el tiempo de llegada a la posición deseada y el tiempo deseado,  $O_{error}$  es el sobrepaso máximo que experimenta el sistema.

Luego, se verifica si el tiempo de llegada previo es muy cercano al nuevo, dentro de una tolerancia especificada en el método. Si esta condición es verdadera, entonces se suma una cuenta de error al contador que está adentro del bucle (hasta llegar a un máximo de 3 cuentas), si no, se sigue con el bucle. Posteriormente, se ejecuta nuevamente el método “start\_simulation”, utilizando las ganancias nuevas que fueron calculadas en el bucle y se analiza el tiempo de llegada la posición deseada, cuya magnitud dictará el error que funciona como condición del bucle. Si no se ha cumplido la condición, el bucle continúa iterando, calculando nuevas ganancias después de cada simulación realizada.

Si se logra llegar a un error muy bajo, o se llega a una cuenta de error mayor a cinco, el bucle se detiene y se retornan las últimas ganancias calculadas, las cuales son recibidas por “gain\_searcher”, que termina por imprimirlas. Después de realizar este proceso, el nodo de ajuste de ganancias ha realizado su función.

## **4.6 Resultados y análisis**

En esta sección, se discutirán los resultados de las simulaciones, el control y el ajuste de ganancias que fueron obtenidos mediante el uso de la red de comunicación de nodos en ROS2.

### ***4.6.1 Caso de lazo abierto***

Comenzando con los resultados obtenidos para las simulaciones de lazo abierto con el uso de la interfaz gráfica manual, estos fueron utilizados para validar el modelo de fuerzas y momento para el accionamiento mediante tendones. Los resultados obtenidos para el experimento No. 3 en la sección **3.3.6 Experimento No. 3** muestran que la red de comunicación en ROS2 para el caso del lazo abierto funciona de manera correcta, puesto que al comparar los resultados obtenidos de la simulación con los que se midieron en físico, se tiene una cercanía muy marcada. Por lo tanto, se puede decir que la comunicación entre los nodos es la correcta y se manejan los datos adecuadamente entre los nodos. Los resultados para este experimento se pueden ver en la sección **3.3.6 Experimento No. 3**, al igual que en la sección **6.1 Mediciones y simulaciones del experimento No. 3** en el anexo para observar todos los ensayos y mediciones.

### ***4.6.2 Caso de control retroalimentado***

Analizando los resultados del caso de control retroalimentado, se puede observar que se logró un control adecuado de la punta del robot simulado mediante el accionamiento por tendones y la utilización de la red de comunicación en ROS2. Los resultados para este caso de control retroalimentado se separan en cuatro tablas, cuya separación yace en el cuadrante en el espacio dentro del cual se

escogió el punto deseado allegar. Es decir, el espacio tridimensional de trabajo del robot se dividió en cuatro cuadrantes y para cada uno de los cuadrantes se escogieron tres posiciones deseadas. Se permitió que la simulación y su control ajustara su posición por cinco segundos.

Los resultados incluyen la posición XYZ deseada, la posición XYZ obtenida después de cinco segundos en tiempo de simulación, el error relativo entre la posición XYZ deseada y la obtenida, las tensiones de los tendones al final de la simulación, al igual que el tiempo de llegada a las posiciones XYZ deseadas dentro de un margen de tolerancia. Cabe destacar que las tensiones de los tendones incluyen un signo, cuya significancia es indicar cuál de los tendones fue el activado. Por ejemplo, una tensión negativa para el tendón vertical largo significa que el tendón que fue activado fue el tendón ubicado en la parte inferior (-Z global) del robot. De igual manera, una tensión negativa para el tendón horizontal corto significa que el tendón activado fue el que se ubica en la parte derecha del robot (-Y global). También, para referencia futura, se incluyeron las ganancias utilizadas para los tendones largos y los cortos.

Los resultados se muestran a continuación:

RESULTADOS PRIMER CUADRANTE			
	ENSAYO 1	ENSAYO 2	ENSAYO 3
X deseado (m)	0.22	0.21	0.15
X obtenido en 5s (m)	0.219651916	0.21186636	0.148944417
Y deseado (m)	0.05	0.03	0.1
Y obtenido en 5s (m)	0.049937226	0.029853161	0.100102972
Z deseado (m)	0.08	0.1	0.12
Z obtenido en 5s (m)	0.080162923	0.099847022	0.120106395
Error Rel. X	0.16%	-0.89%	0.70%
Error Rel. Y	0.13%	0.49%	-0.10%
Error Rel. Z	-0.20%	0.15%	-0.09%
Tensión vertical larga (N)	3.461641065	4.020862563	4.861951817

Tensión horizontal larga (N)	1.223572472	0.867856829	2.850387278
Tensión vertical corta (N)	0	-0.23634271	-0.33356084
Tensión horizontal corta (N)	-0.07759137	-0.59118801	-0.73117844
Ganancia larga	3.0	3.0	3.0
Ganancia corta	1.0	1.0	1.0
Tiempo de llegada (s)	0.5997246	0.633043773	1.266087545

**Tabla 4-1.** Resultados obtenidos para los tres ensayos de control realizados para el primer cuadrante del espacio de trabajo del robot.

RESULTADOS SEGUNDO CUADRANTE			
	ENSAYO 1	ENSAYO 2	ENSAYO 3
X deseado (m)	0.18	0.19	0.24
X obtenido en 5s (m)	0.176421347	0.188671978	0.242608249
Y deseado (m)	-0.1	-0.05	-0.04
Y obtenido en 5s (m)	-0.1000351	-0.05009909	-0.03978952
Z deseado (m)	0.1	0.12	0.02
Z obtenido en 5s (m)	0.100015224	0.120001694	0.019791498
Error Rel. X	1.99%	0.70%	-1.09%
Error Rel. Y	-0.04%	-0.20%	0.53%
Error Rel. Z	-0.02%	0.00%	1.04%
Tensión vertical larga (N)	4.105566108	4.532824536	0.083297128
Tensión horizontal larga (N)	-2.55171879	-1.34128581	-0.13181238
Tensión vertical corta (N)	0	0	0
Tensión horizontal corta (N)	0	0.425245456	0
Ganancia larga	3.0	3.0	3.0
Ganancia corta	1.0	1.0	1.0
Tiempo de llegada (s)	0.966224705	0.966224705	0.566407586

**Tabla 4-2.** Resultados obtenidos para los tres ensayos de control realizados para el segundo cuadrante del espacio de trabajo del robot.

RESULTADOS TERCER CUADRANTE			
	ENSAYO 1	ENSAYO 2	ENSAYO 3
X deseado (m)	0.22	0.19	0.16
X obtenido en 5s (m)	0.225629156	0.19854871	0.159043739
Y deseado (m)	-0.05	-0.05	-0.08
Y obtenido en 5s (m)	-0.04954266	-0.04929822	-0.08007928
Z deseado (m)	-0.08	-0.12	-0.14
Z obtenido en 5s (m)	-0.07953857	-0.11925331	-0.14008686
Error Rel. X	-2.56%	-4.50%	0.60%
Error Rel. Y	0.91%	1.40%	-0.10%
Error Rel. Z	0.58%	0.62%	-0.06%
Tensión vertical larga (N)	-0.28870659	-2.31953834	-3.02164267
Tensión horizontal larga (N)	-0.22624916	-1.79383503	-2.64085402
Tensión vertical corta (N)	0	1.368962271	0.395583655
Tensión horizontal corta (N)	0	1.420874224	0.430955145
Ganancia larga	3.0	3.0	3.0
Ganancia corta	1.0	1.0	1.0
Tiempo de llegada (s)	0.533089493	2.865356023	1.299405638

**Tabla 4-3.** Resultados obtenidos para los tres ensayos de control realizados para el tercer cuadrante del espacio de trabajo del robot.

RESULTADOS CUARTO CUADRANTE			
	ENSAYO 1	ENSAYO 2	ENSAYO 3
X deseado (m)	0.16	0.17	0.21
X obtenido en 5s (m)	0.159043714	0.175165225	0.215296655
Y deseado (m)	0.08	0.02	0.1
Y obtenido en 5s (m)	0.080079282	0.019591346	0.099593859
Z deseado (m)	-0.14	-0.15	-0.04
Z obtenido en 5s (m)	-0.14008687	-0.14951625	-0.03966898
Error Rel. X	0.60%	-3.04%	-2.52%
Error Rel. Y	-0.10%	2.04%	0.41%
Error Rel. Z	-0.06%	0.32%	0.83%

Tensión vertical larga (N)	-1.35027083	-3.39562477	-0.00811777
Tensión horizontal larga (N)	0.875865462	1.00407809	2.824932375
Tensión vertical corta (N)	0	0.951235243	1.763597717
Tensión horizontal corta (N)	0.044703067	-1.598196	-0.54233427
Ganancia larga	3.0	3.0	3.0
Ganancia corta	1.0	1.0	1.0
Tiempo de llegada (s)	1.299405638	1.599268478	1.132815172

**Tabla 4-4.** Resultados obtenidos para los tres ensayos de control realizados para el cuarto cuadrante del espacio de trabajo del robot.

Como se puede observar en las cuatro tablas de resultados presentadas, los errores relativos entre las posiciones deseadas y las obtenidas son bastante bajos. Esto significa que el control, por lo general, tiende a ser efectivo y se puede llegar a la posición deseada. Sin embargo, se debe notar que esto no es cierto para todos los casos, puesto que existen poses que simplemente están fuera del espacio de trabajo del robot. Esto se puede observar, por ejemplo, en la **Tabla 4-3**, en el segundo ensayo, donde el error relativo en X es de -4.5%. Hay casos en los que no se puede llegar a las posiciones deseadas por el usuario, y lo que ocurre normalmente es que se cumplan las posiciones deseadas de Y y Z, dejando un error relativo mayor en la coordenada X. En las tablas presentadas, solamente hay posiciones en el espacio a las cuales el robot puede llegar con facilidad y con la configuración de tendones que se planteó para el control.

En base a estos resultados, se puede concluir que el nodo de control puede efectivamente comunicarse con el nodo de simulación para ajustar las tensiones de los tendones para lograr el control de la posición de la punta del robot. Esto significa que las conexiones funcionan de manera correcta y la red de comunicación en ROS2 es exitosa.

Las gráficas finales de las simulaciones realizadas para las tablas presentadas se pueden ver en la sección **6.2 Resultados de los ensayos de control** en el anexo.

### **4.6.3 Caso de ajuste de ganancias para el control retroalimentado**

Como se explicó en la sección **4.5 Nodo de ajuste de ganancias**, este nodo se encarga de iterar tras simulaciones en PyElastica y ajustar los valores de las ganancias para que el control del robot cumpla con criterios establecidos por el usuario.

Los resultados obtenidos al utilizar este nodo dentro de la red de comunicación en ROS2 muestran una buena efectividad para encontrar las mejores ganancias para el criterio escogido por el usuario de tiempo de llegada a la posición deseada. Todos los ajustes de ganancias comenzaron con las ganancias default, las cuales son de 3.0 para los tendones largos y 1.0 para los tendones cortos. A través de varias simulaciones, el nodo de ajuste de ganancias logró ajustar las ganancias hasta lograr el criterio establecido.

Los resultados presentados a continuación tienen la siguiente información: el tiempo deseado por el usuario, el tiempo obtenido después del ajuste de las ganancias, el sobrepaso máximo experimentado por el sistema después de llegar a la cercanía del punto deseado, la cantidad de iteraciones (simulaciones) que fueron llevadas a cabo para terminar el ajuste de ganancias, al igual que la ganancia de los tendones largos y la de los tendones cortos. La tabla de resultados es la siguiente:

RESULTADOS PARA AJUSTE DE GANANCIAS				
	1C-1	2C-2	3C-3	4C-1
Tiempo deseado (s)	1.5	1.0	1.0	1.5
Tiempo obtenido (s)	1.499314198	0.999542799	0.999542799	1.499314198
Sobrepaso máximo (m)	0.008977217	0.009551161	0.007466983	0.009301721
Iteraciones	8	4	2	5
Ganancia larga	1.539950815	2.468720188	3.872372858	2.608717027
Ganancia corta	0.513316938	0.822906729	1.290790953	0.869572342

**Tabla 4-5.** Resultados del ajuste de ganancias, donde 1C-1 se refiere al primer cuadrante y su primer ensayo, 2C-2 se refiere al segundo cuadrante y su

segundo ensayo, 3C-3 se refiere al tercer cuadrante y su tercer ensayo y 4C-1 se refiere al cuarto cuadrante y su primer ensayo

Se puede observar de los resultados obtenidos que el ajuste de ganancias fue exitoso. Se nota que los valores del tiempo obtenido para cada uno de los casos son muy cercanos a los tiempos deseados. Se observa también que el sobrepasso máximo obtenido para cada uno de los casos es relativamente pequeño dado la longitud del robot de 0.25cm.

Las ganancias obtenidas son diferentes para los casos puesto que se tienen tiempos deseados diferentes, al igual que algunas posiciones requieren diferentes magnitudes de ajuste por los tendones. Es decir, algunas posiciones requieren que los tendones cortos estén más activos que en otras, y dado que estos tienen menor magnitud de ganancia, son más lentos en recibir aumento en la magnitud de su tensión el nodo de control.

Estos resultados son consistentes al comprobar las ganancias encontradas con la interfaz gráfica de usuario de control, con el ajuste de ganancias desactivado. Se nota que, con cualquiera de las posiciones deseadas y sus ganancias respectivas de la **Tabla 4-5**, se llega a la posición deseada en el tiempo que fue deseado previamente.

## 5 Conclusión

Este trabajo ha abordado de manera integral el estudio y aplicación de la teoría de varillas de Cosserat para el caso de estudio de robótica continua acoplada con un accionamiento mediante tendones. Se desarrollaron los fundamentos de la teoría de modelado de varillas de Cosserat, al igual que se comentaron los avances hechos por los desarrolladores del paquete de simulación Elastica para llegar al simulador utilizado en este trabajo, PyElastica.

Para lograr el acople del accionamiento de tendones al sistema modelado por una varilla de Cosserat dentro del software de simulación PyElastica, se desarrolló un modelo de fuerzas y momentos que pudiera ser fácilmente implementado en código en forma de módulo, el cual fue posteriormente validado mediante experimentación en físico para asegurar que tuviera resultados certeros.

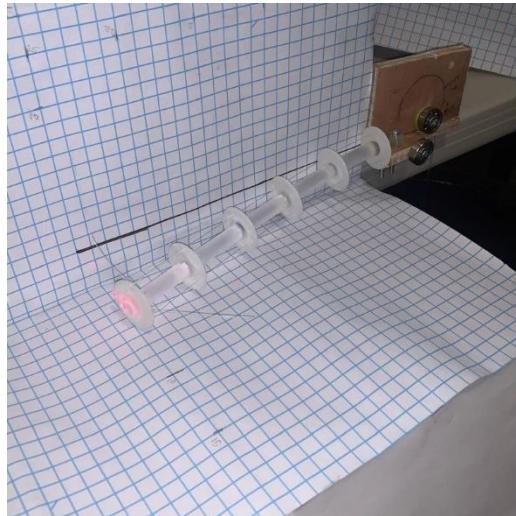
Como tema central al trabajo, se diseñó una red de comunicación en ROS2 para permitir la interconexión de nodos, que, juntos funcionan para crear un entorno multifuncional de simulación y control. Esta red de comunicación en ROS2 se demostró ser efectiva en la simulación del robot accionado por tendones al igual que facilita la experiencia del usuario puesto que no se necesita interactuar con el código directamente, sino que todo se puede hacer a través de la interfaz gráfica de usuario. Además, la comunicación de los nodos relacionados al control permitió obtener un control exitoso para los puntos ubicados en el espacio de trabajo funcional del robot presentado, puesto que la integración con el simulador PyElastica fue exitosa también.

Este trabajo permite el desarrollo de futuros proyectos relacionados a la robótica continua, especialmente aquella que utiliza tendones para el accionamiento del robot. La facilidad de simulación mediante las interfaces gráficas de usuario, al igual que el beneficio de utilizar una estructura modular en ROS2, da camino a muchas diferentes posibilidades de desarrollo a futuro, tales como: implementación de un nodo conectado a un microprocesador para interactuar con

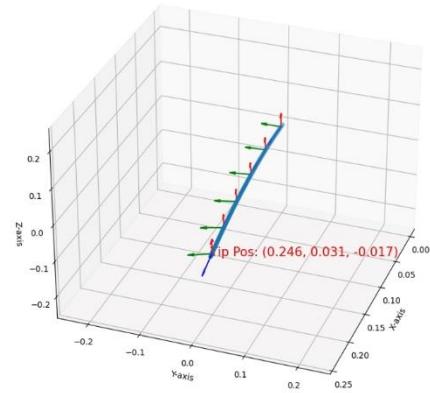
un sistema físico, lograr simulaciones con múltiples robots a la misma vez o implementar esquemas de control más robustos.

## 6 Anexos

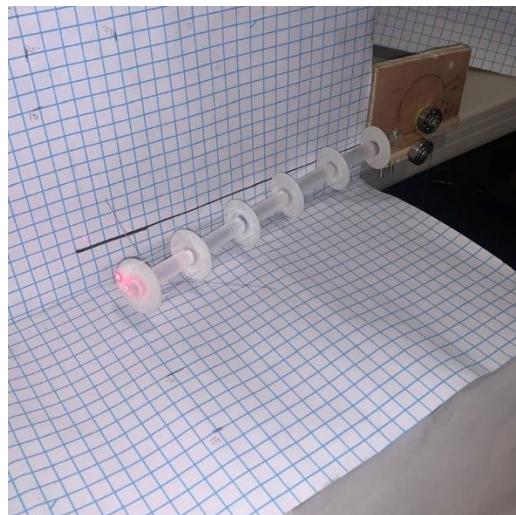
### 6.1 Mediciones y simulaciones del experimento No. 3



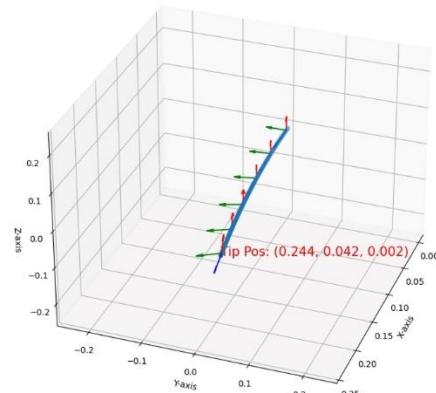
Pose para ensayo No. 1 del experimento No. 3,  
 $T_{\text{vertical}} = 1.0333\text{N}$ ,  $T_{\text{horizontal}} = 0.7882\text{N}$ ,  
 $E = 16.5986 \text{ MPa}$ ,  $G = 7.2169 \text{ MPa}$



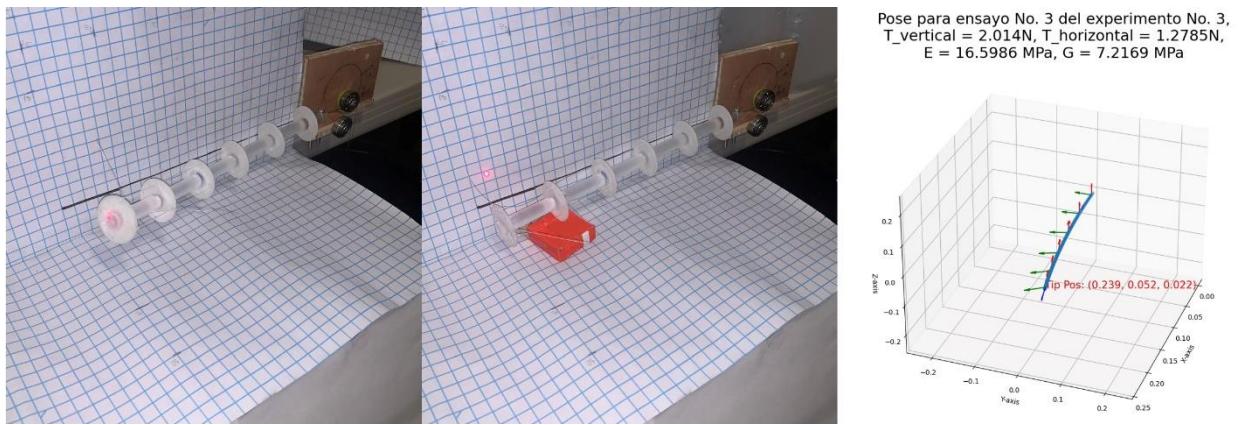
**Figura 6-1.** Medición y simulación para el ensayo 1 del experimento No. 3.



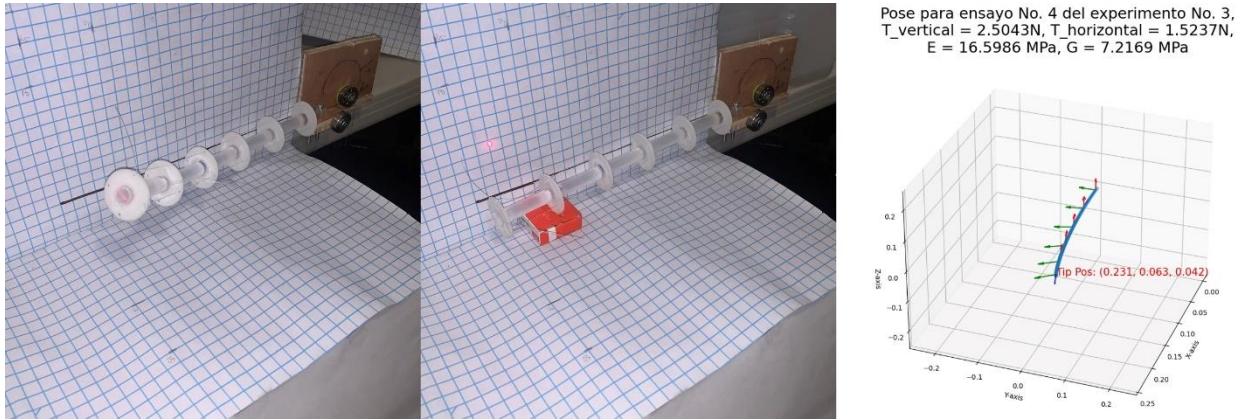
Pose para ensayo No. 2 del experimento No. 3,  
 $T_{\text{vertical}} = 1.5237\text{N}$ ,  $T_{\text{horizontal}} = 1.0333\text{N}$ ,  
 $E = 16.5986 \text{ MPa}$ ,  $G = 7.2169 \text{ MPa}$



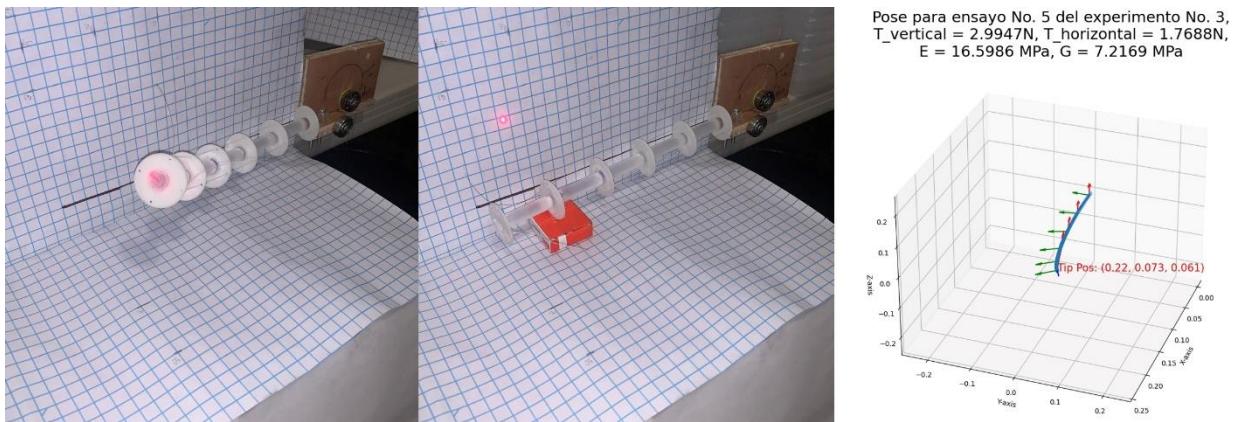
**Figura 6-2.** Medición y simulación para el ensayo 2 del experimento No. 3.



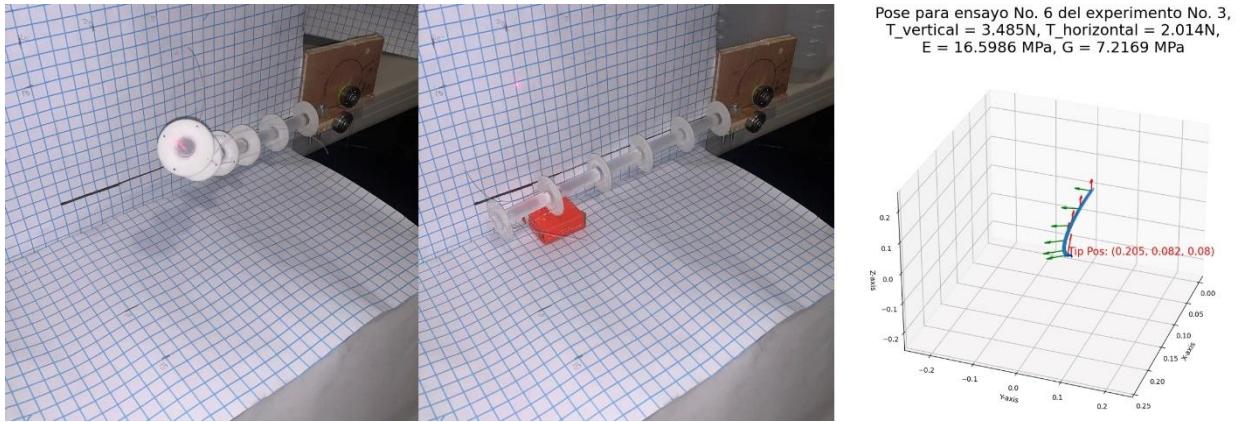
**Figura 6-3.** Medición y simulación para el ensayo 3 del experimento No. 3.



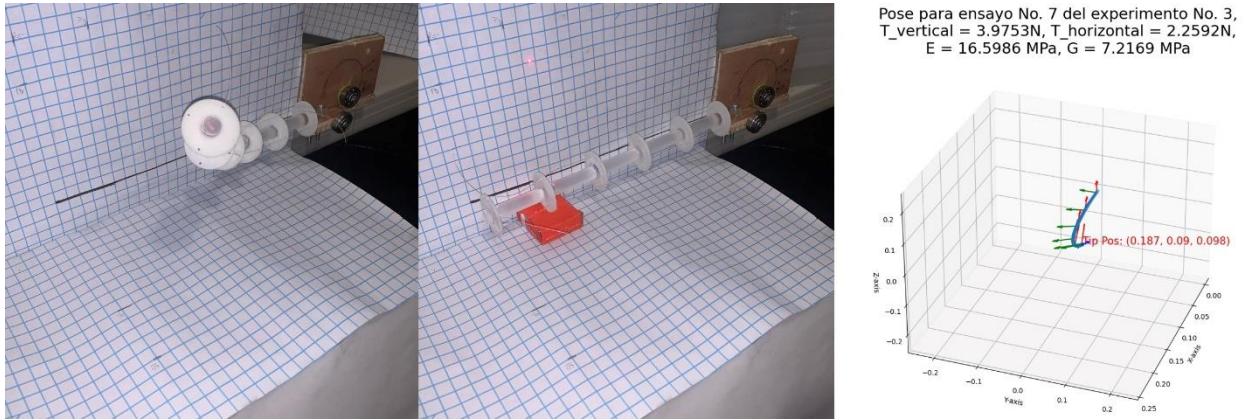
**Figura 6-4.** Medición y simulación para el ensayo 4 del experimento No. 3.



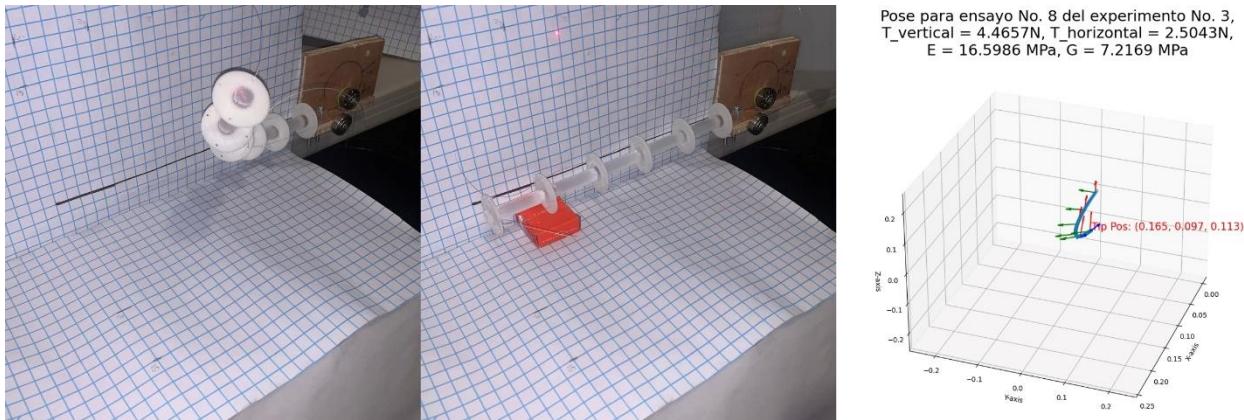
**Figura 6-5.** Medición y simulación para el ensayo 5 del experimento No. 3.



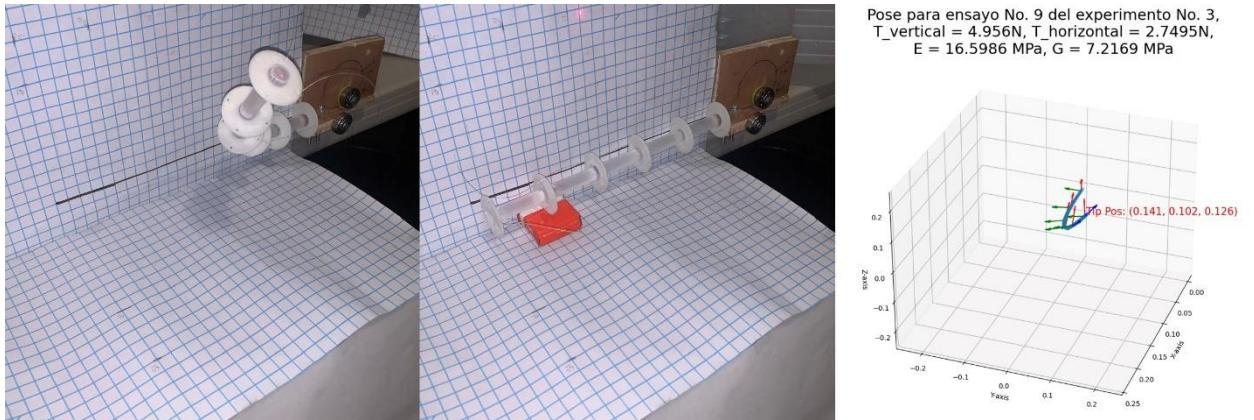
**Figura 6-6.** Medición y simulación para el ensayo 6 del experimento No. 3.



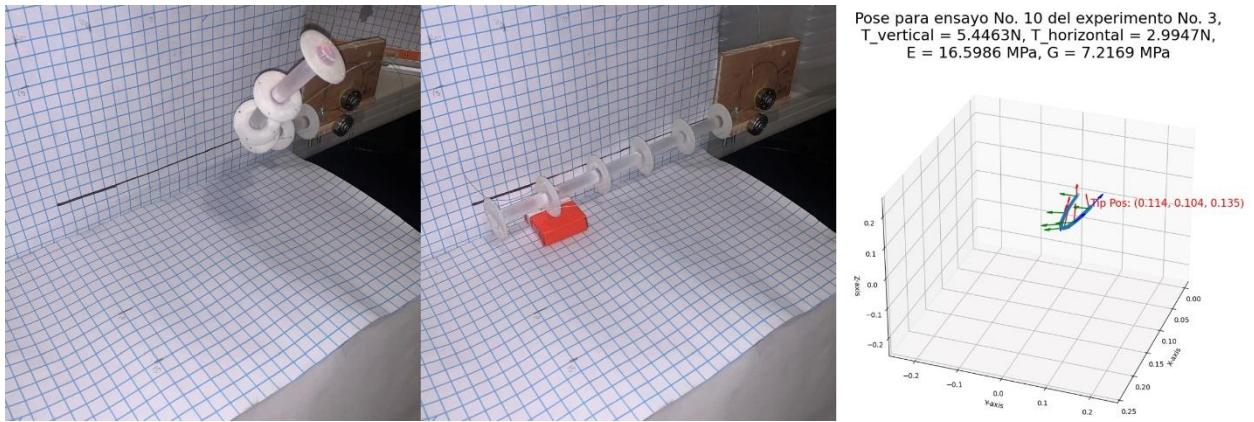
**Figura 6-7.** Medición y simulación para el ensayo 7 del experimento No. 3.



**Figura 6-8.** Medición y simulación para el ensayo 8 del experimento No. 3.



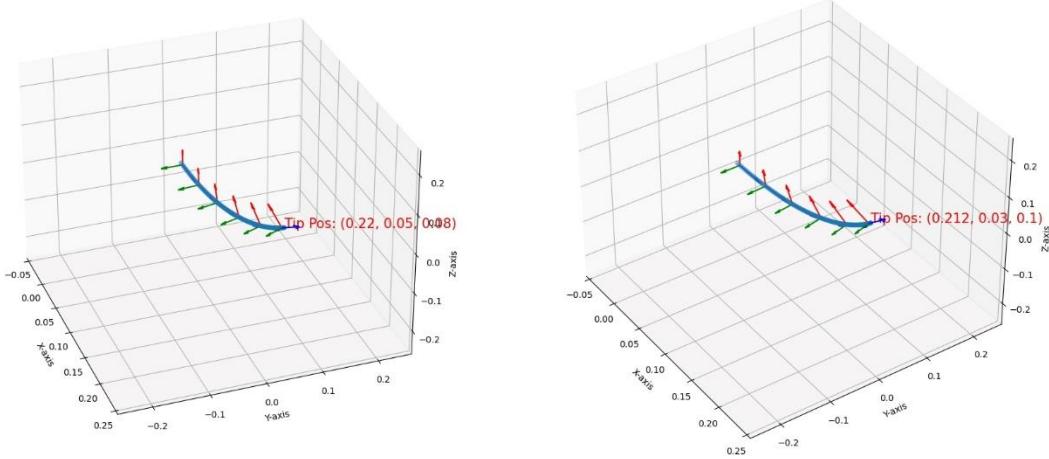
**Figura 6-9.** Medición y simulación para el ensayo 9 del experimento No. 3.



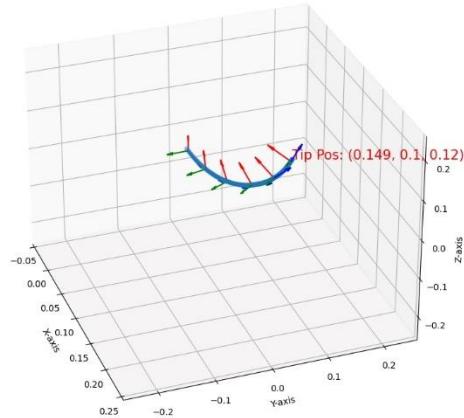
**Figura 6-10.** Medición y simulación para el ensayo 10 del experimento No. 3.

## 6.2 Resultados de los ensayos de control

Pose final para el control del robot con posición deseada: Pose final para el control del robot con posición deseada:  
 X:0.22, Y:0.05, Z:0.08 X:0.21, Y:0.03, Z:0.1

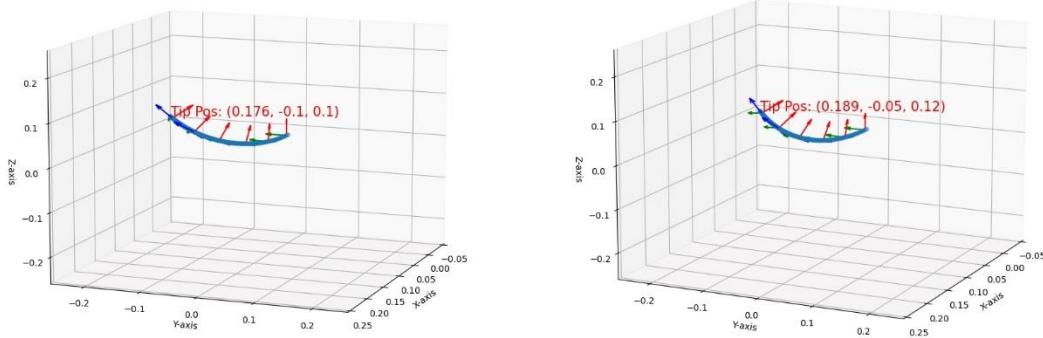


Pose final para el control del robot con posición deseada:  
 X:0.15, Y:0.1, Z:0.12

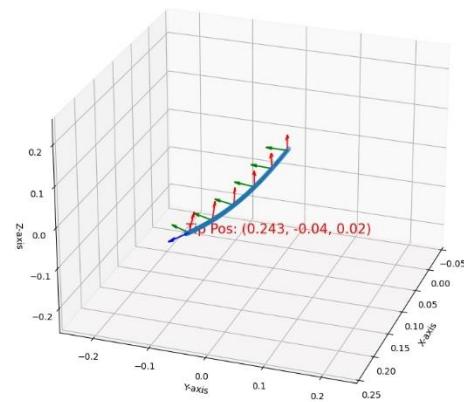


**Figura 6-11.** Poses finales de las simulaciones controladas después de 5 segundos en tiempo de simulación. Estos tres ensayos son para el primer cuadrante del espacio de trabajo.

Pose final para el control del robot con posición deseada: Pose final para el control del robot con posición deseada:  
 X:0.18, Y:-0.1, Z:0.1 X:0.19, Y:-0.05, Z:0.12

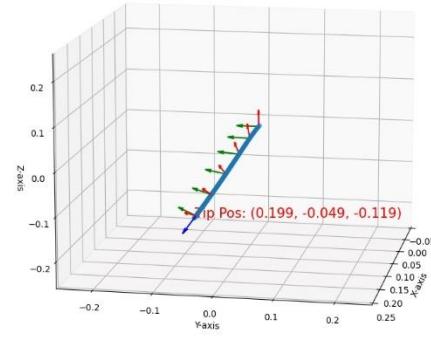
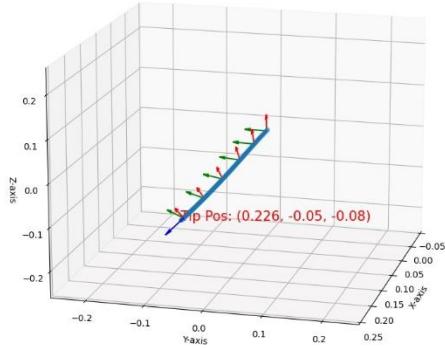


Pose final para el control del robot con posición deseada:  
 X:0.24, Y:-0.04, Z:0.02

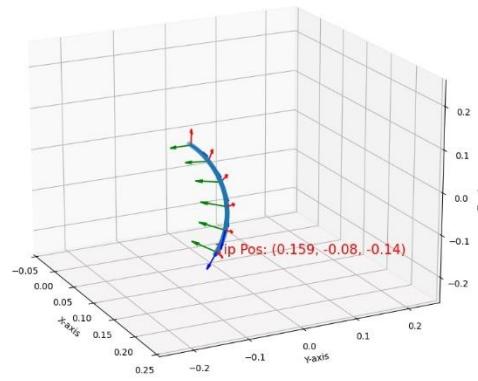


**Figura 6-12.** Poses finales de las simulaciones controladas después de 5 segundos en tiempo de simulación. Estos tres ensayos son para el segundo cuadrante del espacio de trabajo.

Pose final para el control del robot con posición deseada: Pose final para el control del robot con posición deseada:  
 X:0.22, Y:-0.05, Z:-0.08 X:0.19, Y:-0.05, Z:-0.12



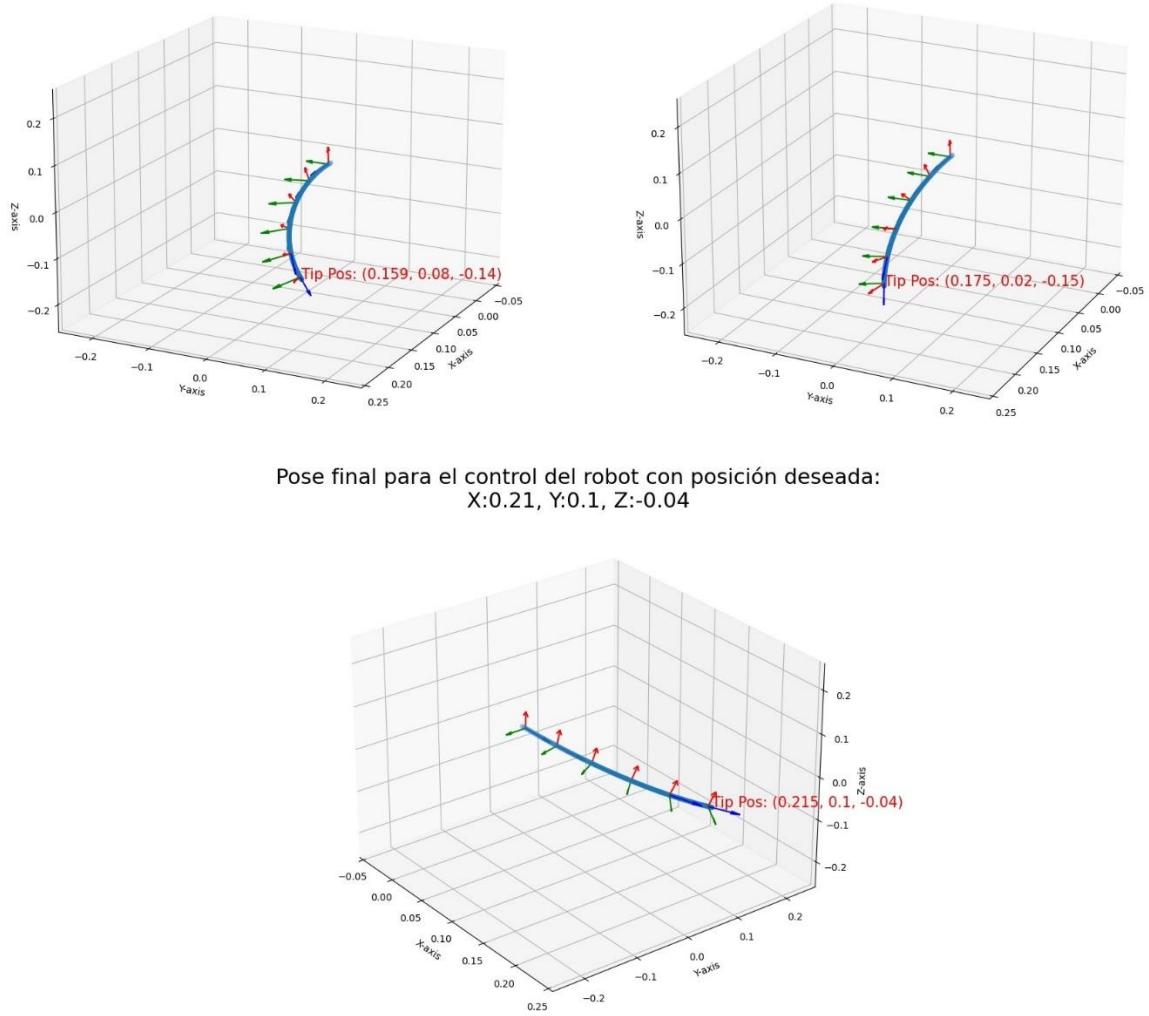
Pose final para el control del robot con posición deseada:  
 X:0.16, Y:-0.08, Z:-0.14



**Figura 6-13.** Poses finales de las simulaciones controladas después de 5 segundos en tiempo de simulación. Estos tres ensayos son para el tercer cuadrante del espacio de trabajo.

Pose final para el control del robot con posición deseada: X:0.16, Y:0.08, Z:-0.14

Pose final para el control del robot con posición deseada: X:0.21, Y:0.1, Z:-0.04



Pose final para el control del robot con posición deseada:  
X:0.21, Y:0.1, Z:-0.04

**Figura 6-14.** Poses finales de las simulaciones controladas después de 5 segundos en tiempo de simulación. Estos tres ensayos son para el cuarto cuadrante del espacio de trabajo.

### 6.3 Código desarrollado e implementado

Todo el código que fue desarrollado en este trabajo puede ser encontrado en los siguientes repositorios:

- **TendonForces**: Módulo de forzado externo validado experimentalmente para ser utilizado en el software de simulación PyElastica. Este módulo aplica los efectos de forzado externo para la actuación mediante tendones. (<https://github.com/gabotuzl/TendonForces>)
- **tendon\_cr\_environment**: Entorno en ROS2 para el simulado y control de un robot continuo accionado por tendones. Incluye todo lo necesario para llevar a cabo las simulaciones realizadas en este trabajo.  
([https://github.com/gabotuzl/tendon\\_cr\\_environment](https://github.com/gabotuzl/tendon_cr_environment))

## 7 Referencias bibliográficas

- Anaconda Inc. (n.d.). *Numba documentation*. Retrieved August 21, 2024, from <https://numba.readthedocs.io/en/stable/index.html>
- Antman, S. S. (1995). The Special Cosserat Theory of Rods. In F. John, J. E. Marsden, & L. Sirovich (Eds.), *Nonlinear Problems of Elasticity* (Vol. 107, pp. 259–324). Springer New York. [https://doi.org/10.1007/978-1-4757-4147-6\\_8](https://doi.org/10.1007/978-1-4757-4147-6_8)
- Burgner, J., Swaney, P. J., Lathrop, R. A., Weaver, K. D., & Webster, R. J. (2013). Debulking From Within: A Robotic Steerable Cannula for Intracerebral Hemorrhage Evacuation. *IEEE Transactions on Biomedical Engineering*, 60(9), 2567–2575. <https://doi.org/10.1109/TBME.2013.2260860>
- Burgner-Kahrs, J., Rucker, D. C., & Choset, H. (2015). Continuum Robots for Medical Applications: A Survey. *IEEE Transactions on Robotics*, 31(6), 1261–1280. <https://doi.org/10.1109/TRO.2015.2489500>
- Chairopoulos, N. C., Vartholomeos, P., & Papadopoulos, E. (2019). Modeling, Simulation and Experimental Validation of a Tendon-driven Soft-arm Robot Configuration—A Continuum Mechanics Method. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5695–5700. <https://doi.org/10.1109/IROS40897.2019.8968556>
- Chang, H.-S., Halder, U., Shih, C.-H., Tekinalp, A., Parthasarathy, T., Gribkova, E., Chowdhary, G., Gillette, R., Gazzola, M., & Mehta, P. G. (2020). Energy Shaping

- Control of a CyberOctopus Soft Arm. *2020 59th IEEE Conference on Decision and Control (CDC)*, 3913–3920. <https://doi.org/10.1109/CDC42340.2020.9304408>
- Clark, J. A. (n.d.). *Pillow*. Pillow (PIL Fork). Retrieved August 23, 2024, from <https://pillow.readthedocs.io/en/stable/index.html>
- Cosserat, E., & Cosserat, F. (1909). *Théorie des corps déformables*. Hermann.
- Dixon, M., & Reich, S. (2004). Symplectic Time-Stepping for Particle Methods. *GAMM-Mitteilungen*, 27(1), 9–24. <https://doi.org/10.1002/gamm.201490005>
- Dong, X., Axinte, D., Palmer, D., Cobos, S., Raffles, M., Rabani, A., & Kell, J. (2017). Development of a slender continuum robotic system for on-wing inspection/repair of gas turbine engines. *Robotics and Computer-Integrated Manufacturing*, 44, 218–229. <https://doi.org/10.1016/j.rcim.2016.09.004>
- Gao, A., Zou, Y., Wang, Z., & Liu, H. (2017). A General Friction Model of Discrete Interactions for Tendon Actuated Dexterous Manipulators. *Journal of Mechanisms and Robotics*, 9(4), Article 4. <https://doi.org/10.1115/1.4036719>
- Gazzola Lab. (2024). *PyElastica Documentation—PyElastica*. PyElastica Documentation. <https://docs.cosserat rods.org/en/latest/>
- Gazzola, M., Dudte, L. H., McCormick, A. G., & Mahadevan, L. (2018). Forward and inverse problems in the mechanics of soft filaments. *Royal Society Open Science*, 5(6), Article 6. <https://doi.org/10.1098/rsos.171628>

- Giri, N., & Walker, I. (2011). Continuum robots and underactuated grasping. *Mechanical Sciences*, 2(1), 51–58. <https://doi.org/10.5194/ms-2-51-2011>
- Hockstein, N. G., Gourin, C. G., Faust, R. A., & Terris, D. J. (2007). A history of robots: From science fiction to surgical robotics. *Journal of Robotic Surgery*, 1(2), Article 2. <https://doi.org/10.1007/s11701-007-0021-2>
- Huang, W., Huang, X., Majidi, C., & Jawed, M. K. (2020). Dynamic simulation of articulated soft robots. *Nature Communications*, 11(1), Article 1. <https://doi.org/10.1038/s41467-020-15651-9>
- Kato, T., Okumura, I., Song, S.-E., Golby, A. J., & Hata, N. (2015). Tendon-Driven Continuum Robot for Endoscopic Surgery: Preclinical Development and Validation of a Tension Propagation Model. *IEEE/ASME Transactions on Mechatronics*, 20(5), Article 5. <https://doi.org/10.1109/TMECH.2014.2372635>
- Kumar, A. (2010). *Theoretical and Computational Challenges with Rods* [Cornell University]. <https://ecommons.cornell.edu/server/api/core/bitstreams/406f1c5e-1b51-4bda-98ee-33c3b132c0bb/content>
- Kumar, A., Kumar, S., & Gupta, P. (2016). A Helical Cauchy-Born Rule for Special Cosserat Rod Modeling of Nano and Continuum Rods. *Journal of Elasticity*, 124(1), 81–106. <https://doi.org/10.1007/s10659-015-9562-1>
- Li, M., Kang, R., Geng, S., & Guglielmino, E. (2018). Design and control of a tendon-driven continuum robot. *Transactions of the Institute of Measurement and Control*, 40(11), Article 11. <https://doi.org/10.1177/0142331216685607>

Naughton, N., Sun, J., Tekinalp, A., Chowdhary, G., & Gazzola, M. (2020). *Elastica: A compliant mechanics environment for soft robotic control* (arXiv:2009.08422; Issue arXiv:2009.08422). arXiv. <http://arxiv.org/abs/2009.08422>

NumPy Developers. (n.d.). *NumPy documentation*. Retrieved August 21, 2024, from <https://numpy.org/doc/stable/>

O'Reilly, O. M. (2017). *Modeling Nonlinear Problems in the Mechanics of Strings and Rods*. Springer International Publishing. <https://doi.org/10.1007/978-3-319-50598-5>

Python Software Foundation. (n.d.). *The Python Standard Library*. Python Documentation. Retrieved August 20, 2024, from <https://docs.python.org/3/library/index.html>

Qadloori Fenjan, S., & Fathollahi Dehkordi, S. (2024). Soft Robotic System with Continuum Manipulator and Compliant Gripper: Design, Fabrication, and Implementation. *Actuators*, 13(8), 298. <https://doi.org/10.3390/act13080298>

Rao, P., Peyron, Q., Lilge, S., & Burgner-Kahrs, J. (2021). How to Model Tendon-Driven Continuum Robots and Benchmark Modelling Performance. *Frontiers in Robotics and AI*, 7, 630245. <https://doi.org/10.3389/frobt.2020.630245>

Reddy, J. N. (2004). *An Introduction to Nonlinear Finite Element Analysis*. Oxford University Press. <https://doi.org/10.1093/acprof:oso/9780198525295.001.0001>

Robinson, G., & Davies, J. B. C. (1999). Continuum robots—A state of the art. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat.*

No.99CH36288C), 4, 2849–2854 vol.4.  
<https://doi.org/10.1109/ROBOT.1999.774029>

Rodrigues, O. (n.d.). *Des lois géométriques qui régissent les déplacements d'un système solide dans l'espace, et de la variation des coordonnées provenant de ces déplacements considérés indépendamment des causes qui peuvent les produire.*

Rucker, D. C., & Webster III, R. J. (2011). Statics and Dynamics of Continuum Robots With General Tendon Routing and External Loading. *IEEE Transactions on Robotics*, 27(6), Article 6. <https://doi.org/10.1109/TRO.2011.2160469>

Russo, M., Sadati, S. M. H., Dong, X., Mohammad, A., Walker, I. D., Bergeles, C., Xu, K., & Axinte, D. A. (2023). Continuum Robots: An Overview. *Advanced Intelligent Systems*, 5(5), Article 5. <https://doi.org/10.1002/aisy.202200367>

Schimansky, T. (n.d.). *CustomTkinter*. Retrieved August 23, 2024, from <https://customtkinter.tomschimansky.com/>

Siles, I., & Walker, I. D. (2009). Design, construction, and testing of a new class of mobile robots for cave exploration. *2009 IEEE International Conference on Mechatronics*, 1–6. <https://doi.org/10.1109/ICMECH.2009.4957126>

Sun, C., Chen, L., Liu, J., Dai, J., & Kang, R. (2019). A hybrid continuum robot based on pneumatic muscles with embedded elastic rods. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 234, 095440621882201. <https://doi.org/10.1177/0954406218822013>

Tekinalp, A., Kim, S. H., Bhosale, Y., Parthasarathy, T., Naughton, N., Albaizroun, A., Joon, R., Cui, S., Nasiriziba, I., Stölzle, M., Shih, C.-H., & Gazzola, M. (2024). *GazzolaLab/PyElastica: V0.3.2 (Version v0.3.2) [Python]*. <https://doi.org/10.5281/zenodo.10883271>

The Matplotlib development team. (n.d.). *Matplotlib documentation*. Retrieved August 21, 2024, from <https://matplotlib.org/stable/>

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., & SciPy, C. (2020). *Author Correction: SciPy 1.0: fundamental algorithms for scientific computing in Python* (*Nature Methods*, (2020), 17, 3, (261-272), 10.1038/s41592-019-0686-2).

Wang, X., Li, Y., & Kwok, K.-W. (2021). A Survey for Machine Learning-Based Control of Continuum Robots. *Frontiers in Robotics and AI*, 8. <https://doi.org/10.3389/frobt.2021.730330>

Zhang, J., Fang, Q., Xiang, P., Sun, D., Xue, Y., Jin, R., Qiu, K., Xiong, R., Wang, Y., & Lu, H. (2022). A Survey on Design, Actuation, Modeling, and Control of Continuum Robot. *Cyborg and Bionic Systems*, 2022, 9754697. <https://doi.org/10.34133/2022/9754697>

- Zhang, X., Chan, F. K., Parthasarathy, T., & Gazzola, M. (2019). Modeling and simulation of complex dynamic musculoskeletal architectures. *Nature Communications*, 10(1), Article 1. <https://doi.org/10.1038/s41467-019-12759-5>
- Zhong, Y., Hu, L., & Xu, Y. (2020). Recent Advances in Design and Actuation of Continuum Robots for Medical Applications. *Actuators*, 9(4), Article 4. <https://doi.org/10.3390/act9040142>
- Zulko. (n.d.). *MoviePy 1.0.2 documentation*. Retrieved August 21, 2024, from <https://zulko.github.io/moviepy/index.html#>