# ESCUELA POLITÉCNICA NACIONAL

## FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

# UTILIZACIÓN DEL MIDDLEWARE OROCOS EN UN SISTEMA EMBEBIDO DE PLACA REDUCIDA PARA IMPLEMENTAR UN CONTROLADOR PID DE POSICIÓN EN UN ROBOT PARALELO TIPO DELTA SIMULADO EN VREP

**TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE MAGISTER EN AUTOMATIZACIÓN Y CONTROL ELECTRÓNICO INDUSTRIAL**

**CARLOS ALBERTO CAPILLA FALCÓN**

**DIRECTOR: ANA VERÓNICA RODAS BENALCÁZAR**

**Quito, junio 2023**

# AVAL

Certifico que el presente trabajo fue desarrollado por Carlos Alberto Capilla Falcón, bajo mi supervisión.

_____

**ANA VERÓNICA RODAS BENALCAZAR**
**DIRECTORA DEL TRABAJO DE TITULACIÓN**

# DECLARACIÓN DE AUTORÍA

Yo, Carlos Alberto Capilla Falcón, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

_____

CARLOS ALBERTO CAPILLA FALCÓN

# DEDICATORIA

Dedico mi logro académico primero a Dios por haberme brindado salud, vida y sabiduría durante esta trayectoria.

También dedico este triunfo a mis padres Willam Capilla, Rosita Falcón, y mi hermano Santi Capilla por ser el pilar fundamental en mi vida y brindarme su apoyo.

A mi pareja sentimental Pame gracias por brindarme su apoyo incondicional y permanecer siempre conmigo.

Adicional dedicó a todas las personas y compañeros de este noble establecimiento que han formado parte a lo largo de mi trayectoria académica

*Carlos Capilla*

# AGRADECIMIENTO

Agradezco primeramente a Dios, a mi familia y a mis amigos por esta siempre para ayudarme cuando los necesite.

Agradezco a la Ing. Ana Rodas, por su apoyo durante mi trayectoria académica en la Escuela Politécnica Nacional y en el desarrollo del proyecto de titulación, que me ha servido para poder cerrar este ciclo de estudio.

Agradezco de todo corazón las enseñanzas brindadas por todos y cada uno de mis profesores a lo largo de mi trayectoria en esta noble institución

A mi pareja sentimental gracias por brindarme su apoyo incondicional y permanecer siempre conmigo.

Agradezco a todas las personas y compañeros de este noble establecimiento que han formado parte a lo largo de mi trayectoria académica.

*Carlos Capilla*

# ÍNDICE DE CONTENIDO

# RESUMEN

El presente trabajo de titulación describe la utilización del middleware Orocos en la implementación de un controlador PID de posición para un robot paralelo tipo delta simulado en VREP utilizando un sistema embebido de placa reducida. Para ello, primero se seleccionó una tarjeta de desarrollo embebida que cumpla con los requerimientos de hardware y adecuada para correr el sistema operativo UBUNTU MATE, en el cual se instaló y ejecutó el paquete OROCOS TOOLCHAIN, que permitió crear componentes y ejecutar algoritmos. Posteriormente se diseñó el controlador en base al modelo dinámico del robot paralelo, tomando en consideración las condiciones de funcionamiento y la estrategia de control a aplicar. A continuación, se implementó la escena del robot paralelo en el simulador VREP el cual está gobernado por el controlador desarrollado en el lenguaje de programación PYTHON conjuntamente con el paquete OROCOS que permitirá su ejecución. Finalmente se estableció la conexión entre la tarjeta embebida y el simulador en el computador personal donde se encuentra el ambiente simulado en VREP del robot paralelo, para comprobar la operación del sistema.



**PALABRAS CLAVE:** Orocos, VREP, Robot paralelo, Python.

# ABSTRACT

The present degree work describes the use of the Orocos middleware in the implementation of a PID position controller for a delta type parallel robot simulated in VREP using a reduced plate embedded system. For it, first an embedded development card was selected that meets the hardware requirements and suitable to run the UBUNTU MATE operating system, in which the OROCOS TOOLCHAIN package was installed and executed, this allowed the creation of components and execution of algorithms. Subsequently, the controller was designed based on the dynamic model of the parallel robot, considering the operating conditions and the control strategy to be applied. Next, the parallel robot scene was implemented in the VREP simulator, which is governed by the controller developed in the PYTHON programming language together with the OROCOS package that will allow its execution. Finally, the connection between the embedded card and the simulator was established in the personal computer where the simulated environment in VREP of the parallel robot is located, to verify the operation of the system.



**KEYWORDS:** Orocos, V-REP, Parallel robot, Python.

# 1 INTRODUCCIÓN

El control automático de los sistemas robotizados actuales involucra cada vez más la implementación de distintas tareas a realizar por el robot con distinto grado de complejidad, de naturaleza periódica o aperiódica, ejecutadas de manera local o distribuida a través de una red de comunicaciones y para lo cual es necesario manejar hardware de distinta naturaleza. Abordar la implementación de controladores para una nueva plataforma robotizada supondría volver a desarrollar código adecuado para el nuevo hardware o en el mejor de los casos adaptar el ya existente. En la última década ha habido un interés creciente en los sistemas de software basados en componentes que faciliten el trabajo a partir del desarrollo de componentes reusables y que puedan interoperar fácilmente entre ellos [1].

Desde el año 2001 inicio el proyecto denominado OROCOS ("Open Robot Control Software", que en un contexto no robótico significa, "Servicios de control en tiempo real abiertos", cuyo objetivo fue convertirse en un paquete de programas de control abiertos para aplicaciones robóticas de uso general. Existen pocos proyectos desarrollados utilizando los paquetes de programación OROCOS, representando una gran desventaja para los usuarios que incursionan en este tema por lo que el presente trabajo actuará como guía para futuros proyectos.

Para el presente trabajo se utilizó el middleware OROCOS con la finalidad de demostrar que es factible la implementación de un controlador de posicionamiento para un robot paralelo utilizando la integración del middleware OROCOS al sistema operativo robótico ROS en un sistema embebido de placa reducida; además de que es posible utilizar el programa de simulación virtual libre VREP para crear el robot paralelo que será un nodo en el sistema ROS. En este sentido a futuro, se podrá establecer un laboratorio virtual para realizar pruebas de control en robots paralelos para proyectos que requieran ejecución de procesos en tiempo real.

## 1.1 Objetivo General

Utilización del middleware OROCOS en un sistema embebido de placa reducida para implementar un controlador PID de posición en un robot paralelo tipo delta simulado en VREP.

## 1.2 Objetivos Específicos

- Revisar bibliografía de artículos científicos referenciales para determinar las características necesarias en el sistema embebido de placa reducida y la operación del middleware OROCOS.
- Implementar el robot paralelo en el simulador de robots V-REP y utilizar la interfaz de programación de aplicaciones API para interactuar con el sistema ROS.
- Utilizar el paquete de integración de OROCOS para desarrollar un controlador PID de posición para el robot paralelo y ejecutarlo en el meta sistema operativo de código abierto ROS en línea.
- Utilizar el ROS Toolbox de Matlab para desarrollar un controlador similar al implementado usando OROCOS.
- Enlazar el sistema de simulación con los controladores desarrollados para comparar su desempeño utilizando en ambos casos el robot paralelo simulado en V-REP.

## 1.3 Alcance

El trabajo de titulación se plantea como un estudio teórico-práctico para la implementación del controlador de posicionamiento PID de un robot paralelo simulado utilizando la integración del middleware OROCOS con ROS (sistema operativo robótico) en un sistema embebido de placa reducida, donde se creará una aplicación de control en línea, integrando al nodo del sistema ROS constituido por el robot paralelo implementado en el simulador V-REP. Así mismo, el controlador del robot se probará posicionando la plataforma móvil utilizando varias secuencias de puntos generados en el espacio de trabajo. Cabe destacar que el trabajo propuesto se enfocará en la utilización de un meta sistema de código abierto para ejecución del controlador integrado a un ambiente simulado.

Al mismo tiempo, con el fin de verificar las prestaciones del middleware OROCOS en aplicaciones robóticas virtuales, se comparará el desempeño del controlador desarrollado en OROCOS con el del controlador similar desarrollado en Matlab empleando su ROS Toolbox. En ambos casos se empleará el robot paralelo simulado en VREP.

Finalmente, se tendrá como producto final demostrable los códigos de programa desarrollados en los lenguajes de programación Python, LUA, C++ y la escena con el modelo del robot paralelo desarrollado en el simulador V-REP.

## 1.4 Marco Teórico

La complejidad actual de los sistemas robotizados y de las aplicaciones que estos deben realizar requiere que los robots dispongan de un control automático que permita la ejecución de las distintas tareas que forman parte del algoritmo de control y que tenga en cuenta cuestiones relacionadas por ejemplo con la periodicidad, el modo de ejecución, el hardware que se utilizaría, etc. [1]. Para el desarrollo de este tipo de aplicaciones de control en los últimos años se tiende a la programación basada en componentes puesto que esta permite obtener código reusable. Así mismo también se está incrementando la utilización de middlewares que permiten la abstracción de los sistemas operativos, el soporte de tiempo real y la infraestructura de comunicaciones. [2]

Uno de estos middlewares, orientados especialmente a la robótica se denomina OROCOS (Open Robot Control Software), con funciones como soporte en línea, servicios middleware entre otras, que tiene desarrolladas una serie de componentes que permiten realizar distintos algoritmos para control de robots y se lo ha venido utilizando desde el año 2001[1].

Entre los middlewares basados en componentes, OROCOS destaca el tener soporte para aplicaciones en tiempo real, debido a la herramienta RTT (Real Time Toolkit) de la librería Orocos ToolChain siendo una de las más recientes del proyecto [2]. Del mismo modo en OROCOS las aplicaciones se han identificado y caracterizado los tipos de componentes que participan en las tareas de robots manipuladores siendo estos: los sensores, actuadores y algoritmos de procesamiento como generadores de trayectorias. [3]

La implementación de aplicaciones robóticas permitió la introducción de computadoras de bajo costo y alta potencia denominadas Raspberry Pi, que por su versatilidad son herramientas útiles no solo exclusivamente para control electrónico sino en aplicaciones que requieran interconectividad e inteligencia colectiva. [4]

En este sentido, el presente proyecto pretende desarrollar aplicaciones robóticas utilizando software de código abierto para facilitar su utilización en proyecto futuros y demostrar las ventajas del software, para lo cual hará uso de un robot paralelo simulado con varias secuencias de posiciones preestablecidas.

### 1.4.1 Software

En esta sección se explicará los softwares que se emplearon para desarrollar este proyecto, middleware de robótica, frameworks y herramientas que ayudaron a construir las aplicaciones robóticas.

### 1.4.1.1 Sistema Operativo

El Sistema Operativo (SO) a utilizar es Ubuntu 16.4.2 basado en Linux y se descartó utilizar Windows debido a que no es un SO libre ni gratuito. Además, Linux es la opción más popular para realizar el control mediante tarjetas embebidas, debido a la versatilidad y poseer código abierto [5].

En [6] se explica la comparación de los principales atributos de middleware de robótica (Tabla 1.1.) en donde se puede evidenciar que ROS y OROCOS manejan conceptos similares facilitando de esta manera la integración entre estos middlewares, cuya instalación se realizará sobre un sistema operativo basado en Linux.

**Tabla 1.1.** Atributos ROS y Orocos trabajando en Ubuntu [6]

| Concepto | Atributos Orocos | Atributos Ros |
|---|---|---|
| Comunicación de flujo de datos | Puertos de datos | Temas |
| Terminal de datos de salida | Puerto de salida | Editor |
| Terminal de datos de entrada | Puerto de entrada | Suscriptor |
| Servicio | Operación | Servicio |
| Acción | Acción | Acción |
| Estructura de datos | Juego de tipos | Mensaje |
| Recopilación de datos | Propiedad, Atributo, Constante | Parámetro |
| Unidad de computación | Proceso OROCOS, Componente | Nodo |

### 1.4.1.2 Middleware OROCOS

El middleware de robótica es una capa de abstracción que reside entre el sistema operativo y las aplicaciones de software, permitiendo simplificar el diseño del software y reducir los costos de desarrollo. En este sentido, al estar diseñado para administrar la heterogeneidad del hardware permite mejorar la calidad de las aplicaciones de software. Además, el middleware al ser un sistema basado en componentes puede combinarse o interrelacionarse con otros componentes existentes, y además tiene la flexibilidad para modificar o reemplazar estos [6]. Las principales ventajas del middleware de robótica son:

- Modularidad del software.
- Abstracción de la arquitectura del hardware.
- Independencia de la plataforma.
- Portabilidad.

Por lo tanto, Orocos actualmente se ha convertido en un proyecto middleware importante que dispone de un conjunto de herramientas para el desarrollo de software de robótica. Ahora bien, las partes principales están constituidas por la cadena de herramienta en tiempo real (RTT) y la biblioteca de componentes de Orocos (OCL). [7]

En el proyecto Orocos también se desarrollaron bibliotecas adicionales para complementar el paquete para control avanzado de máquinas y robots como se puede observar en la Figura 1.1.

Estas bibliotecas incluyen:

- Cálculo de cadenas cinemáticas y dinámicas.
- Filtrado.
- Especificación de tareas avanzadas.

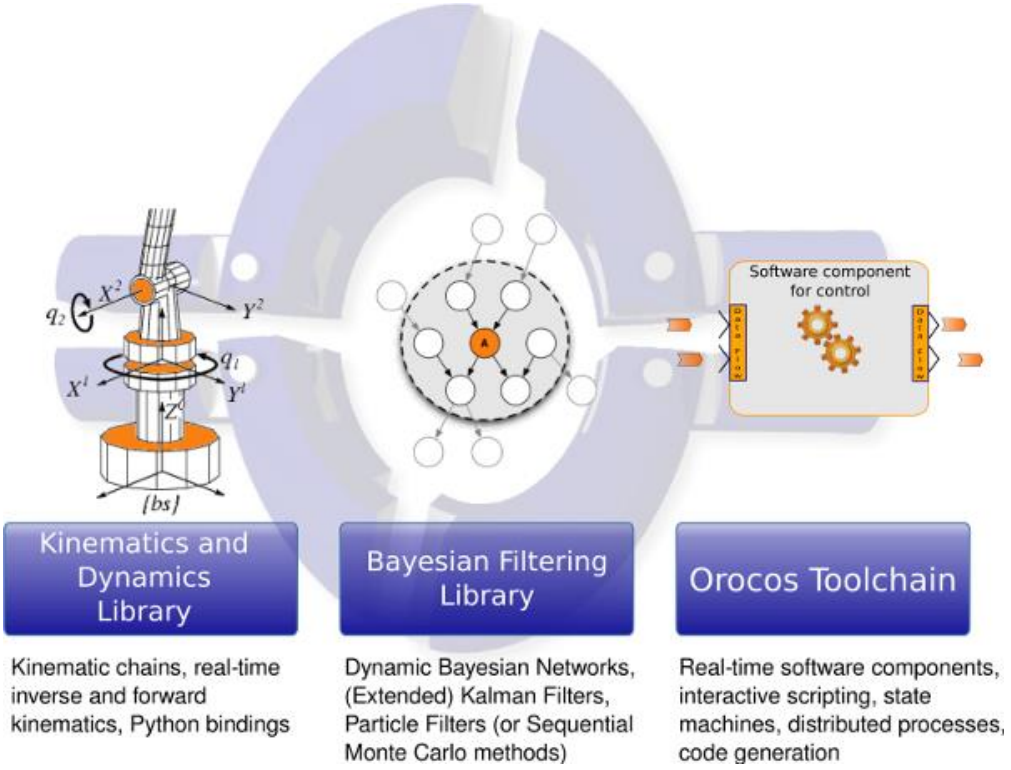


**Figura 1.1.** Librerías proyecto Orocos. [7]

#### 1.4.1.2.1 Orocos Toolchain

Orocos Toolchain se constituye como la librería principal para crear aplicaciones robóticas en tiempo real utilizando componentes de software modulares [2].

Esta librería proporciona:

- Soporte multiplataforma: Linux, Windows (Visual Studio) y Mac OS-X.
- Extensiones a otros marcos de robótica: ROS, Rock, Yarp.
- Generadores de código para transferir datos definidos por el usuario entre componentes distribuidos.
- Componentes configurables y programables en tiempo de ejecución y en tiempo real.
- Registro y reporte de eventos del sistema y datos comunicados.

Incluye las siguientes herramientas:

- **AutoProj**, una herramienta para descargar y compilar las bibliotecas necesarias.
- **Real-Time Toolkit**, un marco de componentes que permite escribir componentes en tiempo real utilizando el lenguaje de programación C ++.
- **La biblioteca de componentes de Orocos**, los componentes necesarios para iniciar una aplicación e interactuar con ella en tiempo real.
- **OroGen y TypeGen**, herramientas para generar código preparado para compilar y ejecutar a partir de cabeceras existentes o ficheros de descripción de componentes

Esta librería permite la creación de los componentes utilizados en este proyecto. Además, tiene un soporte multi plataforma debido a que se puede trabajar tanto en Linux como en Windows o MAC y en ciertos casos sin utilizar la parte en tiempo real [2].

#### 1.4.1.2.2 Componentes en OROCOS

Un componente en el middleware OROCOS, permite ejecutar uno o más programas en un hilo según el mecanismo de ejecución, cuya estructura puede apreciarse en la Figura 1.2.

Estos componente está definido por:

- Puertos.
- Operaciones.
- Propiedades.
- Servicios.

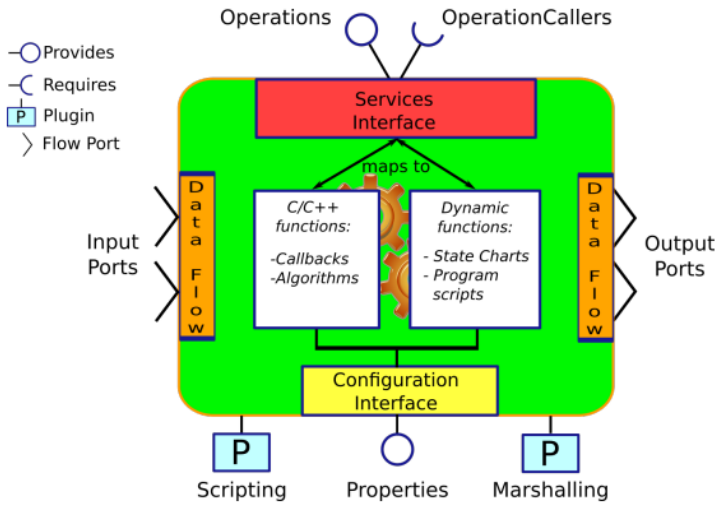


**Figura 1.2.** Estructura de un componente en OROCOS. [7]

Otra ventaja es la facilidad de comunicación de manera transparente entre componentes de tiempo de real RTT y componentes que no lo son, constituyen un aspecto importante en el proyecto OROCOS. En tal sentido, este sistema está libre de prioridades y todas las operaciones son no bloqueantes, incluido la transmisión e intercambio de datos. [7]

#### 1.4.1.2.3 Aplicaciones de OROCOS

El middleware OROCOS ha sido tendencia en lo referente a aplicaciones relacionadas al control de robots, proporcionando la abstracción del sistema operativo, el soporte en tiempo real, la infraestructura de comunicaciones y un modelo basado en componentes [1]. Del mismo modo, mediante el middleware OROCOS se han implementado diversos controladores, entre estos se puede mencionar los siguientes:

- Desarrollo de una serie de componentes correspondientes a distintos algoritmos para el control dinámico de robots paralelos de 3 DOF con la utilización de middlewares que permiten la abstracción de los sistemas operativos como lo desarrollaron; Marina Valles, Jose Cazalilla, Ángel Valera, Vicente Mata y Álvaro Page de la Universidad Politécnica de Valencia [4] y [2].
- El sistema de control adaptativo de robots mediante la ingeniería del software basada en componentes [8], donde J. Cazalilla utilizando técnicas relacionadas con la ingeniería de software basada en componentes, se desarrolló de forma modular una variedad de controladores avanzados para un robot paralelo de 3 grados de libertad.
- R. Ceballos, R. Rodriguez, B. Paredes, y D. Vargas desarrolló de un robot de rehabilitación pasiva para la articulación de la muñeca [9], donde utilizando un

microcontrolador Arduino UNO R3 se controla los movimientos de un prototipo funcional robótico desarrollado para realizar ejercicios de rehabilitación en la articulación de la muñeca.

#### 1.4.1.2.4 Limitaciones y alcances del middleware Orocos

El middleware de manera genérica representa la unión de los sistemas de computación distribuida basados en RCP (llamada a procedimiento remoto) y la programación orientada a objetos. Entre sus alcances pueden destacar [10]:

- La capacidad de integrar múltiples ordenadores, para de esta manera operar como un recurso computacional unificado.
- Baja complejidad para creación de frameworks reutilizables.
- Crear servicios distribuidos y aplicaciones eficientes, adaptables y robustas.

No obstante, a pesar del rendimiento, versatilidad y capacidad, el middleware presenta varias limitaciones:

- Carece de limites funcionales.
- Falta de estándares de gestión de software.
- Carece de implementación de software.

### 1.4.1.3 ROS

Ros (Robotic Operating System) es un framework open source para el desarrollo de software de robots, proveyendo servicios de abstracción hardware, control de dispositivos a bajo nivel, mensajes entre procesos, control de paquetes e implementación de las librerías más utilizadas. Es por ello por lo que se le conoce como "meta-sistema operativo" a pesar de que realmente no lo es; además se lo conoce como Robotics Middleware (Middleware de robótica) [6]. No obstante, ROS no es un sistema operativo debido a que funciona sobre Linux.

Este sistema fue desarrollado en 2007 originalmente por la Universidad de Stanford, desde 2008 se mantiene gracias a la colaboración de más de 20 instituciones federadas bajo el instituto Willow Garage [5]. ROS está licenciado bajo BSD, lo que permite plena libertad para uso comercial e investigador. Por lo tanto, ROS está diseñado para facilitar la escritura de software robótico tanto para la industria como para la investigación, siendo sus principales objetivos los siguientes:

- Soportar la programación de diferentes hosts unidos en una topología punto a punto, lo que permite aprovechar los beneficios del diseño multi-proceso y multi-host. Gracias a ello, es posible diseñar un sistema de múltiples computadores conectados vía ethernet.
- Multilenguaje. Dado que hay programadores con distintas preferencias en cuanto al lenguaje de programación, soporta C++, Python, Octave, Lisp y otros.
- ROS provee de un mecanismo de comunicaciones (middleware) distribuido entre nodos del sistema robótico.
- Basado en herramienta. Ros es muy complejo, así que se ha optado por una arquitectura de microkernel con muchas pequeñas herramientas en lugar de un diseño monolítico y grande. De esta forma, el código puede ser reaprovechado, es modular, fácilmente estructurable y acorde a la topología punto a punto.
- Liviano. Ros tiene una gran variedad de librerías que pueden utilizarse para un proyecto (o no) de forma que se cree un ejecutable bastante ligero.
- Gratuito y Opensource. Está distribuido bajo Ucencia BSD.

ROS en este proyecto se usa específicamente para realizar control en Real Time utilizando el middleware OROCOS.

#### 1.4.1.3.1 Integración de Orocos con ROS

El framework Orocos tiene la capacidad de integración con el sistema operativo robótico ROS. De hecho, la integración ROS de Orocos es una colección de paquetes que proporciona una capa de traducción entre los dos frameworks, para facilitar al usuario la transmisión de datos y enlace de uno a otro. [7]

Así mismo, en este proyecto para integrar los componentes de Orocos y el sistema operativo robótico ROS en el sistema de placa embebida se utilizó el paquete rtt_ros_integration, permitiendo de esta manera que los componentes puedan publicar y suscribir los tópicos de ROS necesarios para ejecutar el controlador del robot paralelo delta. Esta integración ayudará a desarrollar el sistema de control basado en ROS y OROCOS.

#### 1.4.1.3.2 ROS Toolbox en Matlab

El ROS Toolbox proporciona una interfaz que conecta MATLAB® y Simulink® con el sistema operativo del robot (ROS y ROS 2) [11], permitiendo crear una red de nodos en el sistema operativo robótico ROS. Igualmente, la caja de herramientas incluye funciones de

MATLAB y bloques de Simulink para importar, analizar y reproducir datos ROS registrados en archivos rosbag. Además, puede conectarse a una red ROS en línea para acceder a los mensajes ROS.

También, dispone de herramientas que permiten verificar los nodos ROS a través de la simulación de escritorio y al conectarse a simuladores de robots externos como Gazebo o VREP. De hecho, Toolbox de Matlab permitirá establecer la comunicación y el adecuado intercambio de datos entre algoritmo del control de posición del robot Delta implementado en MatLAB y el sistema ROS, para de esta manera comparar con el controlador implementado en la tarjeta de desarrollo embebido.

#### 1.4.1.4 Robot Paralelo

Un robot paralelo consiste básicamente en una plataforma móvil que se conecta a una plataforma o base fija por medio de cadenas cinemáticas. El efector final está conectado a la plataforma móvil, para ejecutar varias aplicaciones. Se denominan paralelos en el sentido topológico debido a que las plataformas al estar conectado mediante cadenas cinemáticas de iguales características, los miembros del robot operan juntos, y no se encuentran alineados en forma de líneas paralelas de manera trigonométrica. [1]

HAL open science es un archivo open access multidisciplinario de documentos investigativos del institutos de Francia, en una recolección de información de los robots desarrollados, dan a conocer que los robots paralelos tienen mayor aplicación en el sector industrial; 1) operaciones pick and place (recogida y descarga), 2) centros de mecanizados para altas velocidades. En el sector de la robótica de servicios: 3) la cirugía robótica y 4) desarrollo de dispositivos de rehabilitación y diagnosis [12].

**Ventajas del Robot Paralelo**

Este tipo de robot por su tipología dispone de una serie de ventajas, detalladas a continuación:

- Alta rigidez y robustez.
- Alta capacidad de mover o transportar cargas en su espacio de trabajo.
- Alta velocidad y precisión.
- Facilidad de simular modelos, al tener cinemática inversa sencilla de calcular.

**Desventajas del Robot Paralelo**

Por otro lado, entre sus limitaciones se tienen:

- Espacio de trabajo reducido.
- Problemas de singularidad.
- Complejidad al planificar su control.

El uso de estos robots varía desde 3 hasta 6 GDL según la aplicación, para este proyecto se elige un robot paralelo delta de tres grados de libertad para experimentar el sistema de control propuesto en la tesis, cuyo controlador desarrollado en el middleware Orocos será implementado en la tarjeta de desarrollo.

#### 1.4.1.5 Tarjeta embebida de desarrollo

Para determinar la tarjeta embebida de desarrollo en el cual se implementará el middleware Orocos, se realiza un análisis de las características técnicas de las tarjetas, con la finalidad de determinar una tarjeta con las prestaciones necesarias para ejecutar el middleware y el meta-sistema operativo ROS.

A continuación, se detallarán las tarjetas embebidas existentes:

##### 1.4.1.5.1 Raspberry Pi

Raspberry Pi es un ordenador de placa reducida que tiene todas la funcionalidades y capacidad de interactuar con periféricos como un ordenador convencional. Esta tarjeta de desarrollo opera con un sistema operativo de código abierto, siendo su sistema oficial una versión adaptada de Debian, denominada Raspberry Pi OS, sin embargo, puede utilizar otros sistemas operativos. Para todas sus versiones incluye un microprocesador Broadcom, memoria RAM, GPU, puertos USB, puertos HDMI, Ethernet, 40 pines GPIO y un conector para cámara [13]

Esta tarjeta ha sido utilizada en diversos proyectos electrónicos enfocados principalmente a la enseñanza informática. En ese mismo contexto se ha venido utilizando en proyectos robóticos. Así mismo, esta tarjeta es una alternativa como ordenador de uso general de potencia baja y consumo energético reducido, cuya limitante es aplicaciones que demanden altos recursos de procesamiento.

En la Tabla 1.2 se detalla un análisis entre algunas tarjetas de desarrollo Raspberry disponibles en el mercado:

**Tabla 1.2.** Tarjetas de desarrollo embebidas Raspberry Pi [13].

| Especificación | Raspberry Pi 3B | Raspberry Pi 4B |
|---|---|---|
| CPU | 1.2 GHz Broadcom Cortex-A53 | 1.4 GHz Broadcom Cortex-A53 |
| GPU | VideoCore IV | VideoCore IV |
| Memoria | 1024 MB | 2/4/8 GB |
| USB | 4 | 2 x USB 3.0<br>2 x USB 2.0 |
| Salida video | HDMI | 2 x micro HDMI |
| Salida audio | HDMI | 2 x micro HDMI |
| Sistema operativo | Archlinux ARM<br>LibreELEC<br>OpenELEC<br>Pidora<br>Raspbmc<br>RISC OS<br>Raspbian<br>Ubuntu | Archlinux ARM<br>LibreELEC<br>OpenELEC<br>Pidora<br>Raspbmc<br>RISC OS<br>Raspbian<br>Ubuntu |
| Compatibilidad ROS | Si | Si |

Raspberry Pi es una buena opción como computadora de escritorio de uso general y de baja potencia, sin embargo, no podrá alcanzar el mismo nivel de rendimiento que una computadora de escritorio o una computadora portátil estándar. De la misma manera, su consumo de energía es bajo y respetuoso con el medio ambiente lo que ayuda a compensar cualquier problema con el rendimiento ocasionalmente lento [5].

#### 1.4.1.5.2 BeagleBone

La Beaglebone ha sido desarrollada por la BeagleBoard Foundation, de origen estadounidense, cuyo objetivo es promocionar el uso de software y hardware open-source para el desarrollo de sistemas empotrados. En la Figura 1.3 se puede observar la estructura de tarjeta de desarrollo BeagleBoneBlack.


**Figura 1.3.** Tarjeta de desarrollo BeagleBoneBlack [14].

La célula de BeagleBoard se puede encontrar en la empresa Texas Instruments, patrocinadora de la fundación dado su interés de crear dispositivos empotrados de carácter abierto y con distintas posibilidades. El objetivo de BeagleBoard es crear "dispositivos de bajo coste, basados en procesadores de bajo consumo de Texas Instruments, utilizando núcleos ARM Cortex-A" con distintas posibilidades [14].

En la Tabla 1.3 se detallan los modelos disponibles.

**Tabla 1.3.** Tarjetas de desarrollo embebidas Beaglebone [5]

| | **Modelo Principal** | **Modelo Black** | **Modelo xM** |
|---|---|---|---|
| SoC | TI AM3358 | TI AM3359 | TI DM3730 |
| CPU | 720 MHz ARM Cortex-8 | 1 GHz ARM Cortex-8 | 1 GHz ARM Cortex-8 2 |
| GPU | Power VR SGX530 | Power VR SGX530 | Unspecified PowerVR SGX |
| Memoria | 256 MB DDR2 | 512 MB DDR3 | 512 MB DDR2 |
| USB | 1 | 1 | 2 |
| Salida video | micro HDMI | micro HDMI | DVI, S-Video |
| Salida audio | micro HDMI | micro HDMI | 3.5 mm jack |
| Periféricos | 4x UART, 8x PWM, LCD, GPMC, MMC1, 2x SPI, 2x I2C, A/D Converter, 2xCAN Bus, 4 Timers, FTDI USB to Serial, JTAG via USB | 4x UART, 8x PWM, LCD, GPMC, MMC1, 2x SPI, 2x I2C, A/D Converter, 2xCAN Bus, 4 Timers | McBSP, DSS, I2C, UART, LCD, McSPI, PWM, JTAG, Camera Interface |

| | | | |
|---|---|---|---|
| Sistema | Debian<br>Android<br>Ubuntu<br>Cloud9 IDE | Debian<br>Android<br>Ubuntu<br>Cloud9 IDE | Debian<br>Android<br>Ubuntu<br>Cloud9 IDE |

### 1.4.1.5.3 Selección de Tarjeta

A continuación, se detallan las características técnicas de las tarjetas de desarrollo elegibles para el presente trabajo.

**Tabla 1.4.** Tarjetas embebidas de desarrollo.

| **Especificación** | **Raspberry Pi 3B** | **Raspberry Pi 4B** | **Beaglebone Black** |
|---|---|---|---|
| CPU | 1.2 GHz<br>Broadcom<br>Cortex-A53 | 1.4 GHz<br>Broadcom<br>Cortex-A53 | 1 GHz ARM<br>Cortex-8 |
| GPU | VideoCore IV | VideoCore IV | Power VR<br>SGX530 |
| Memoria | 1024 MB | 2/4/8 GB | 512 MB DDR3 |
| USB | 4 | 2 x USB 3.0<br>2 x USB 2.0 | 1 |
| Salida video | HDMI | 2 x micro HDMI | micro HDMI |
| Salida audio | HDMI | 2 x micro HDMI | micro HDMI |
| Sistema operativo | Archlinux ARM<br>LibreELEC<br>OpenELEC<br>Pidora<br>Raspbmc<br>RISC OS<br>Raspbian<br>Ubuntu | Archlinux ARM<br>LibreELEC<br>OpenELEC<br>Pidora<br>Raspbmc<br>RISC OS<br>Raspbian<br>Ubuntu | Debian<br>Android<br>Ubuntu<br>Cloud9 IDE |
| Compatibilidad ROS | Si | Si | Si |
| Compatibilidad middleware OROCOS | Si | No | No |

Se ha intentado instalar en las tarjetas de desarrollo mencionadas el sistema ROS y la integración del middleware OROCOS utilizando la documentación disponible al ser un sistema de código de abierto o libre, sin embargo, únicamente la tarjeta de desarrollo Raspberry Pi modelo 3B permitió la instalación y ejecución del módulo de integración del middleware Orocos con el sistema operativo ROS, razón por lo cual se ha seleccionado como tarjeta de desarrollo para ejecutar el sistema de control.

# 2 METODOLOGÍA

En este capítulo se desarrollará la metodología a implementarse en el proyecto, desde el análisis cinemático del robot a simularse, el controlador PID con su sintonización y el enlace de comunicación con el sistema operativo ROS y el middleware OROCOS.

La metodología utilizada para el presente trabajo se puede observar en la Figura 2.1:

- Inicialmente se determinó la tarjeta embebida de desarrollo y el sistema operativo para la operación de Orocos basado en la revisión bibliográfica de artículos referenciales.
- Se realiza el estudio del análisis cinemático del robot delta, para tal efecto, los parámetros físicos de cada componente del robot serán tomados del diseño mecánico elaborado en SOLIDWORKS con la finalidad de implementar el Robot simulado en V-REP.
- Se diseña e implementa el controlador PID de posición para cada articulación revoluta a implementarse; tanto en la tarjeta embebida de desarrollo quien contiene el sistema operativo Ubuntu y ROS con OROCOS integrado, como en Matlab con ROS Toolbox implementado en el ordenador.
- Se realiza la comunicación entre robot delta y el controlador utilizando nodos, considerando como nodo Máster al sistema donde se implementará la simulación del robot en V-REP y nodos Esclavos a los controladores tanto en la placa embebida como el ordenador.
- Se simula el control de posición del robot delta utilizando el programa V-REP, quien a su vez se comunica con la red ROS.
- Se realiza la etapa de pruebas y análisis de resultados del controlador PID implementado en el ordenador con MatLAB y en la tarjeta de desarrollo embebida con el middleware Orocos.


**Figura 2.1.** Metodología.
Fuente: Propia.

## 2.1 Análisis cinemático del robot paralelo 3PRS

### 2.1.1 Geometría del robot

Para obtener el modelo matemático que describa la cinemática del robot se debe analizar su geometría. El robot delta está conformado por: (1) una base fija, donde se ubican tres actuadores ($R_1, R_2\ y\ R_3$) equidistantes del centro de la base y separados 120° entre sí; y, de (2) una base móvil conectada a la base fija mediante tres cadenas cinemáticas, cada una compuesta por dos eslabones y conectados entre sí por medio de juntas universales (U) (Ver Figura 2.2).

El sistema de referencia $XYZ$ se ubica en el centro de masa de la base fija, colocando el eje $X$ perpendicular al eje del actuador $R_1$ y el eje $Z$ perpendicular a la base fija.

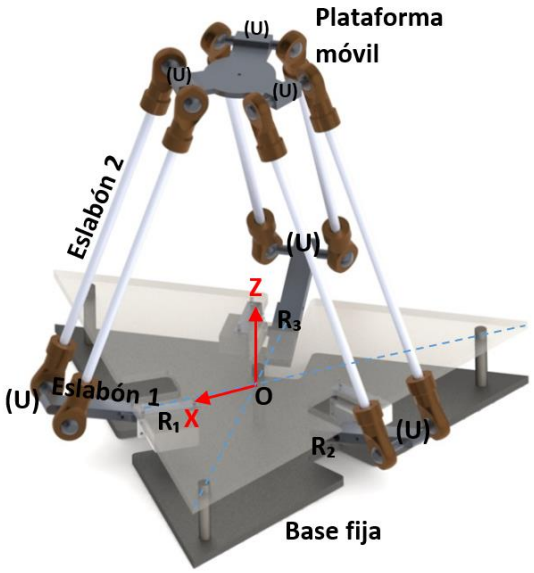


**Figura 2.2.** Estructura del robot paralelo.
Fuente: Propia.

### 2.1.2 Características generales

La condición de paralelismo del robot delta se atribuye a que la base móvil siempre tiene un movimiento paralelo respecto a la base fija de sí mismo, es decir, tendrá un movimiento de traslación para seguimiento de trayectorias rectas o curvas según el posicionamiento requerido. La figura 2.3. indica el robot delta con sus respectivas partes.

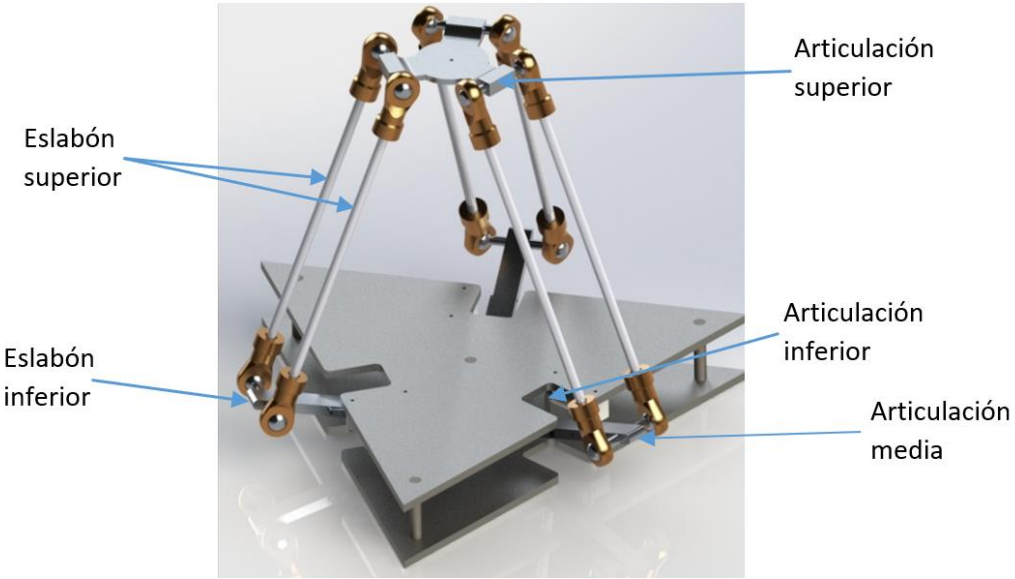


**Figura 2.3.** Articulaciones y eslabones en el robot paralelo de 3GDL invertido.
Fuente: Propia.

### 2.1.3 Parámetros físicos del robot.

La propuesta para el robot paralelo fue desarrollada en el programa para modelamiento mecánico Solidworks (ver Figura 2.3.), tomando en consideración las características particulares de este tipo de robot. En el programa se crearon los diferentes componentes del robot para su posterior ensamble y estudio.

Para este proyecto, se utilizó 30cm para $L_1$y 50cm para $L_2$ dimensiones de los eslabones inferior y superior respectivamente, con el propósito de simular un tamaño que en la actualidad los investigadores proponen en sus diseños[15] [16] [17] .

En la Tabla 2.1 se detallan los parámetros físicos de cada componente del robot, los materiales de construcción del prototipo modelado son aluminio y plástico ABS (acrilonitrilo butadieno estireno).

Los parámetros físicos de cada componente del robot se detallan a continuación:

**Tabla 2.1.** Parámetros físicos del robot delta.
Fuente: Propia.

| | | | | **Masa** | **Masa Total** | | **Ix** | **Iy** | **Iz** |
|---|---|---|---|---|---|---|---|---|---|
| **Item** | **Componente** | **Subcomponente** | **Material** | $gr$. | **Cant.** | $gr$. | $gr.*mm^2$ | | |
| 1 | Eslabón inferior | Acople motor | Aluminio | 2.44 | 3.0 | 7.32 | 7570.56 | 964130.42 | 969868.45 |
| 2 | | Articulación inferior | Aluminio | 54.08 | 3.0 | 162.24 | | | |
| 3 | | Eje de rotula inferior | Aluminio | 19.91 | 3.0 | 59.73 | | | |
| 4 | Eslabón superior | Eje izquierdo | Aluminio | 74.98 | 3.0 | 224.94 | 236720507.89 | 251919055.86 | 252400930.12 |
| 5 | | Rotula izquierda | ABS | 10.21 | 3.0 | 30.63 | | | |
| 6 | | Rotula derecha | ABS | 10.21 | 3.0 | 30.63 | | | |
| 7 | | Eje derecho | Aluminio | 74.98 | 3.0 | 224.94 | 236720507.89 | 251919055.86 | 252400930.12 |
| 8 | | Rotula izquierda | ABS | 10.21 | 3.0 | 30.63 | | | |
| 9 | | Rotula derecha | ABS | 10.21 | 3.0 | 30.63 | | | |
| 10 | Plataforma móvil | Plataforma | ABS | 7.89 | 1.0 | 7.89 | 1430.48 | 1430.49 | 2687.26 |
| 11 | | Eje de rotula superior | Aluminio | 19.91 | 3.0 | 59.73 | | | |
| | | | | **Subtotal** | | 869.31 | | | |
| 12 | Plataforma fija | Plataforma superior | ABS | 1735.48 | 1.0 | 1735.48 | - | - | - |
| 13 | | Plataforma inferior | ABS | 1735.78 | 1.0 | 1735.78 | | | |
| 14 | | Soporte | ABS | 12.51 | 3.0 | 37.53 | | | |

Los parámetros físicos Ix, Iy, Iz corresponden a los momentos de inercia respecto a un eje paralelo, cuyos datos son ingresados en VREP para permitir simular los componentes.

Utilizando los datos de la Tabla 2.1 se calcula la masa total del robot delta exceptuando la plataforma fija cuyo valor es de 869.31 gramos. Así mismo, aplicando un factor de seguridad del 10% se selecciona el actuador que tendrá la capacidad de manipular 956.24 gramos,

En este sentido, para calcular el par motor requerido por el actuador, se toma en consideración la configuración de máximo esfuerzo de la longitud del brazo formado desde el centro de masa del motor hacia el centro de masa de la plataforma móvil (aproximado de 275 mm); y se utiliza la ecuación (2.1):

$$T = F * r \tag{2.1}$$

Donde:

$T$: Torque o par motor (N*m)

$F$: Fuerza (N)

$r$: Radio del brazo (N)

Obteniendo el siguiente resultado:

$$T = 956.24\cancel{gr}.* 9.8\frac{m}{s^2} * 275\cancel{mm} * \frac{1\ kg}{1000\ \cancel{gr}} * \frac{1\ m}{1000\ \cancel{mm}}$$

$$\boxed{T = 2.577\ N * m}$$

El torque o par motor calculado se utilizará para definir las propiedades dinámicas de las articulaciones revolutas del robot paralelo.

### 2.1.4 Configuración geométrica

El robot delta cuenta con tres grados de libertad para posicionar su efector final en el espacio de trabajo. Las cadenas cinemáticas ($i = 1,2,3$), están formadas por dos eslabones (ver Figura 2.4), cuyas dimensiones son $L_1$ y $L_2$ respectivamente.

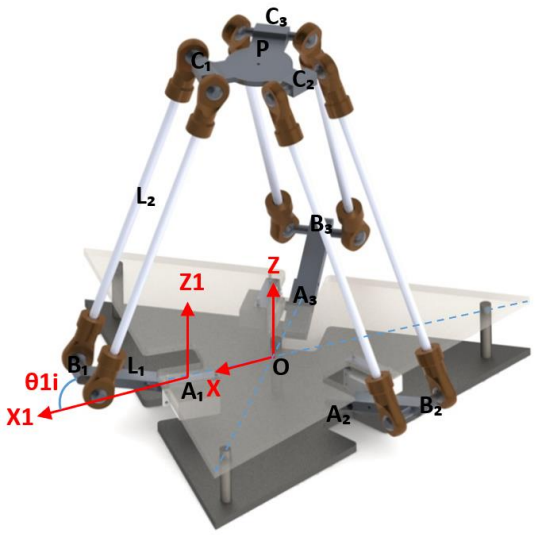


**Figura 2.4.** Modelo geométrico del robot delta.
Fuente: Propia.

En la Figura 2.4 se puede observar que el eslabón $L_1$ se encuentra conectado a la base fija mediante la Articulación $A_i$ y al eslabón $L_2$ por medio de la articulación $B_i$. El eslabón $L_2$ se encuentra conectado a la placa móvil por la articulación $C_i$. Por último, entre el eslabón $L_1$ y $L_2$ se encuentra el ángulo $\beta_i$ donde están colocados los motores. El centro de la plataforma móvil representa la posición del robot $P = [P_x\ P_y\ P_z]^T$.

Donde:

$L_1$ y $L_2$ son eslabones del robot.
$P$ es el centro de la plataforma móvil
$C_i$ son las articulaciones que representan a $C_1$, $C_2$ y $C_3$.
$B_i$ son las articulaciones que representan a $B_1$, $B_2$ y $B_3$.
$A_i$ son las articulaciones conectado al actuador y representan a $A_1$, $A_2$ y $A_3$.
$\theta_{1i}$ es el ángulo generado por el actuador.
$O$ es el centro de la plataforma fija.

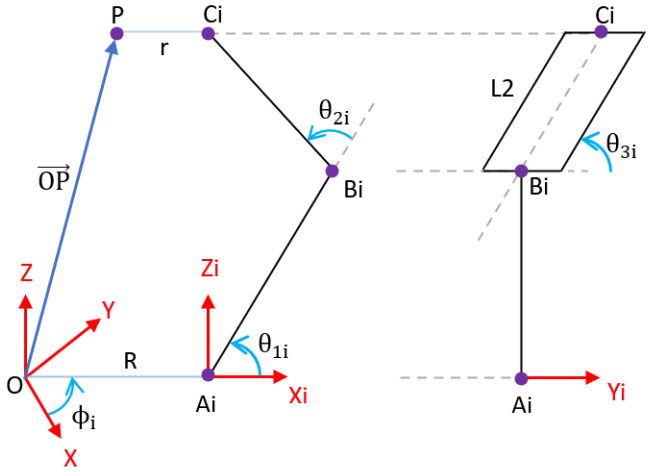


**Figura 2.5.** Descripción de ángulos vista frontal y lateral.
Fuente: Propia.

De la misma manera, en la Figura 2.5 se describen los ángulos de las articulaciones asociados con la *i-ésima* cadena cinemática del robot delta, donde el vector $\overrightarrow{OP}$ determina la posición del centro de la plataforma móvil, $\theta_{1i}$ es el ángulo que se forma entre la dirección $X_{i1}$ y el vector $\overrightarrow{A_iB_i}$, $\theta_{2i}$ es el ángulo que se forma ente la proyección de $\overrightarrow{A_iB_i}$ y la intersección del plano del paralelogramo y el plano $X_{i1} - Z_{i1}$ y $\theta_{3i}$ es el ángulo que se forma entre la dirección del eje $Y_{i1}$hasta el vector $\overrightarrow{B_iC_i}$.

Considerando las matrices de rotación y traslación en la cadena cinemática del robot se pueden obtener los siguientes vectores:

$$\overrightarrow{OA_i} = \begin{bmatrix} c\phi_i & -s\phi_i & 0 & Rc\phi_i \\ s\phi_i & c\phi_i & 0 & Rs\phi_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} Rc\phi_i \\ Rs\phi_i \\ 0 \\ 1 \end{bmatrix} \tag{2.2}$$

$$\overrightarrow{OB_i} = \begin{bmatrix} c\phi_i & -s\phi_i & 0 & Rc\phi_i \\ s\phi_i & c\phi_i & 0 & Rs\phi_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} L_1c\theta_{1i} \\ 0 \\ L_1s\theta_{1i} \\ 1 \end{bmatrix} = \begin{bmatrix} c\phi_i(R + L_1c\theta_{1i}) \\ s\phi(R + L_1c\theta_{1i}) \\ L_1s\theta_{1i} \\ 1 \end{bmatrix} \tag{2.3}$$

$$\overrightarrow{OC_i} = \begin{bmatrix} c\phi_i & -s\phi_i & 0 & Rc\phi_i \\ s\phi_i & c\phi_i & 0 & Rs\phi_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_xc\phi_i + P_ys\phi_i + r - R \\ P_yc\phi_i - P_xs\phi_i \\ P_Z \\ 1 \end{bmatrix} = \begin{bmatrix} P_x + rc\phi_i \\ P_y + rs\phi_i \\ P_z \\ 1 \end{bmatrix} \tag{2.4}$$

$$\overrightarrow{B_iA_i} = \overrightarrow{OA_i} - \overrightarrow{OB_i} = \begin{bmatrix} -c\phi_i L_1 c\theta_{1i} \\ -s\phi_i L_1 c\theta_{1i} \\ -L_1 s\theta_{1i} \end{bmatrix} \tag{2.5}$$

$$\overrightarrow{B_iC_i} = \overrightarrow{OC_i} - \overrightarrow{OB_i} = \begin{bmatrix} P_x - c\phi_i(R + L_1 c\theta_{1i} - r) \\ P_y - s\phi_i(R + L_1 c\theta_{1i} - r) \\ P_z - L_1 s\theta_{1i} \end{bmatrix} \tag{2.6}$$

$$\overrightarrow{A_iC_i} = \overrightarrow{OC_i} - \overrightarrow{OA_i} = \begin{bmatrix} P_x - c\phi_i(r - R) \\ P_y - s\phi_i(r - R) \\ P_Z \end{bmatrix} \tag{2.7}$$

Finalmente, en la Figura 2.6 se aprecia que la base fija y la plataforma móvil del robot paralelo son dos triángulos equiláteros de lados $L$ y $l$ respectivamente, de la misma manera la distancia desde el centro de la base $A_i$ es $R$, así como la distancia desde el punto $P$ hacia $C_i$ es $r$.

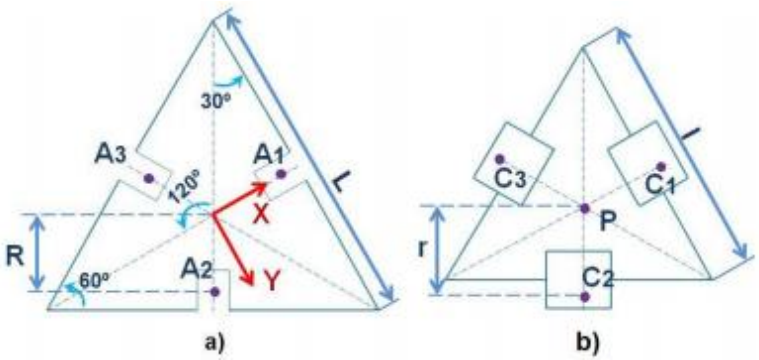


**Figura 2.6.** Forma geométrica de la a) base fija y b) plataforma móvil
Fuente: Propia.

### 2.1.5 Cinemática inversa

La cinemática inversa permite conocer las posiciones articulares de cada actuador conociendo la posición del efector final, en este caso particular sería la posición de la plataforma móvil. Solucionar la cinemática inversa permite al sistema de control del robot orientar los actuadores para llevar el efector final hasta un punto deseado en el espacio de trabajo.

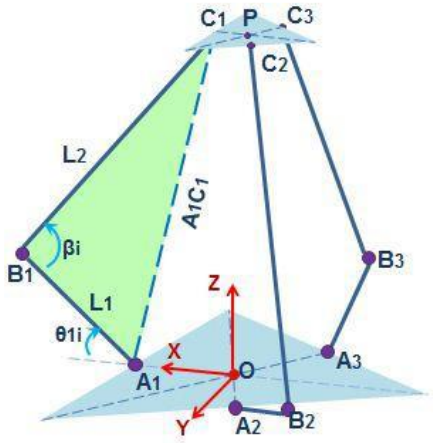


**Figura 2.7.** Representación vectorial de la primera cadena cinemática.
Fuente: Propia.

Basado en la representación vectorial de la primera cadena cinemática (ver Figura 2.7), se plantea lo siguiente utilizando la ecuación (2.8).

$$A_i C_i = L_1^2 + L_2^2 - 2L_1 L_2 \cos(\beta_i) \tag{2.8}$$

Bajo este contexto, debido a la simetría del robot, cada cadena cinemática puede ser analizada independientemente, y se obtiene la ecuación (2.9) para una determinada posición del efector final en el punto $P(P_x, P_y, P_z)$ con dimensiones $R$ para el diametro de la plataforma fija y $r$ para el diámetro de la plataforma móvil.

$$\theta_{1i} = \sin^{-1}\left(\frac{c_i}{\sqrt{a_i^2 + b_i^2}}\right) - \tan^{-1}\left(\frac{b_i}{a_i}\right) \tag{2.9}$$

Donde:

$a_i = 2P_z L_1$

$b_i = 2L_1\left(P_x c\phi_i + P_y s\phi_i + r - R\right)$

$c_i = \left(P_x + c\phi_i(r - R)\right)^2 + \left(P_y + s\phi_i(r - R)\right)^2 + P_z^2 + L_1^2 - L_2^2$

### 2.1.6 Cinemática directa

Para determinar la posición del efector final $\left(P_x, P_y, P_z\right)$ en base a las variables de las articulaciones $(\theta_{11}, \theta_{12}, \theta_{13})$, se estable las relaciones matemáticas donde se define el vector $\overrightarrow{\|B_I C_I\|}$ y cuya magnitud es $L_2$, tomando en cuenta las restricciones para acceder la plataforma móvil en base a los ángulos $\left(\beta_i, \theta_{2i}, \theta_{3i}\right)$.

$$\overrightarrow{\|BC_I\|} = \left(P_x - c\phi_i(R + L_1 c\theta_{1i} - r)\right)^2 + \left(P_y - s\phi_i(R + L_1 c\theta_{1i} - r)\right)^2 + (P_z + L_1 s\theta_{1i})^2 = L_2^2 \quad (2.10)$$

Para determinar las ecuaciones de la posición cartesiana se reemplaza los valores del ángulo $\phi_i = (0°, 120°, 240°)$ , los cuales fueron determinados por la simetría del robot y se obtiene lo siguiente:

$$P_x^2 + 2a_1 P_x + P_y^2 + P_z^2 - 2a_2 P_z = a_3 \quad (2.11)$$

Donde:

$a_1 = r - R - L_1 c\theta_{11}$
$a_2 = L_1 s\theta_{11}$
$a_3 = L_2^2 - a_1^2 - a_2^2$

$$P_x^2 + b_1 P_x + P_y^2 - \sqrt{3} b_1 P_y + P_z^2 - 2b_2 P_z = b_3 \quad (2.12)$$

Donde:

$b_1 = R - r + L_1 c\theta_{12}$
$b_2 = L_1 s\theta_{12}$
$b_3 = L_2^2 - b_1^2 - b_2^2$

$$P_x^2 + c_1 P_x + P_y^2 + \sqrt{3} c_1 P_y + P_z^2 - 2c_2 P_z = c_3 \quad (2.13)$$

Donde:

$c_1 = R - r + L_1 c\theta_{13}$
$c_2 = L_1 s\theta_{13}$
$c_3 = L_2^2 - c_1^2 - c_2^2$

Utilizando las ecuaciones (2.14), (2.15) y (2.16) se calculan las posiciones $(P_x, P_y, P_z)$.

$$P_y = \frac{A + BP_z}{C} \quad (2.14)$$

Donde:

$$A = \frac{a_3 - b_3}{2a_1 - b_1} - \frac{a_3 - c_3}{2a_1 - c_1}$$
$$B = \frac{2(c_2 - a_2)}{2a_1 - c_1} - \frac{2(b_2 - a_2)}{2a_1 - b_1}$$
$$C = \frac{\sqrt{3} b_1}{2a_1 - b_1} + \frac{\sqrt{3} c_1}{2a_1 - c_1}$$

$$P_x = D - EP_z \quad (2.15)$$

Donde:

$$D = \frac{a_3 - b_3}{2a_1 - b_1} - \frac{\sqrt{3} b_1 A}{C(2a_1 - b_1)}$$

$$E = \frac{2(b_2 - a_2)}{2a_1 - b_1} + \frac{\sqrt{3}b_1 A}{C(2a_1 - b_1)}$$

$$P_z = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{2.16}$$

Donde:

$$a = E^2 + (\frac{B}{C})^2 + 1$$
$$b = 2(\frac{AB}{C^2} - DE - a_1 E - a_2)$$
$$c = D^2 + (\frac{A}{C})^2 + 2a_1 D - a_3$$

### 2.1.7 Restricciones

No todos los puntos cartesianos pueden ser alcanzados por el robot, debido a que ocasionan inestabilidad y provocan colisiones entre sus componentes.

A continuación, se exponen las ecuaciones para encontrar los ángulos necesarios para detectar una colisión entre componentes.

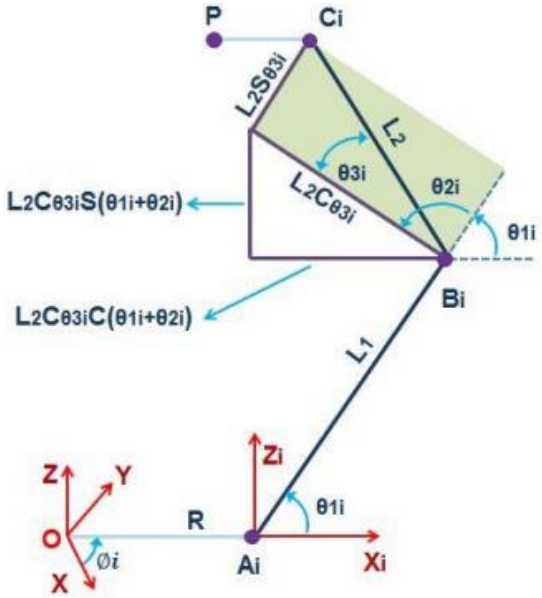


**Figura 2.8.** Cadena cinemática $i$ en las coordenadas $x_i\ y_i\ z_i$
Fuente: Propia.

En la Figura 2.8 se puede observar que el vector $\overrightarrow{A_i C_i}$ es la suma de los vectores $\overrightarrow{A_i B_i}$ y $B_i C_i$. De la ecuación (2.4) se determina que el vector $\overrightarrow{A_i C_i}$ se expresa de la siguiente manera:

$$\overrightarrow{A_iC_i} = \begin{bmatrix} P_x c\phi_i + P_y s\phi_i + r - R \\ P_y c\phi_i - P_x s\phi_i \\ P_Z \\ 1 \end{bmatrix} \tag{2.17}$$

De la misma manera utilizando la representación de la Figura 2.8 se aprecia que:

$$\overrightarrow{A_iC_i} + \overrightarrow{B_iC_i} = \begin{bmatrix} L_1 c\theta_{1i} + L_2 c\theta_{3i} c(180° - \theta_{1i} - \theta_{2i}) \\ L_2 s\theta_{3i} \\ L_1 s\theta_{1i} + L_2 c\theta_{3i} s(180° - \theta_{1i} - \theta_{2i}) \end{bmatrix} \tag{2.18}$$

Igualando las ecuaciones (2.17 y (2.18 se puede determinar los ángulos $\theta_{3i}$ y $\theta_{2i}$.

$$\theta_{3i} = \sin^{-1}\left(\frac{P_y c\phi_i - P_x s\phi_i}{L_2}\right) \tag{2.19}$$

$$\theta_{2i} = 180 - \sin^{-1}\left(\frac{P_z - L_1 s\theta_{1i}}{L_2 c\theta_3 i}\right) - \theta_{1i} \tag{2.20}$$

Para determinar el ángulo $\beta_i$ se requiere la posición de la plataforma móvil respecto a la fija, con la finalidad de encontrar la magnitud del vector $\overrightarrow{A_iC_i}$ y así despejar $\beta_i$.

$$\beta_i = \cos^{-1}\left(\frac{L_1^2 + L_2^2 - A_i C_i^2}{2L_1 L_2}\right) \tag{2.21}$$

Estas restricciones serán definidas en el controlador de posición implementado para evitar la colisión entre componentes del robot delta.

### 2.1.8 Espacio de Trabajo

Para encontrar el espacio de trabajo de un robot delta se utilizó el método geométrico, el cual consiste en:

- Definir el volumen de trabajo ($V$) que encierre el volumen de trabajo del robot Delta.
- Generar un número ($n_{total}$) de puntos, seleccionados aleatoriamente que se encuentren dentro del Volumen $V$, con una resolución en pasos de 0,225 mm.
- Utilizar la cinemática inversa para evaluar cada punto y determinar si se encuentra dentro del espacio de trabajo del robot.
- Una vez evaluados todos los puntos, se obtiene el número total de puntos dentro del espacio de trabajo del robot ($n_{in}$).

- El volumen del espacio de trabajo de robot se calcula multiplicando el volumen de trabajo ($V$) con el número total de puntos que alcanzo el robot y dividido por el número total de puntos seleccionados $V' = \frac{n_{in}}{n}$

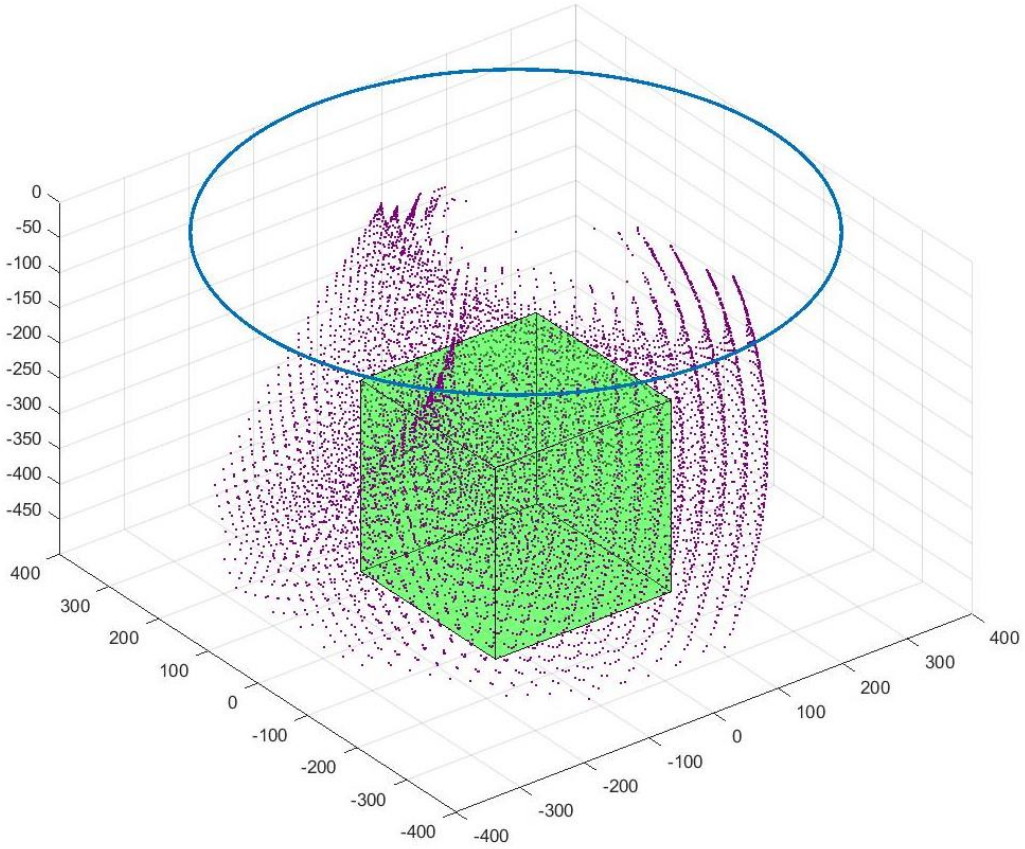


**Figura 2.9.** Espacio de trabajo del robot delta.
Fuente: Propia.

En la Figura 2.9 se puede observar el espacio de trabajo planteado para este proyecto (cubo de color verde), los puntos de color morado corresponden a puntos aleatorios y el círculo azul es la base fija del robot. Para definir el alcance del espacio de trabajo de las tres coordenadas cartesianas se utilizaron los valores máximos y mínimos de la gráfica de MatLAB, mismos que se detallan en la Tabla 2.2.

**Tabla 2.2.** Parámetros del espacio de trabajo.

| Coordenadas | Alcance Mínimo (mm) | Alcance Máximo (mm) |
|---|---|---|
| **X** | -125.00 | 125.00 |
| **Y** | -125.00 | 125.00 |
| **Z** | 0.00 | -450.00 |

Adicionalmente, la plataforma móvil alcanza su altura máxima $P_{zmax}$ cuando las posiciones angulares de los tres actuadores son 90°. Este parámetro se calcula utilizando la ecuación (2.22).

$$P_{zmax} = L_1 + \sqrt{(L_2)^2 - (R - r)^2} \tag{2.22}$$

Donde:

$L_1$: Articulación inferior.

$L_2$: Articulación superior.

$R$: Radio de la plataforma fija.

$r$: Radio de la plataforma móvil.

Obteniendo el siguiente resultado

$$P_{zmax} = 0.30 + \sqrt{(0.58)^2 - (0.4 - 0.05)^2}$$

$$\boxed{P_{zmax} = 0.7625}$$

## 2.2 Entorno de simulación del robot paralelo en V-REP

Para crear el robot delta en el entorno del simulador V-REP se utilizaron formas primitivas (*primitive shape*), concretamente cilindros y cubos para simular los eslabones de este. Para la creación de estas formas se utilizaron las dimensiones de las piezas del robot creadas en SOLIDWOKS. De igual manera, las propiedades dinámicas de estas formas permitieron modificar los parámetros de masa e inercia utilizando los datos obtenidos en SOLIDWOKS (ver Tabla 2.1).

En la Figura 2.11. se puede observar la jerarquía de los elementos que conforman el robot. El robot se ha constituido jerárquicamente desde el elemento denominado plataforma fija (Delta_base), debido a ser la parte no móvil del robot. Posteriormente, se encuentran las tres cadenas cinemáticas constituidas por los eslabones inferiores (Link1_1, Link2_1, Link3_1), eslabones superiores (Link1_2, Link2_2, Link3_2), articulaciones revolutas (R1_B, R2_B, R3_B), juntas universales intermedias (U1_J_PRLB, U1_PPDB, U2_J_PRLB, U2_PPDB, U3_J_PRLB, U3_PPDB) y juntas universales superiores (U1_EE_PPDB, U2_EE_PPDB, U3_EE_PPDB).

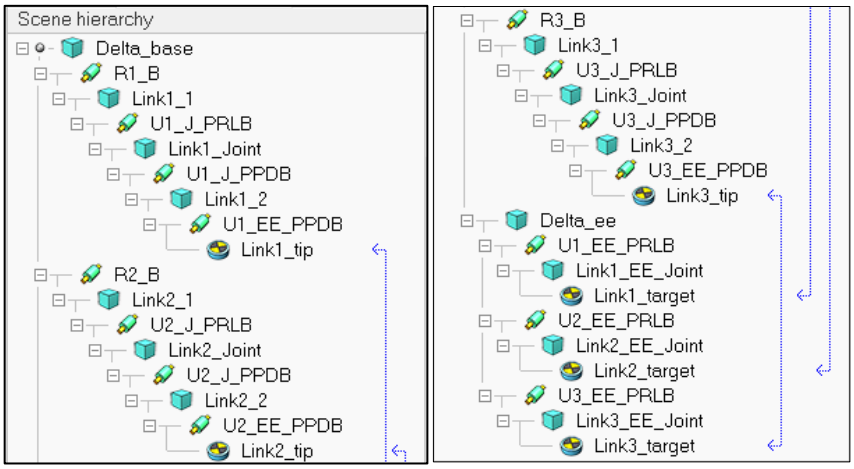


**Figura 2.10.** Jerarquía del robot simulado.
Fuente: Propia

Las tres cadenas cinemáticas están constituidas bajo el mismo patrón debido a que funcionan idénticamente. Cada cadena está constituida por una articulación anclada a la plataforma fija, que permite el movimiento angular, además, dos eslabones unidos por una junta universal, debido a los dos grados de libertad que esta maneja, y finalmente, se tiene una junta universal acoplada a la plataforma móvil como se puede apreciar en la Figura 2.11.

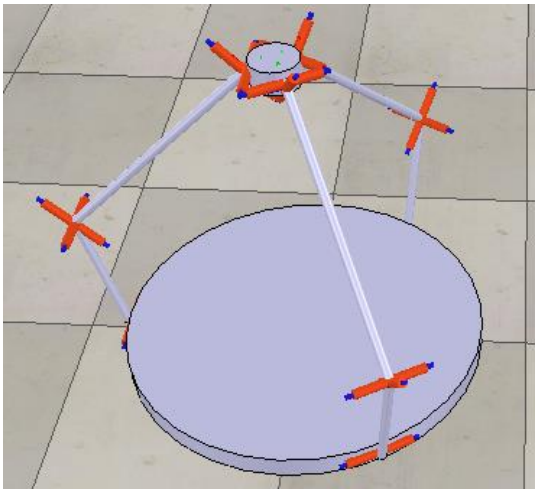

**Figura 2.11.** Diseño del robot configuración primitiva.
Fuente: Propia.

A partir de la jerarquía expuesta, se encuentran las articulaciones revolutas del robot cuya conexión con las cadenas cinemáticas se logra utilizando los *dummies* (Link1_tip, Link2_tip, Link3_tip y Link1_target, Link2_target, Link3_target). Estos objetos (*dummies*) se utilizan para identificar puntos específicos o especificar cierres de bucles o relaciones punto objetivo para cálculos dinámicos o cinemáticos. En este proyecto, se configuran como *dummy Target* de movimiento libre, *dummy Tip* encargado de seguir al Target para

cerrar las cadenas, *Dynamics, overlap constraint* con restricción dinámica entre formas, con la posibilidad de utilizar sensores (ver Figura 2.12).

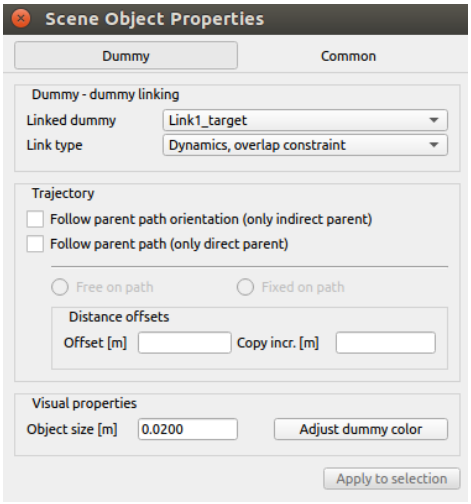


**Figura 2.12.** Propiedades del elemento Dummy.
Fuente: Propia

Una vez configurados los *dummies* para cerrar las tres cadenas cinemáticas que conforman el robot delta, se procede a definir las propiedades en los elementos *Joint* (R1_B, R2_B, R3_B) utilizando el modo *Torque/force mode* y su configuración cíclica para la posición angular (ver Figura 2.13). Esta configuración permitirá controlar independientemente la posición angular de las articulaciones mediante el parámetro de velocidad calculado por el controlador.

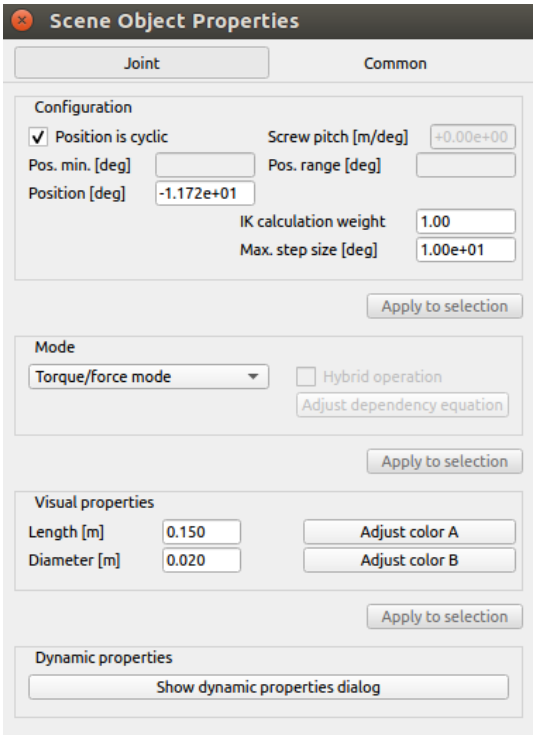


**Figura 2.13.** Configuración elemento Joint.
Fuente: Propia.

Para finalizar la configuración del Robot delta en el simulador V-REP, se definirán las propiedades dinámicas de los eslabones o cuerpos rígidos (ver Figura 2.14), utilizando los parámetros físicos de la Tabla 2.1. Concluido este procedimiento se podrá simular movimientos angulares en las cadenas cinemáticas del robot.

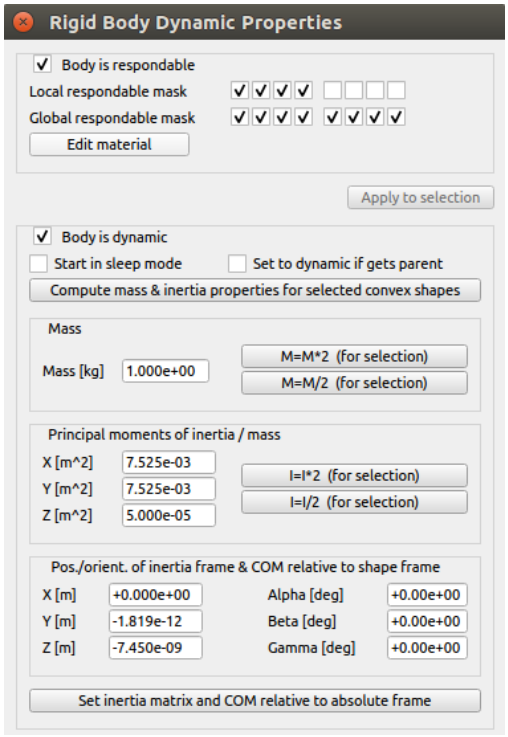


**Figura 2.14.** Configuración propiedades dinámicas de eslabones.
Fuente: Propia.

### 2.2.1 Programación del entorno de simulación

Para este proyecto se ha utilizado el lenguaje de programación Lua, scripts de programación incrustados, API ROSInterface y el motor de física *Bullet Physics v2.83* para establecer los requisitos físicos del entorno de simulación.

En la Figura 2.15 se puede observar la configuración del motor de física para aplicar la amplitud y dirección de la gravedad a todos los elementos del entorno de simulación.

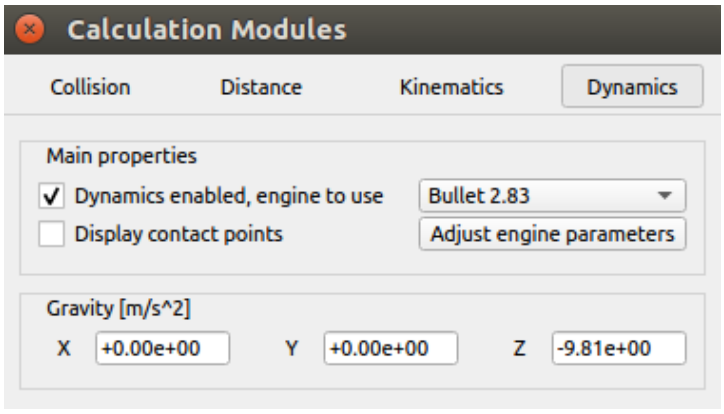


**Figura 2.15.** Diálogo de propiedades dinámicas generales.
Fuente: Propia.

A continuación, se detallan los tópicos que se utilizaron para intercambiar datos entre el simulador VREP y el controlador desarrollado en ROS.

**Tabla 2.3.** Tópicos establecidos en el simulador V-REP.

| Variable | Tipo de Dato | Función |
|---|---|---|
| jointpub | sensor_msgs/JointState | Publica la posición angular de las articulaciones revolutas. |
| eePospub | geometry_msgs/Pose | Publica la posición cartesiana del actuador final. |
| timepub | std_msgs/Float32 | Publica el tiempo de ejecución en el simulador. |
| jointVelSub | sensor_msgs/JointState | Suscribe la velocidad establecida por el controlador. |
| baseParamPub | std_msgs/Float64 | Publica el radio de la plataforma fija. |
| eeParamPub | std_msgs/Float64 | Publica el radio de la plataforma móvil. |
| lowLinkParamPub | std_msgs/Float64 | Publica la dimensión del eslabón inferior. |
| upLinkParamPub | std_msgs/Float64 | Publica la dimensión del eslabón superior. |

En el apartado 2.3 se detallará la función de estos tópicos. De la misma manera, una vez cargado el modelo del robot delta en la escena del simulador, se inicia la comunicación ROS para que el nodo /modelParamListener pueda cargar los parámetros del robot en los tópicos definidos para el efecto como se puede apreciar en la Figura 2.16.

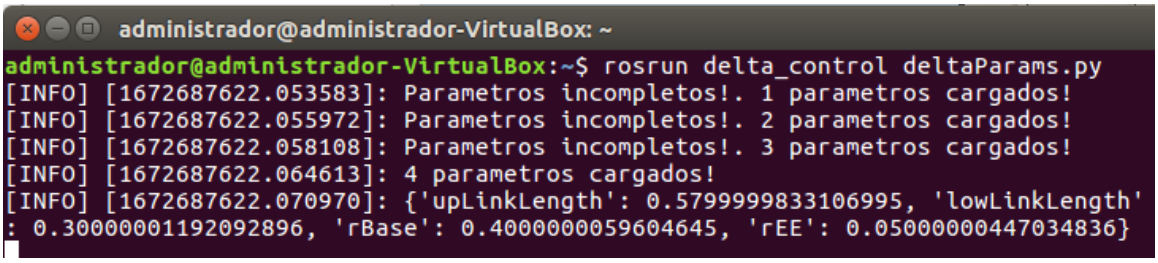


**Figura 2.16.** Parámetros del robot cargado en el nodo deltaParams.
Fuente: Propia.

Así mismo, el simulador publica en el tópico /Delta_base/rev_joint la posición angular actual de las articulaciones revolutas y suscribe en el tópico /Delta_base/joint_vel la velocidad calculada del controlador PID para mantener la posición cartesiana deseada.

En la Figura 2.17 se enlista los tópicos disponibles cuando el sistema de control se encuentra en ejecución.

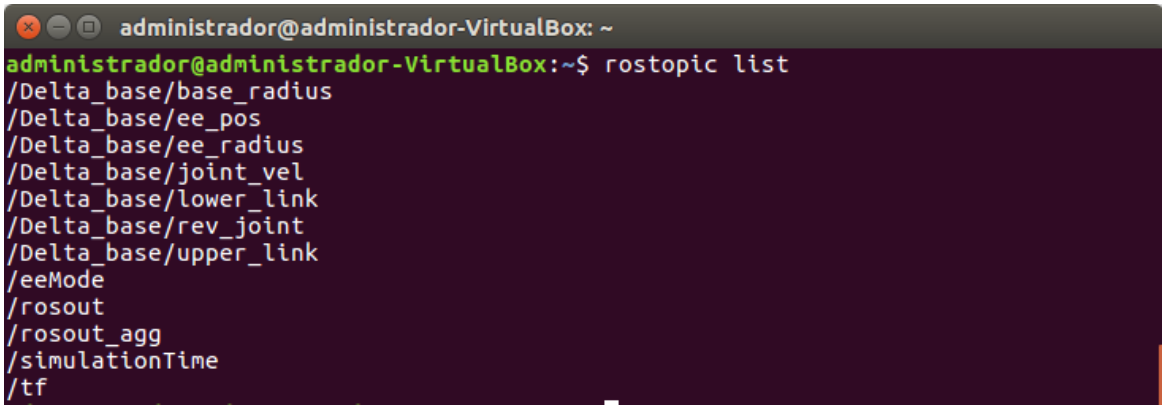


**Figura 2.17.** Tópicos del sistema ROS.
Fuente: Propia.

## 2.3 Diagrama de control implementado en la tarjeta de desarrollo

En la Figura 2.18 se puede apreciar el flujo de datos entre los nodos activos del sistema ROS utilizando la tarjeta de desarrollo embebida.

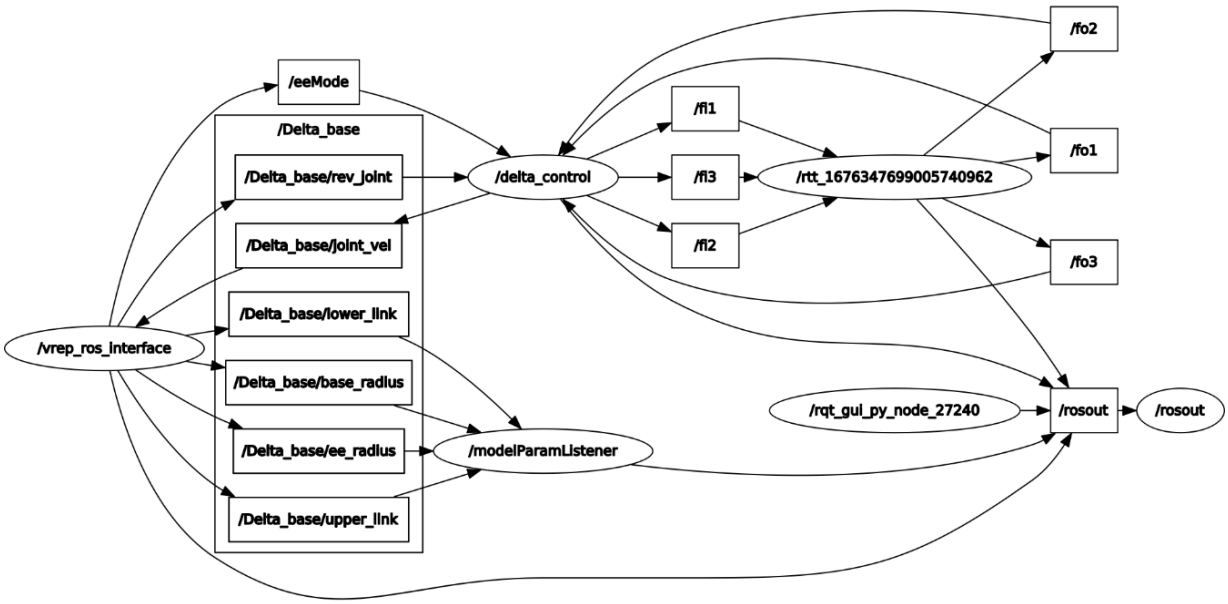


**Figura 2.18.** Diagrama de flujo del controlador implementado en OROCOS.
Fuente: Propia

Los nodos se detallan a continuación:

- El simulador V_REP utiliza el nodo /vrep_ros_interface para intercambiar los siguientes datos:
    - Salida: /eeMode bandera para indicar el inicio de la comunicación ROS.
    - Salida: /Delta_base/rev_joint entrega la posición angular de las articulaciones revolutas.

- Entrada: /Delta_base/joint_vel recibe la velocidad angular de las articulaciones revolutas.
- Salida: /Delta_base/lower_link, /Delta_base/base_radius, /Delta_base/ee_radius, /Delta_base/upper_link, entrega los parámetros geométricos de los componentes del robot.

- El nodo /rtt_1676347699005740962 es el paquete Orocos integrado a ROS que permite la ejecución del controlador PID de posición angular.
- El nodo /delta_control estable la interfaz de comunicación entre los nodos Orocos /rtt_1676347699005740962 y el simulador en V_REP nodo /vrep_ros_interface, estructurando los mensajes obtenidos en el controlador de posición utilizando la librería sensor_msgs.msg propia del sistema ROS.
- El nodo /modelParamListener almacena los parámetros físicos del robot publicados desde el simulador al activar la comunicación ROS.

## 2.4 Diagrama de control implementado en MatLAB

En la Figura 2.19 se puede apreciar el flujo de datos entre los nodos activos del sistema ROS utilizando el ordenador con el RosToolbox de MatLAB.

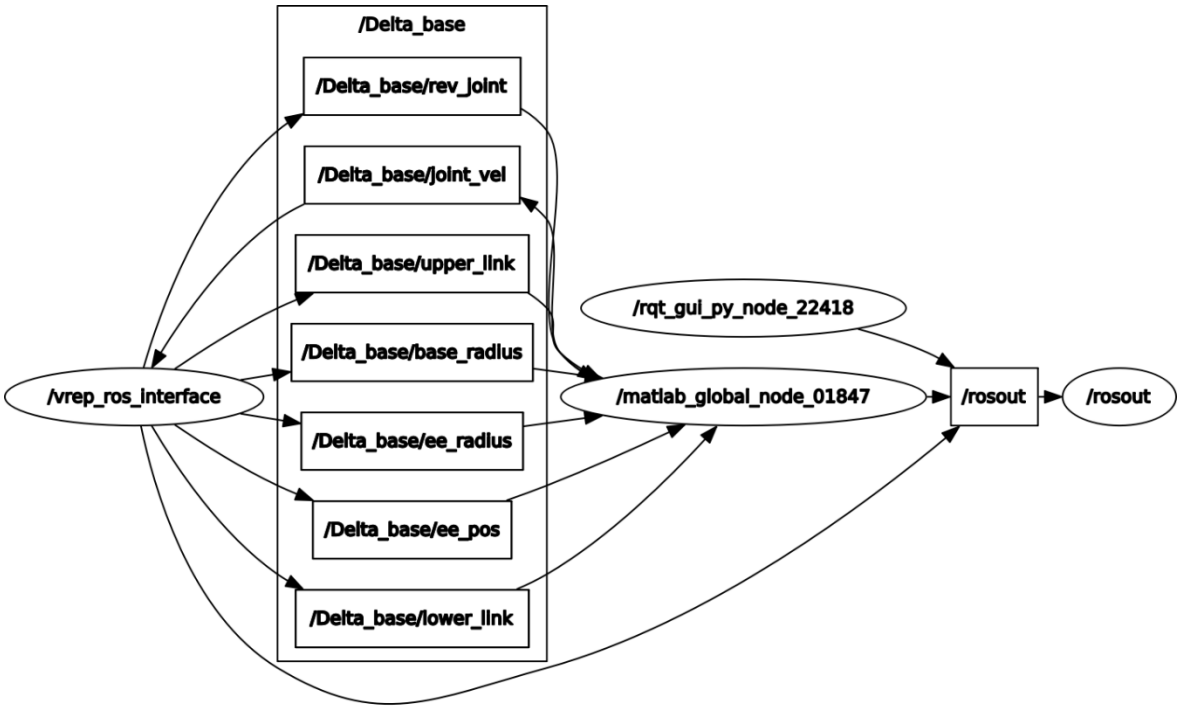


**Figura 2.19.** Diagrama de flujo del controlador implementado con el RosToolbox de MatLAB.
Fuente: Propia.

Los nodos se detallan a continuación:

- El simulador V_REP utilizado el nodo /vrep_ros_interface para manejar los datos de entrada y salida, entre estos se encuentran los parámetros físicos del robot, la posición angular medida y la recepción de la velocidad calcula del controlador de posición.
- El nodo /Matlab_global_node_01847 almacena los parámetros físicos del robot publicados desde el simulador y estable la comunicación entre el controlador PID de posición y el robot simulado en V_REP.

## 2.5 Sistema de control de posición

En primera instancia, el parámetro de referencia en el controlador es la posición cartesiana del efector final entregada por el planificador; parámetro que mediante la cinemática inversa del robot definirá la posición angular de los actuadores del robot, debido a esta necesidad de la posición angular y aplicaciones de medicina como [17] es necesario realizar un control de posición para analizar el posicionamiento en el espacio de trabajo.

El diagrama general de la operación del sistema de control de posición para el robot se puede visualizar en la Figura 2.21. La posición cartesiana deseada se genera mediante el planificador de trayectoria; los parámetros de referencia son las posiciones angulares de las articulaciones revolutas, los cuales son controlados por el PID hasta alcanzar su posición por el robot delta.

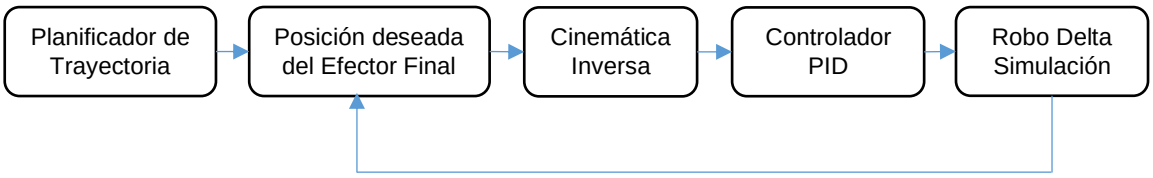


**Figura 2.21.** Diagrama del sistema de control de posición del robot delta.
Fuente: Propia.

El controlador PID fue utilizado, debido a que en la mayoría de los casos suele dar buenos resultados al disminuir el error entre el valor deseado y el valor medido. Un caso similar al presente trabajo es la simulación de un controlador para un péndulo invertido virtual ejecutando el middleware Orocos [18] donde se implementa un controlador PID.

Este controlador viene dado por la siguiente ecuación:

$$u(t) = K[e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d\frac{de(t)}{dt}] \quad (2.23)$$

Donde:

$u(t)$: variable de control.

$e(t)$: error de control dado por $e = y_{SP} - y$; diferencia entre la referencia especificada por la entrada y la salida medida del proceso.

$K$: ganancia proporcional.

$T_i$: tiempo integral.

$T_d$: tiempo derivativo.

En este proyecto, el controlador es aplicado independientemente a cada actuador para evitar oscilaciones producidas al momento que el efector final alcanza su posición deseada. En el apartado apartado 2.5.2 se detallará la sintonización del regulador PID para controlar el posicionamiento angular de los actuadores del robot, y de esta manera controlar el proceso (ver Figura 2.22).

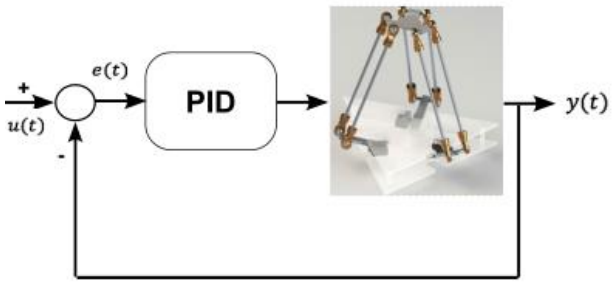


**Figura 2.22**. Diagrama de bloque del control PID propuesto para el robot delta.
Fuente: Propia.

La variable manipulada es la velocidad angular aplicada a los actuadores, y la variable controlada es la posición angular de estos. Estas variables son monitoreadas y controladas mediante los reguladores PID implementados en: middleware OROCOS del sistema embebido y Matlab, utilizando la red de comunicación ROS.

### 2.5.1 Función de transferencia.

Para estimar la función de transferencia de los actuadores del robot paralelo, es necesario centrarse en una cadena cinemática.

En primera instancia, se requiere adquirir la información del comportamiento del robot a una entrega de señal tipo escalón, por lo tanto, a un actuador del robot se envía una señal

para generar un movimiento con un valor intermedio de 0 rad a $\pi/8$ utilizando el middleware OROCOS del sistema en lazo abierto para obtener los vectores de entrada y salida.

En la Figura 2.23 se puede apreciar la respuesta del sistema, cuyo comportamiento debido a la inercia de la cadena cinemática la posición angular del motor inicia en un valor negativo. El tiempo de muestreo para la adquisición de datos es $Ts = 10\ ms$.

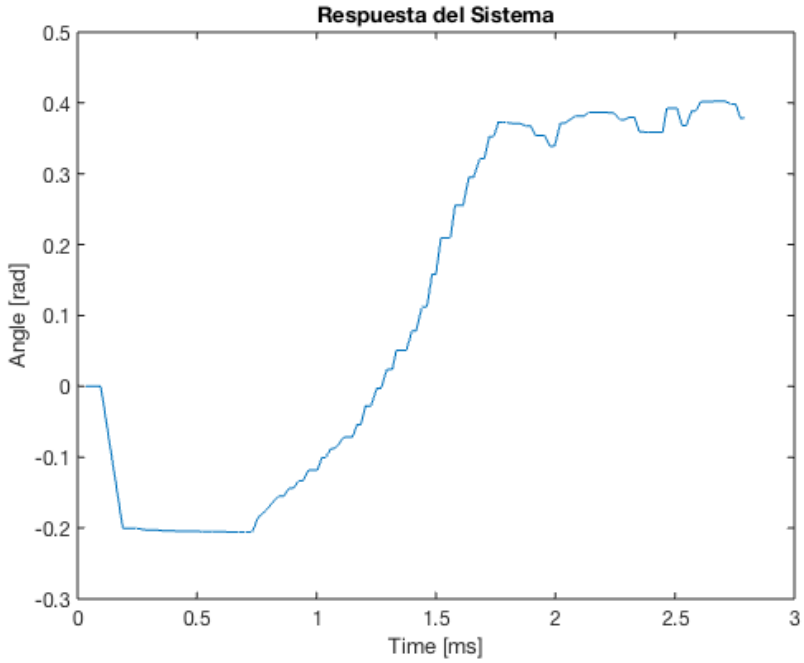


**Figura 2.23.** Respuesta del sistema ante una entrada tipo escalón.
Fuente: Propia.

Utilizando la herramienta de identificación IDENT de Matlab se identifica varios modelos indicados en la Tabla 2.4; en base al mejor porcentaje de estimación se selecciona la primera Función de Transferencia con su ecuación paramétrica (2.24). La estimación de la FT 1 y FT 3 no varía el porcentaje de estimación en gran medida, razón por lo cual se elige la FT1 por tener menos polos y ceros.

**Tabla 2.4.** Estimación de la función de transferencia.

| FT | Polos | Ceros | % de estimación |
|---|---|---|---|
| 1 | 2 | 1 | 87.36 |
| 2 | 3 | 1 | 23.35 |
| 3 | 2 | 2 | 87.63 |
| 4 | 3 | 2 | 30.67 |

$$G(s) = \frac{-1.572s + 6.905}{s^2 + 2.741s + 35.14} \tag{2.24}$$

La respuesta a una señal paso de la función de transferencia se puede observar en la Figura 2.24.

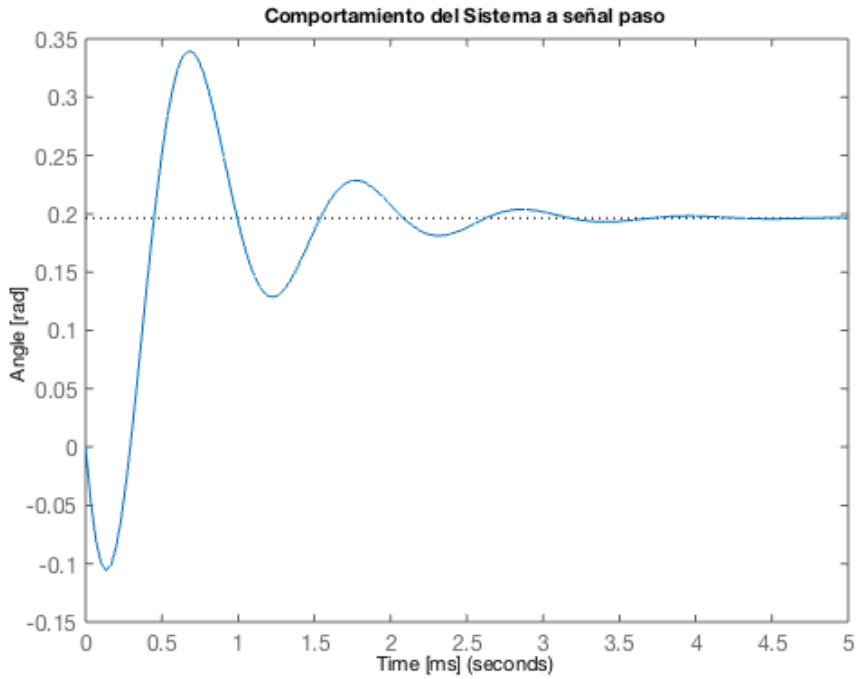


**Figura 2.24.** Comportamiento del sistema a señal paso.
Fuente: Propia.

### 2.5.2 Diseño del controlador PID de posición

El controlador PID será aplicado independientemente a cada actuador del robot, con la finalidad de que el movimiento de las cadenas cinemáticas permita que la posición cartesiana deseada en el espacio de trabajo sea alcanzada. El controlador diseñado se implementará tanto en Orocos como en Matlab. El esquema del controlador se puede observar en la Figura 2.25.

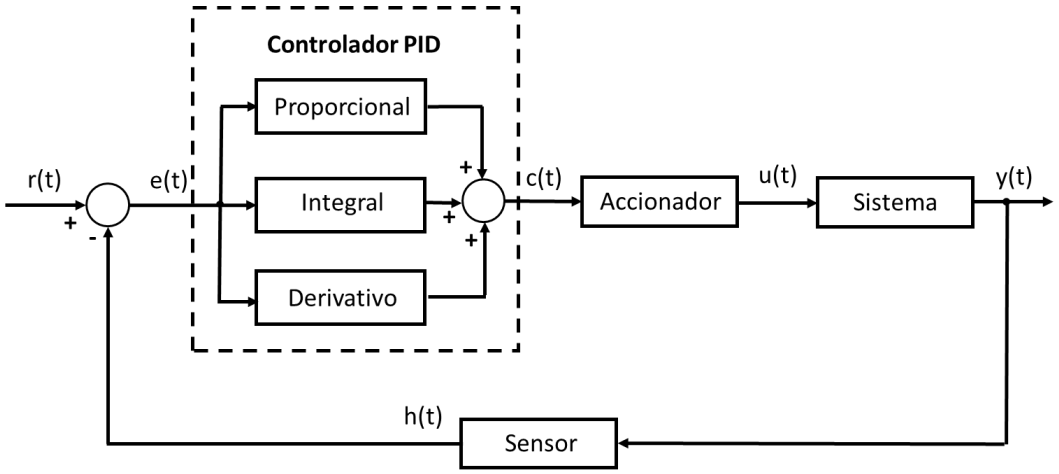


**Figura 2.25.** Controlador PID [19].

La fórmula del controlador PID viene dada por la siguiente expresión:

$$G_C(s) = k_p + \frac{k_i}{s} + k_d * s \tag{2.25}$$

Los términos de la expresión tienen correspondencia de la siguiente manera con la ecuación (2.23):

**Tabla 2.5.** Correspondencia función del controlador PID.

| | |
|---|---|
| $k_p$= | $K$ |
| $k_i$= | $\frac{K}{T_i}$ |
| $k_d$= | $K * T_d$ |

Para realizar la sintonización del controlador utilizando el método de Ziegler y Nichols en lazo cerrado se utilizarán las siguientes expresiones [20]; estas se encuentran en función de la ganancia proporcional crítica ($K_C$) y el período de oscilación sostenida ($Tc$)

$$K = 0.6 * K_C \tag{2.26}$$

$$T_i = 0.5 * T_C \tag{2.27}$$

$$T_d = 0.125 * T_C \tag{2.28}$$

Utilizando la representación de las raíces de la función del sistema $G$ (ver Figura 2.26), se definirá los puntos de corte de las raíces con el eje imaginario para en consecuencia obtener los parámetros $K_C$ ganancia crítica y la velocidad angular $\omega_C$ de la oscilación sostenida.

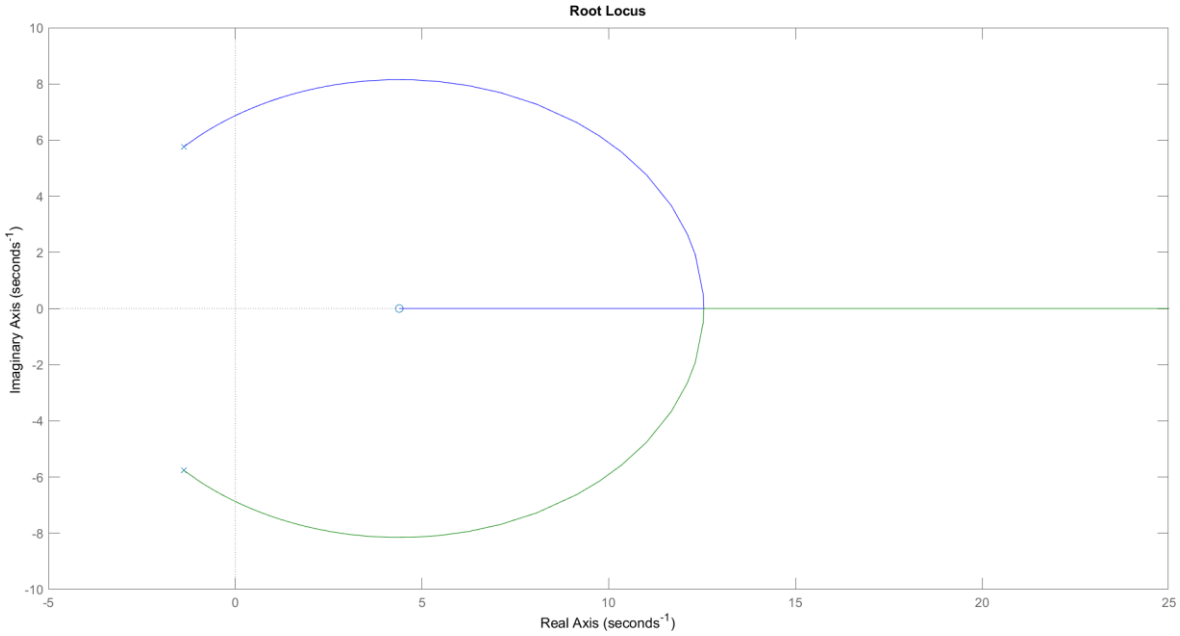


**Figura 2.26.** Representación inicial del lugar de las raíces.
Fuente: Propia

Para este caso los puntos de corte de las raíces con el eje imaginario son:

$$0.0005 + 6.8691i$$
$$0.0005 - 6.8691i$$

De donde se obtiene también que:

$$K_C = 1.7443$$
$$\omega_C = 6.8691\ rad$$

El periodo de oscilación sostenida $T_C$ que se consigue con la ganancia crítica $K_C$ es determinado con la siguiente relación:

$$T_C = \frac{2\pi}{\omega_C} \tag{2.29}$$

Obteniendo el siguiente resultado:

$$T_C = 0.9147 \tag{2.30}$$

De la misma manera utilizando las expresiones de sintonía para el regulador PID se obtiene los parámetros del regulador en función de las especificaciones en lazo cerrado:

$$k_p = 1.047$$
$$k_d = 0.1197$$
$$k_i = 2.288$$

Cuya función de transferencia queda definida como:

$$G_C(s) = \frac{0.1197 * s^2 + 1.047 * s + 2.288}{s} \tag{2.31}$$

Para comprobar el funcionamiento se realiza la comparación del sistema con respecto al sistema con regulador PID mismo que se puede visualizar en la Figura 2.27. Las oscilaciones producidas se deben a que la estimación de la planta se realizó con una variación angular intermedia de 0 a $\pi/8$, sin embargo, las articulaciones variaran angularmente en pasos pequeños al tener un conjunto de puntos cartesianos que definen la trayectoria.

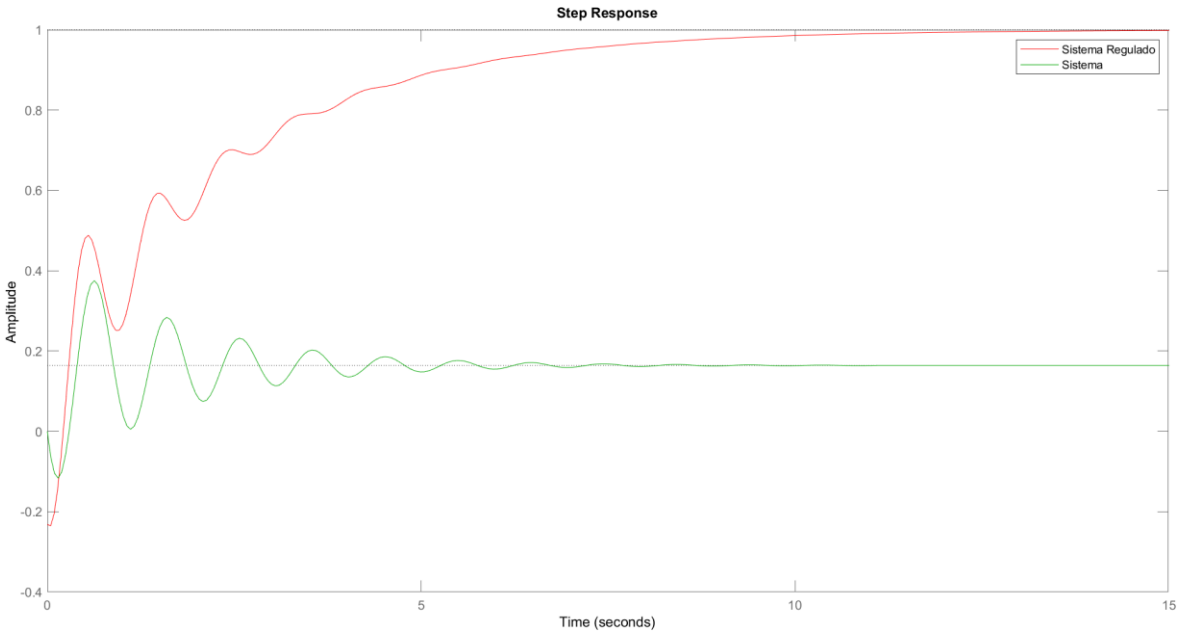


**Figura 2.27.** Comparación de la respuesta del Sistema vs Sistema regulado.
Fuente: Propia

### 2.5.3 Programación en ROS

Para implementar el controlador PID de posición en la tarjeta Raspberry con el middleware Orocos se utilizó tres programas desarrollados en Python.

El script deltaParams.py obtiene los parámetros del robot simulado, cuyos datos son almacenados en los tópicos detallados en la Tabla 2.6.

**Tabla 2.6.** Tópicos de los parámetros del robot.

| Tópico | Dato |
|---|---|
| /Delta_base/base_radius | Radio de la plataforma fija. |
| /Delta_base/ee_radius | Radio de la plataforma móvil. |
| /Delta_base/lower_link | Longitud del eslabón inferior. |
| /Delta_base/upper_link | Longitud del eslabón superior. |

El script **traj_control.py** realiza los cálculos de cinemática inversa y directa del robot paralelo mediante las funciones de deltaModule.py. La trayectoria cartesiana es generada por la función **setUpTraj()** cuyos datos son almacenados en la matriz traj cuyas posiciones angulares deseadas se calculan utilizando la cinemática inversa.

Utilizando la función **loadJointsPos(self, joint_data)** se obtiene la posición angular de las articulaciones del robot simulado en VREP, para de esta manera calcular el error respecto a la posición deseada y publicarla en los tópicos /fi1, /fi2, /fi3 que se utilizarán en Orocos.

Las velocidades angulares calculadas utilizando el controlador de Orocos se suscribirán en los tópicos /fo1, /fo2, /fo3 respectivamente y se publicarán a través del nodo delta_control debido a la facilidad que tiene ROS para el manejo de mensajes comunes y genéricos específicos para robots; siendo este caso el robot paralelo simulado en VREP.

### 2.5.4 Programación en Orocos

El controlador PID de posición implementado con el middleware Orocos, utiliza un script de extensión OPS que conecta los componentes y un script programado en C++ denominado Control.cpp donde se ejecuta el controlador utilizando la herramienta Orocos Real-Time Toolkit (RTT).

El script **inicio.ops** conecta los componentes de entrada y salida del controlador con los tópicos a publicar en el sistema ROS.

**Tabla 2.7.** Tópicos utilizados en el controlador de Orocos.

| Tópico | Dato |
|---|---|
| /fi1 | Error angular de la articulación 1. |
| /fi2 | Error angular de la articulación 2. |
| /fi3 | Error angular de la articulación 3. |
| /fo1 | Velocidad calculada de la articulación 1. |
| /fo2 | Velocidad calculada de la articulación 2. |

| | |
|---|---|
| /fo3 | Velocidad calculada de la articulación 3. |

El script Control.cpp ejecuta el controlador PID de posición para las tres articulaciones revolutas del robot simulado en VREP utilizado su herramienta RTT en tiempo real como se puede apreciar en la Figura 2.28.

```
if(NewData==input_1.read(fdata_1)){
  log(Info)<<"error q1: "<<fdata_1<<endlog();
  edata[0]=fdata_1.data;
  eInt[0]+=edata[0];
  Derror[0]=(edata[0]-eprev[0])/dt;
  eprev[0]=edata[0];
  udata_1.data=(Kp*edata[0]+Ki*eInt[0]+Kd*Derror[0])*(M_PI/180);
  output_1.write(udata_1);
}

if(NewData==input_2.read(fdata_2)){
  log(Info)<<"error q2: "<<fdata_2<<endlog();
  edata[1]=fdata_2.data;
  eInt[1]+=edata[1];
  Derror[1]=(edata[1]-eprev[1])/dt;
  eprev[1]=edata[1];
  udata_2.data=(Kp*edata[1]+Ki*eInt[1]+Kd*Derror[1])*(M_PI/180);
  output_2.write(udata_2);
}

if(NewData==input_3.read(fdata_3)){
  log(Info)<<"error q3: "<<fdata_3<<endlog();
  edata[2]=fdata_3.data;
  eInt[2]+=edata[2];
  Derror[2]=(edata[2]-eprev[2])/dt;
  eprev[2]=edata[2];
  udata_3.data=(Kp*edata[2]+Ki*eInt[2]+Kd*Derror[2])*(M_PI/180);
  output_3.write(udata_3);
}
```

**Figura 2.28.** Implementación del controlador en Orocos.

Para ejecutar el controlador, primero se abre la terminal de Ubuntu desde el directorio del controlador y se utiliza la función mostrada en la Figura 2.29 para ejecutar el script inicio.ops desde del deployer de Orocos.

```
~/catkin_ws/src/robot_control$ rosrun rtt_ros deployer inicio.ops
```

**Figura 2.29.** Instrucción para ejecutar el deployer de Orocos.

En la Figura 2.30 se puede observar la ejecución del controlador mediante el deployer de orocos satisfactoriamente.

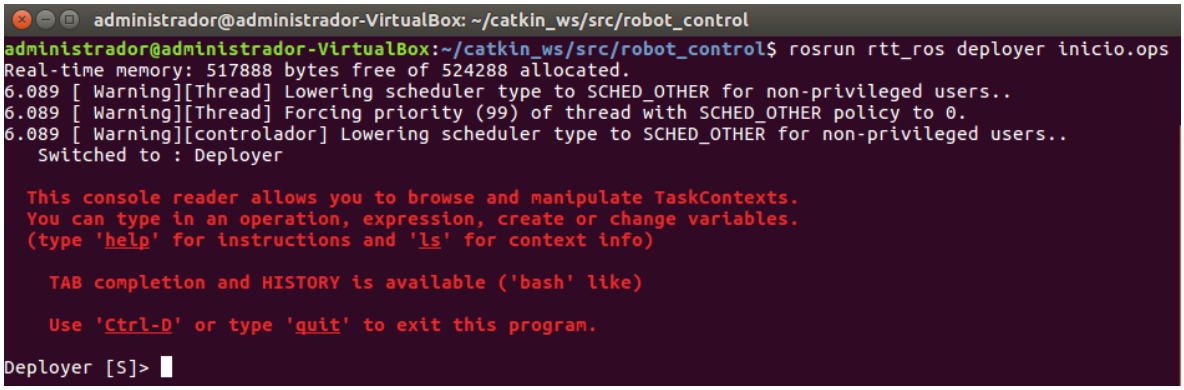


**Figura 2.30.** Deployer en Orocos ejecutado.

Finalmente, en la Figura 2.31 se pueden observar todos los tópicos utilizados en el sistema ROS para utilizar el controlador implementado en Orocos que controla el robot simulado en VREP.

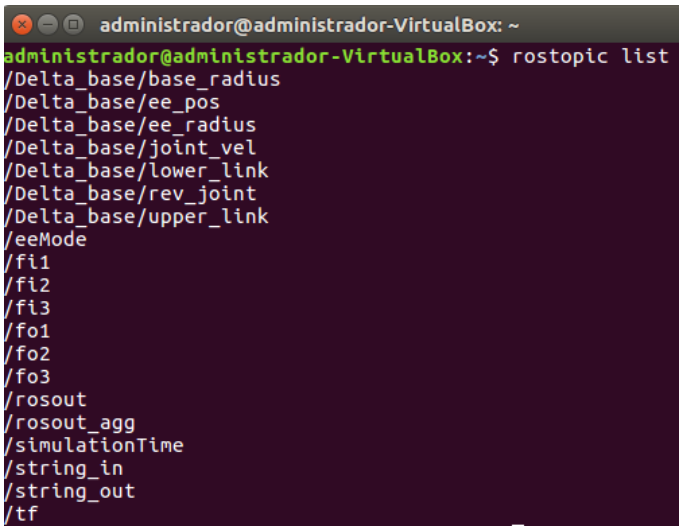


**Figura 2.31.** Tópicos utilizados en el sistema de control.

### 2.5.5 Programación en MatLAB

El controlador PID de posición implementado en MatLAB, utiliza el ROS Toolbox para comunicarse con el simulador mediante los tópicos detallados en la Tabla 2.8.

**Tabla 2.8.** Tópicos del controlador MatLAB.

| Tópico | Dato | Tipo |
|---|---|---|
| /Delta_base/base_radius | Radio de la plataforma fija. | Entrada |
| /Delta_base/ee_radius | Radio de la plataforma móvil. | Entrada |
| /Delta_base/lower_link | Longitud del eslabón inferior. | Entrada |
| /Delta_base/upper_link | Longitud del eslabón superior. | Entrada |
| /Delta_base/joint_vel | Velocidad angular de las articulaciones | Salida |
| /Traj_desired | Posición cartesiana deseada. | Salida |

| | | |
|---|---|---|
| /Delta_base/rev_joint | Posición angular de las articulaciones. | Entrada |
| /Delta_base/ee_pos | Posición cartesiana | Salida |

La comunicación del nodo ROS generado en MatLAB se comunica con el nodo master levantado en la máquina virtual donde se ejecuta el simulador V-REP. Este controlador utiliza la función ik_delta para obtener mediante la cinemática inversa la posición angular deseada en base a la trayectoria cartesiana generada al inicio del programa. Para el controlador PID implementado en MatLAB se han utilizado las mismas constantes que en Orocos, el código del controlador se puede apreciar en la Figura 2.32.

```
errorAng = (trajAng - curAng)
eInt = eInt + errorAng
Derror = (errorAng - eprev)/dt
eprev = errorAng
control_vel = (Kp*errorAng + Ki*eInt + Kd*Derror)*pi/180

controlInfo.Name = jointNames;
controlInfo.Velocity = [control_vel(1),control_vel(2),control_vel(3)];
send(jVelPub,controlInfo);
```

**Figura 2.32.** Controlador implementado en MatLAB.

### 2.5.6 Esquema general del sistema de comunicación

La red de comunicación ROS está conformada por el nodo ROS master, el nodo esclavo simulador VREP, y el nodo esclavo del controlador implementado en la tarjeta de desarrollo Raspberry. La comunicación entre los nodos se realiza mediante WIFI TCP lo cual permite intercambio de datos. El diagrama general del sistema de comunicación se puede apreciar en la Figura 2.33.

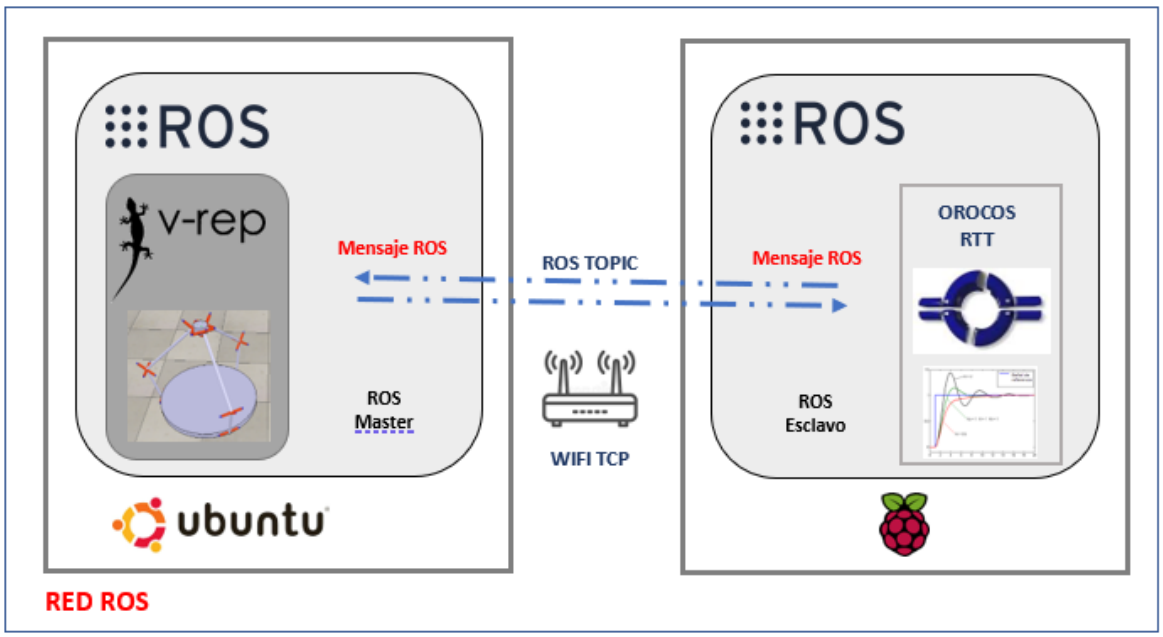


**Figura 2.33.** Diagrama de la red ROS.
Fuente Propia

El nodo ROS master y el simulador VREP, se ejecutarán en la máquina virtual; para comunicarse con:

- La tarjeta de desarrollo embebido utilizará un nodo esclavo ROS con la finalidad de estructurar un mensaje a interpretar en el simulador utilizando la señal de control calculada con el paquete de integración de OROCOS.
- El controlador implementado en MatLAB utilizará el nodo mediante el ROS Toolbox de MatLAB, y así controlar directamente al robot del simulador.

# 3 RESULTADOS Y DISCUSIÓN

En el presente capítulo se explica los resultados de la implementación del sistema de control de posición del robot delta utilizando: OROCOS en la placa embebida y ROS Toolbox de Matlab, evidentemente; el sistema (robot delta) será simulado en la plataforma virtual V-REP.

## 3.1 Resultados

Los resultados se analizarán de la siguiente manera:

- Se realizará el ensayo establecido en la norma ISO 9283 del controlador implementado en OROCOS.
- Se realizará las pruebas de trayectoria de los controladores implementados en OROCOS y ROS Toolbox de Matlab.
- Se realizará la comparación de los resultados de los controladores calculando el índice de desempeño.

El sistema de control del robot delta tiene como entrada: la posición cartesiana X, Y, Z del efector final que mediante la cinemática inversa entrega los ángulos deseados en los actuadores, el control de la posición angular será realizado con la velocidad angular entregada a los actuadores en base al regulador PID implementado.

La norma ISO 9283 [21] detalla una serie de pruebas estandarizadas para realizar ensayos de repetibilidad y precisión de la trayectoria, con la finalidad de determinar la capacidad de un robot para realizar tareas específicas.

### 3.1.1 Ensayo para precisión

Para este ensayo el robot inicia el ciclo por P1 y se mueve sucesivamente a las posiciones P5, P4, P3, P2, P1. Cada una de estas posiciones debe visitarse con una aproximación unidireccional como se muestra en cada ciclo de la Figura 3.1.

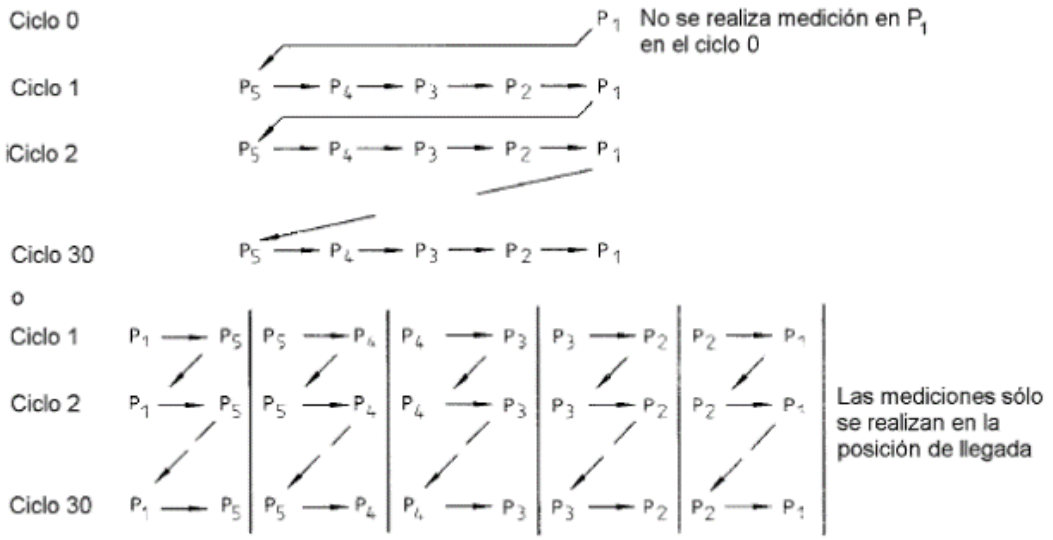


**Figura 3.1.** Ilustración de ciclos posibles. [21]

Para realizar los ensayos del control básico de posición del robot delta se ubican cinco puntos a medir en las diagonales del plano que corresponden a los puntos P1 a P5, como se puede observar en la Figura 3.2.

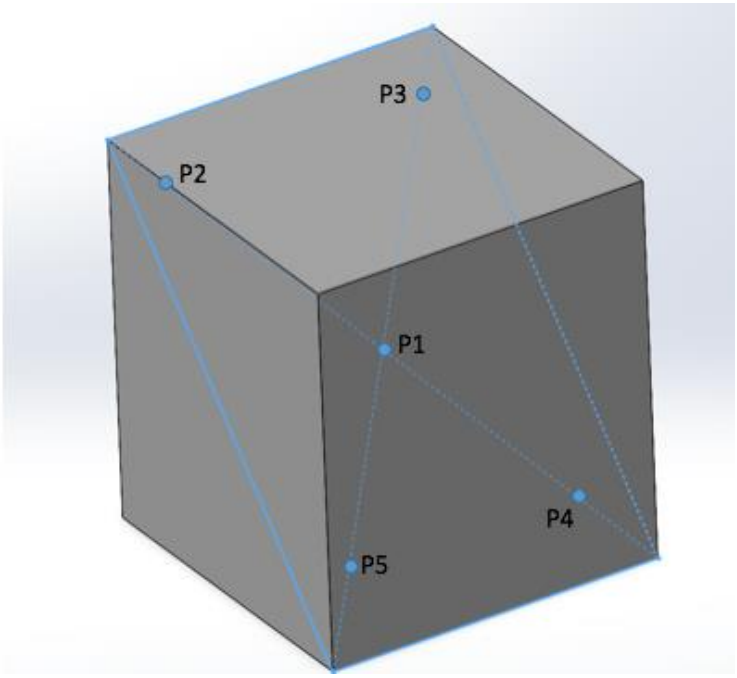


**Figura 3.2.** Plano y puntos para el ensayo.
Fuente: Propia.

P1 es la intersección de las diagonales y es el centro del cubo que forma el espacio de trabajo. Los puntos P2 a P5 están ubicados a una distancia del extremo de las diagonales

igual al (10±2) % de la longitud de la diagonal. En la Tabla 3.1. se detallan las coordenadas de las pruebas a realizar:

**Tabla 3.1.** Coordenadas para probar las características de posicionamiento.

| Punto | Coordenadas | | |
|---|---|---|---|
| | X(mm) | Y(mm) | Z(mm) |
| $P_1$ | 0.00 | 0.00 | 225.00 |
| $P_2$ | -100.00 | -100.00 | 425.00 |
| $P_3$ | -100.00 | 100.00 | 425.00 |
| $P_4$ | 100.00 | 100.00 | 25.00 |
| $P_5$ | 100.00 | -100.00 | 25.00 |

El número de ciclos a realizar al probar la característica de posición se define en la Tabla 3.2.

**Tabla 3.2.** Número de ciclos.[21]

| Características para probar | Número de ciclos |
|---|---|
| Precisión de posición. | 30 |
| Repetibilidad de posición. | 30 |
| Precisión de distancia. | 30 |
| Repetibilidad de distancia. | 30 |
| Tiempo de estabilización de posición. | 3 |
| Rebose de posición. | 3 |

### 3.1.2 Resultados del ensayo.

Los resultados obtenidos en el ensayo de precisión y repetibilidad de posicionamiento del robot delta con regulador PID implementado en el middleware Orocos se encuentran detallados en la siguiente tabla. Se han almacenado datos de la posición cartesiana del efector final del robot delta simulado en VREP para los diferentes puntos y ciclos del ensayo.

**Tabla 3.3.** Resultados ciclos de ensayo.

| Ciclo | Eje | P1 | P2 | P3 | P4 | P5 |
|---|---|---|---|---|---|---|
| **1** | x | 0.08 | -98.91 | -99.51 | 99.02 | 100.01 |
| | y | 0.02 | -99.03 | 98.12 | 100.02 | -99.81 |
| | z | 225.01 | 425.01 | 424.87 | 24.87 | 25.07 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **2** | x | 0.03 | -99.02 | -97.42 | 100.05 | 100.02 |
| | y | 0.05 | -100.05 | 97.05 | 99.54 | -98.01 |
| | z | 224.91 | 424.12 | 425.15 | 25.09 | 25.02 |
| **3** | x | 0.91 | -100.02 | -98.05 | 100.01 | 99.81 |
| | y | 0.07 | -98.01 | 99.12 | 98.97 | -100.09 |
| | z | 225.13 | 423.91 | 423.91 | 25.01 | 24.91 |
| **4** | x | 0.01 | -100.02 | -99.01 | 100.05 | 99.96 |
| | y | 0.05 | -100.05 | 99.08 | 100.07 | -100.06 |
| | z | 224.87 | 424.97 | 425.07 | 25.09 | 24.92 |
| **5** | x | 1.01 | -100.06 | -100.07 | 101.21 | 99.36 |
| | y | 0.96 | -99.83 | 100.61 | 99.87 | -100.05 |
| | z | 224.97 | 424.97 | 425.32 | 25.01 | 24.97 |
| **6** | x | 0.01 | -99.02 | -99.63 | 99.12 | 99.85 |
| | y | 0.01 | -99.87 | 99.18 | 99.65 | -101.01 |
| | z | 225.01 | 425.01 | 424.95 | 25.01 | 24.53 |
| **7** | x | 0.05 | -97.13 | -98.12 | 98.97 | 99.21 |
| | y | 0.05 | -99.65 | 99.87 | 99.63 | -99.68 |
| | z | 224.98 | 425.12 | 424.91 | 25.34 | 24.15 |
| **8** | x | 1.12 | -100.12 | -100.02 | 99.16 | 99.56 |
| | y | 1.01 | -100.01 | 99.63 | 99.61 | -102.01 |
| | z | 224.56 | 425.16 | 424.18 | 25.64 | 24.52 |
| **9** | x | 0.09 | -100.01 | -100.01 | 99.48 | 99.12 |
| | y | 1.02 | -98.65 | 98.15 | 99.45 | -98.65 |
| | z | 224.51 | 425.90 | 425.01 | 25.14 | 24.01 |
| **10** | x | 0.05 | -99.98 | -100.07 | 99.65 | 99.85 |
| | y | 0.01 | -99.12 | 97.65 | 99.87 | -99.12 |
| | z | 224.26 | 425.12 | 426.12 | 25.48 | 24.98 |
| **11** | x | 1.01 | -100.02 | -99.63 | 100.02 | 101.21 |
| | y | 0.07 | -100.12 | 100.01 | 98.25 | -101.23 |
| | z | 225.01 | 425.01 | 426.70 | 24.95 | 25.06 |
| **12** | x | 2.17 | -99.87 | -99.71 | 99.87 | 102.50 |
| | y | 0.09 | -100.39 | 101.21 | 98.12 | -98.65 |
| | z | 225.12 | 425.15 | 425.17 | 24.56 | 25.02 |
| **13** | x | 0.03 | -100.27 | -98.13 | 99.01 | 101.20 |
| | y | 1.03 | -99.86 | 102.10 | 99.02 | -101.12 |
| | z | 225.61 | 425.18 | 425.01 | 25.01 | 25.06 |
| **14** | x | 0.07 | -98.91 | -99.12 | 98.67 | 101.78 |
| | y | 2.00 | -97.63 | 99.79 | 101.23 | -100.91 |
| | z | 225.18 | 425.78 | 425.12 | 25.14 | 25.09 |
| **15** | x | 0.09 | -100.01 | -100.01 | 101.70 | 102.96 |
| | y | 0.09 | -100.01 | 101.45 | 101.25 | -101.21 |
| | z | 225.14 | 424.98 | 426.12 | 25.60 | 25.07 |
| **16** | x | 0.01 | -99.97 | -101.12 | 102.01 | 13.01 |
| | y | 0.08 | -101.12 | 99.87 | 101.14 | -101.15 |
| | z | 225.03 | 424.52 | 425.07 | 25.98 | 25.00 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **17** | x | 1.23 | -99.99 | -99.51 | 99.87 | 99.86 |
| | y | 0.01 | -99.98 | 98.62 | 102.12 | -99.86 |
| | z | 225.01 | 425.32 | 425.12 | 25.12 | 25.14 |
| **18** | x | 2.01 | -100.12 | -98.14 | 99.86 | 99.84 |
| | y | 0.07 | -100.71 | 101.10 | 101.95 | -101.23 |
| | z | 225.12 | 424.12 | 425.01 | 25.14 | 25.69 |
| **19** | x | 0.05 | -98.12 | -100.01 | 99.85 | 99.57 |
| | y | 0.06 | -99.65 | 103.01 | 100.91 | -99.87 |
| | z | 225.96 | 423.08 | 425.02 | 24.98 | 25.24 |
| **20** | x | 0.07 | -99.12 | -100.14 | 99.23 | 98.25 |
| | y | 0.05 | -100.01 | 101.29 | 101.00 | -99.12 |
| | z | 225.01 | 426.78 | 424.91 | 23.91 | 25.14 |
| **21** | x | 0.02 | -100.01 | -101.25 | 100.01 | 98.79 |
| | y | 0.04 | -99.62 | 100.90 | 99.81 | -102.10 |
| | z | 226.12 | 425.01 | 424.12 | 25.01 | 25.32 |
| **22** | x | 1.09 | -99.78 | -102.12 | 100.02 | 97.96 |
| | y | 0.03 | -98.75 | 99.87 | 96.71 | -102.19 |
| | z | 225.12 | 424.98 | 424.98 | 24.12 | 24.98 |
| **23** | x | 1.25 | -100.12 | -100.07 | 99.87 | 101.21 |
| | y | 0.09 | -101.21 | 99.65 | 99.12 | -99.87 |
| | z | 224.91 | 425.13 | 425.09 | 24.97 | 25.64 |
| **24** | x | 1.01 | -101.12 | -100.09 | 99.45 | 102.17 |
| | y | 1.09 | -102.12 | 99.21 | 101.21 | -101.01 |
| | z | 224.87 | 425.21 | 425.01 | 25.01 | 25.31 |
| **25** | x | 2.08 | -102.10 | -99.97 | 98.12 | 101.97 |
| | y | 1.08 | -99.87 | 99.57 | 105.21 | -102.12 |
| | z | 224.16 | 425.18 | 426.01 | 24.96 | 24.98 |
| **26** | x | 1.92 | -98.12 | -100.05 | 103.01 | 101.78 |
| | y | 0.97 | -101.21 | 99.54 | 99.17 | -101.12 |
| | z | 225.13 | 425.09 | 425.12 | 25.13 | 24.95 |
| **27** | x | 0.98 | -97.91 | -98.96 | 101.12 | 100.98 |
| | y | 0.96 | -99.78 | 99.87 | 102.12 | -100.01 |
| | z | 226.01 | 425.12 | 425.07 | 25.97 | 24.96 |
| **28** | x | 0.95 | -100.01 | -100.12 | 99.84 | 101.78 |
| | y | 0.87 | -101.45 | 98.15 | 99.87 | -100.00 |
| | z | 225.12 | 425.01 | 425.09 | 25.01 | 24.91 |
| **29** | x | 0.01 | -100.02 | -99.12 | 101.23 | 102.01 |
| | y | 1.09 | -99.98 | 101.15 | 101.12 | -100.01 |
| | z | 224.91 | 425.12 | 425.00 | 25.09 | 24.90 |
| **30** | x | 0.02 | -100.12 | -99.57 | 102.10 | 100.01 |
| | y | 1.01 | -101.95 | 102.17 | 99.68 | -100.12 |
| | z | 224.01 | 425.09 | 425.01 | 24.98 | 25.09 |

De los datos obtenidos en los ciclos del ensayo realizado establecido en la norma ISO 9283, se puede observar la repuesta del robot delta en los tres ejes. Para el análisis se utilizó la dispersión como herramienta para visualizar gráficamente los valores del error en posicionamiento de sus ejes coordenados en los puntos programados.

En la Figura 3.3 se presenta el comportamiento del error del robot en el eje X; tomando en cuenta la baja frecuencia de los valores de 3,00 mm y -2,00 mm se descartan del análisis. El error en el eje X está comprendido entre -1,33 mm y 1,21 mm.

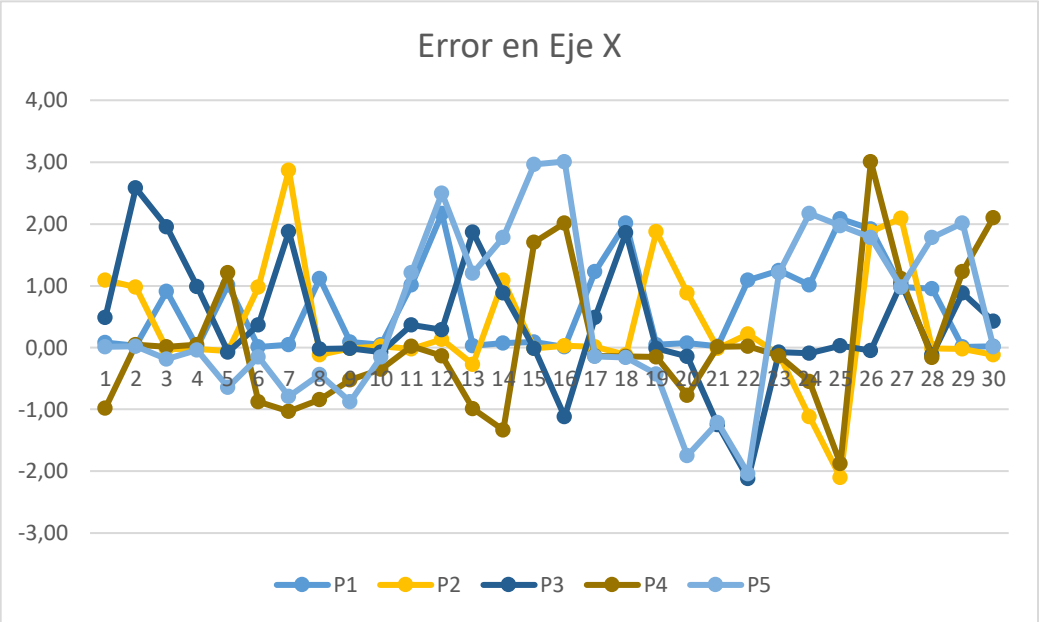


**Figura 3.3.** Resultados de error de precisión en el eje X.
Fuente: Propia

En la Figura 3.4 se presenta el comportamiento del error del robot en el eje Y; tomando en cuenta la baja frecuencia de los valores de 5,00 mm y -3,29 mm se descartan del análisis. El error en el eje X está comprendido entre -1,88 mm y 1,99 mm.

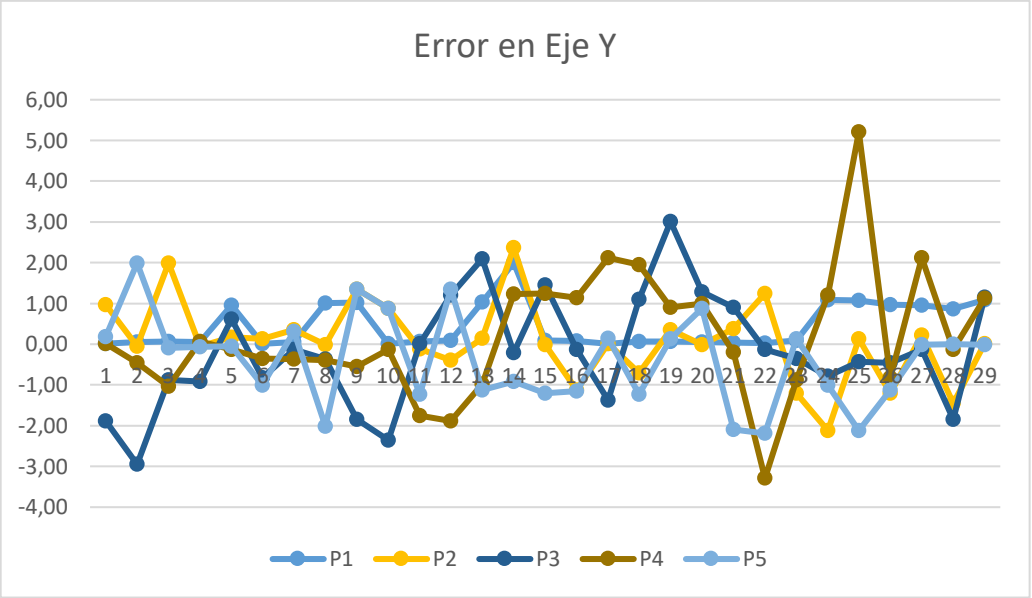


**Figura 3.4.** Resultados de error de precisión en el eje Y.
Fuente: Propia

En la Figura 3.4 se presenta el comportamiento del error del robot en el eje Z; tomando en cuenta la baja frecuencia de los valores de 1,78 mm y -1,92 mm se descartan del análisis. El error en el eje X está comprendido entre -1,09 mm y 1,12 mm.

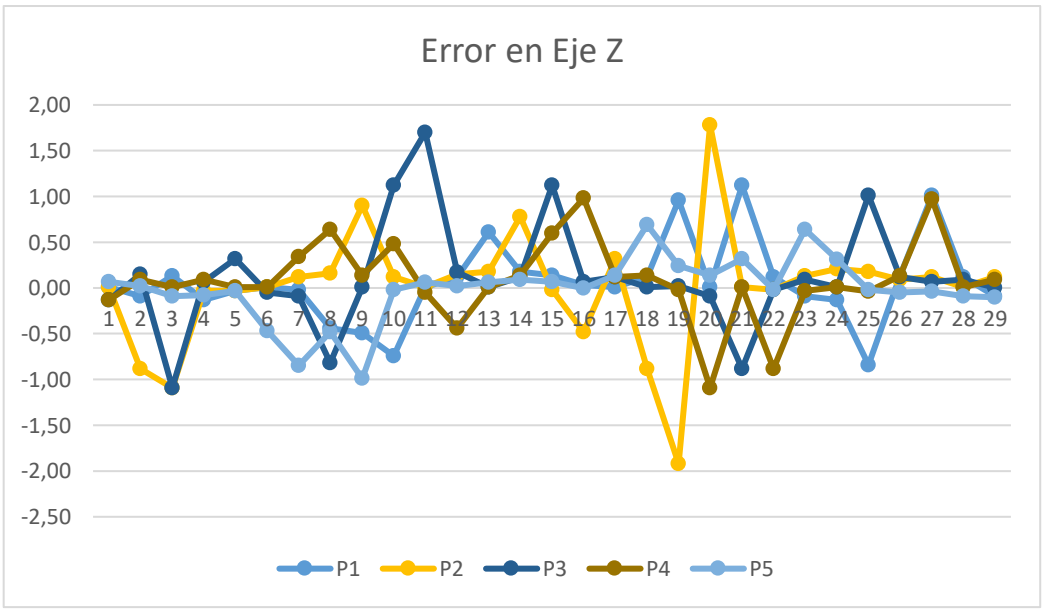


**Figura 3.5.** Resultados de error de precisión en el eje Z.
Fuente: Propia

**Precisión de posición (AP).**

La precisión de posición es la desviación entre las posiciones programadas y las posiciones alcanzadas por el robot simulado, cuando se aproxima siempre en la misma dirección, al punto programado.

**Precisión de posicionamiento:** Es la diferencia entre la posición de un punto programado y la posición alcanzada. Esto se calcula mediante la

$$AP_P = \sqrt{(\bar{x} - x_c)^2 + (\bar{y} - y_c)^2 + (\bar{z} - z_c)^2} \quad (3.1)$$

$$AP_x = (\bar{x} - x_c) \quad (3.2)$$

$$AP_y = (\bar{y} - y_c) \quad (3.3)$$

$$AP_z = (\bar{z} - z_c) \quad (3.4)$$

con

$$\bar{x} = \frac{1}{n}\sum_{j=1}^{n} x_j \quad (3.5)$$

$$\bar{y} = \frac{1}{n}\sum_{j=1}^{n} y_j \quad (3.6)$$

$$\bar{z} = \frac{1}{n}\sum_{j=1}^{n} z_j \quad (3.7)$$

Donde:

| | |
|---|---|
| $AP_P$ | Precisión de posición en el punto. |
| $AP_x, AP_y \; y \; AP_z$ | Precisión de posición en los ejes X, Y y Z. |
| $\bar{x}, \bar{y}$ y $\bar{z}$ | Coordenadas promedio de la nube de puntos obtenidos tras repetir la misma posición n veces. |
| $x_c, y_c$ y $z_c$ | Coordenadas de los puntos programados. |
| $x_j, y_j$ y $z_j$ | Coordenadas de la j-ésima posición alcanzada. |

Una vez que se dispone de las fórmulas con que se calcula la precisión, se procede a capturar los datos desde el simulador según lo indicado en el ensayo, obteniendo los siguientes resultados.

**Tabla 3.4.** Resultados de precisión.

| Posicionamiento (mm) | | | | |
|---|---|---|---|---|
| **Posición** | **Precisión** | | | |
| | **APx** | **APy** | **APz** | **APp** |
| **P1** | 0.648 | 0.468 | 0.025 | 0.468 |
| **P2** | 0.333 | 0.010 | 0.005 | 0.011 |
| **P3** | 0.375 | -0.100 | 0.108 | 0.147 |
| **P4** | 0.053 | 0.190 | 0.077 | 0.205 |
| **P5** | 0.520 | -0.380 | -0.012 | 0.380 |
| | | | **Promedio** | 0.242 |

Los datos de precisión de cada punto en los tres ejes coordenados establecieron un valor promedio de 0.242 mm. En el ensayo realizado a un robot industrial de seis ejes según la norma UNE-EN ISO 9283 [21] obtiene la precisión de posicionamiento para los cinco puntos:

- AP5 = 0.20618373 mm. = 206 µm.
- AP4 = 0.49666813 mm. = 496 µm.
- AP3 = 0.34739395 mm. = 347 µm.
- AP2 = 0.28490163 mm. = 284 µm.
- AP1 = 0.31512692 mm. = 315 µm.

Con un promedio de precisión de posicionamiento de 330 µm equivalente a 0.33 mm, se puede indicar que la precisión obtenida de 0.242 mm con el controlador implementado en Orocos es aceptable.

### 3.1.3 Pruebas en trayectoria.

Para las pruebas del robot paralelo tipo delta simulado se utilizará una trayectoria en el espacio cartesiano:

- Circular con un radio de operación de 0.25 metros.

La trayectoria en el espacio de las articulaciones se generó con movimientos punto a punto. En la Figura 3.6 se puede observar la trayectoria circular generada, que se usarán para guiar la posición cartesiana del efector final del robot.

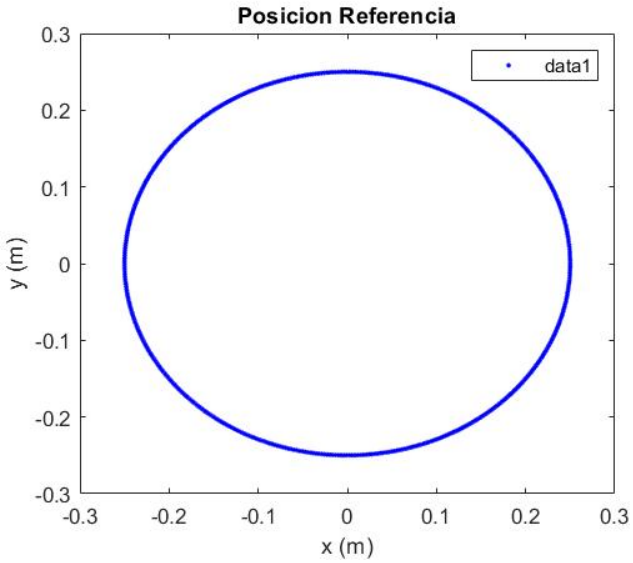


**Figura 3.6.** Trayectoria circular generada.
Fuente: Propia

### 3.1.3.1 Datos de posición angular

Los datos adquiridos en la tarjeta de desarrollo y el ordenador utilizando Matlab, fueron almacenados en archivos de texto para poder ser manejados en la aplicación diseñada para su análisis, siendo un total de 500 puntos a compararse.

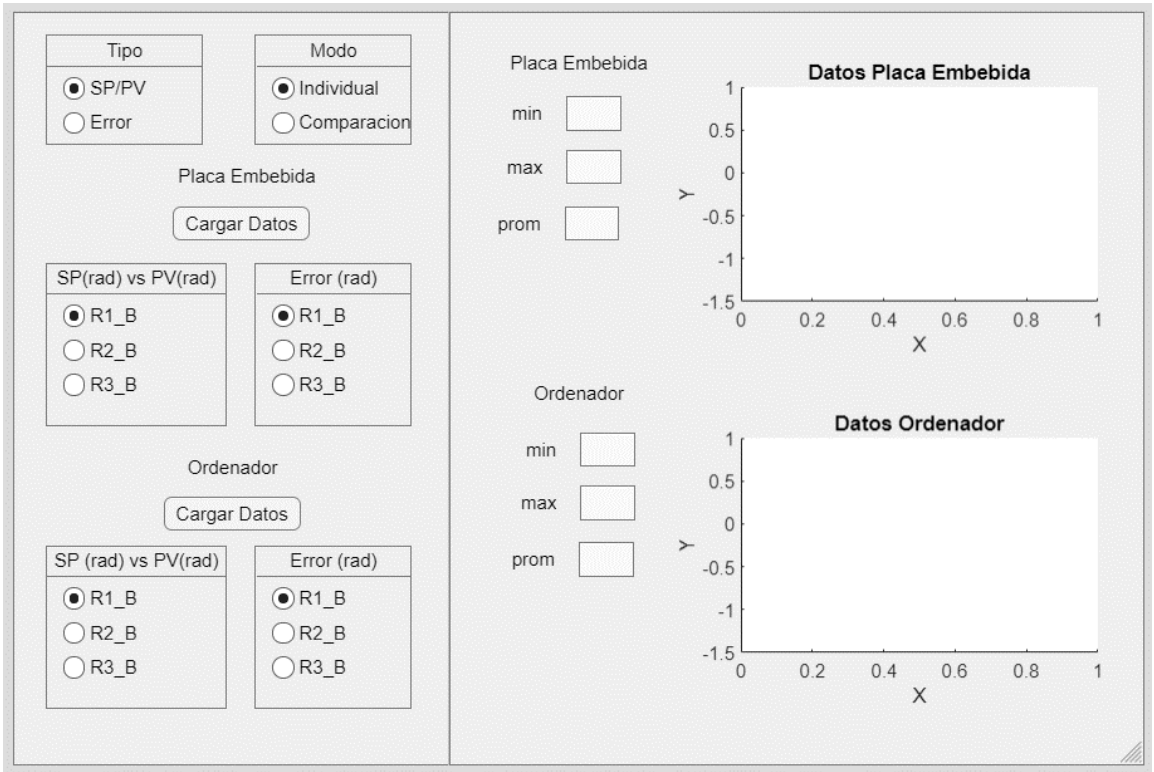


**Figura 3.7.** Aplicación para realizar análisis de datos.
Fuente: Propia.

Como se puede apreciar en la (Figura 3.8), al seleccionar el modo individual de la aplicación se podrá observar los datos de posición angular alcanzados PV respecto a la referencia SP en las tres articulaciones R1_B, R2_B y R3_B.

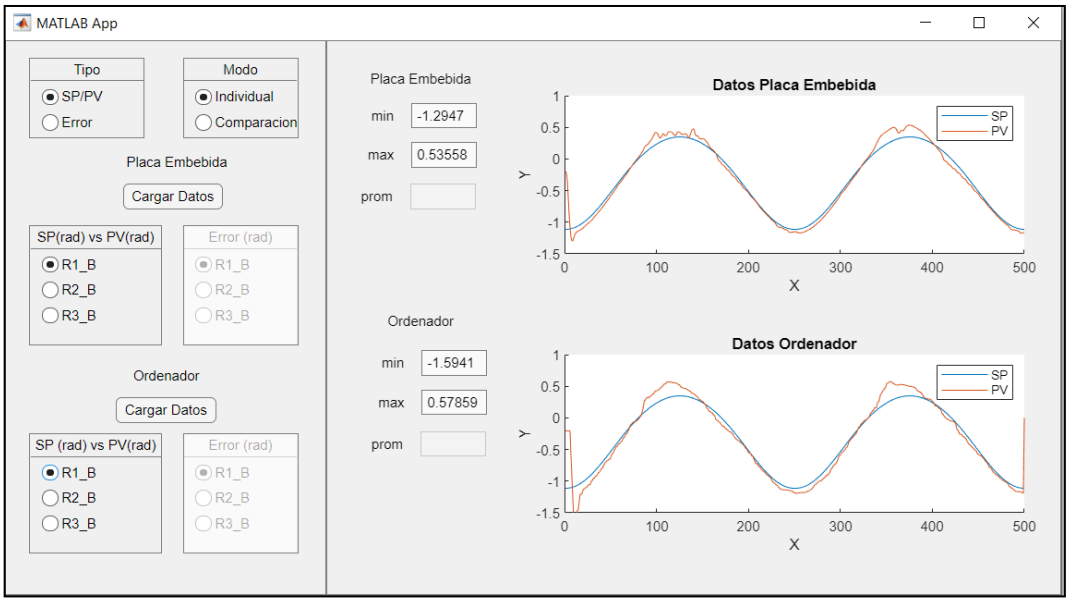


**Figura 3.8.** Representación del SP vs PV en la articulación 1.
Fuente: Propia.

De igual manera, la aplicación permite observar la representación del error a lo largo de los puntos muestreados, obteniendo para la articulación R1_B en la placa embebida un error promedio de -0.00510 y -0.00638 para el ordenador.

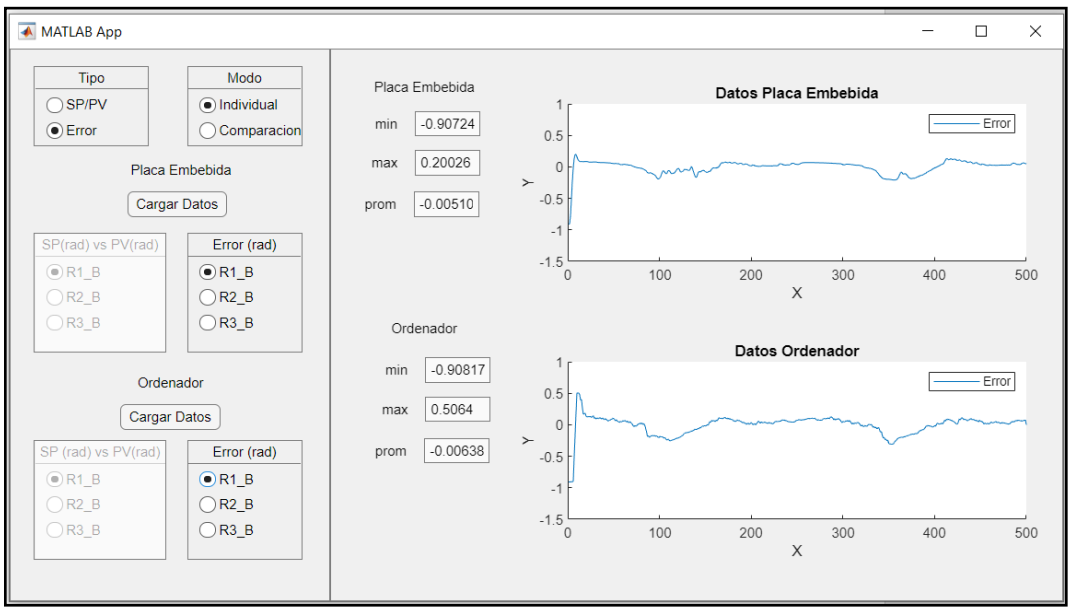


**Figura 3.9.** Representación del error en la articulación 1.
Fuente: Propia.

De la misma manera, para la articulación R2_B en la placa embebida se obtiene un error promedio de -0.001912 y -0.01438 para el ordenador.

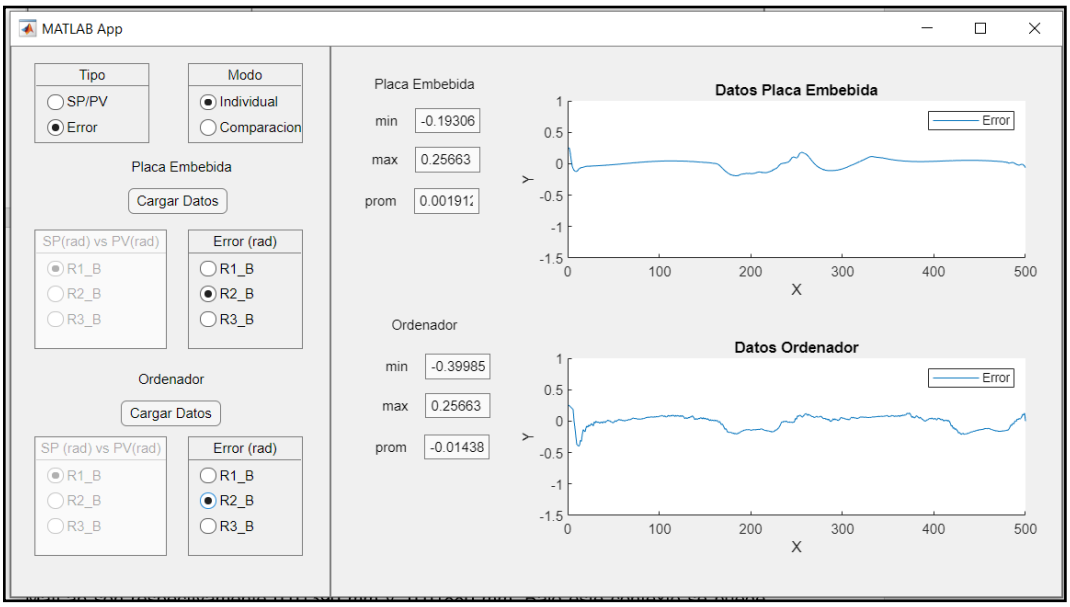


**Figura 3.10.** Representación del error en la articulación 2.
Fuente: Propia.

Igualmente, para la articulación 3 en la placa embebida se obtiene un error promedio de -0.01407 y 0.00378 para el ordenador.

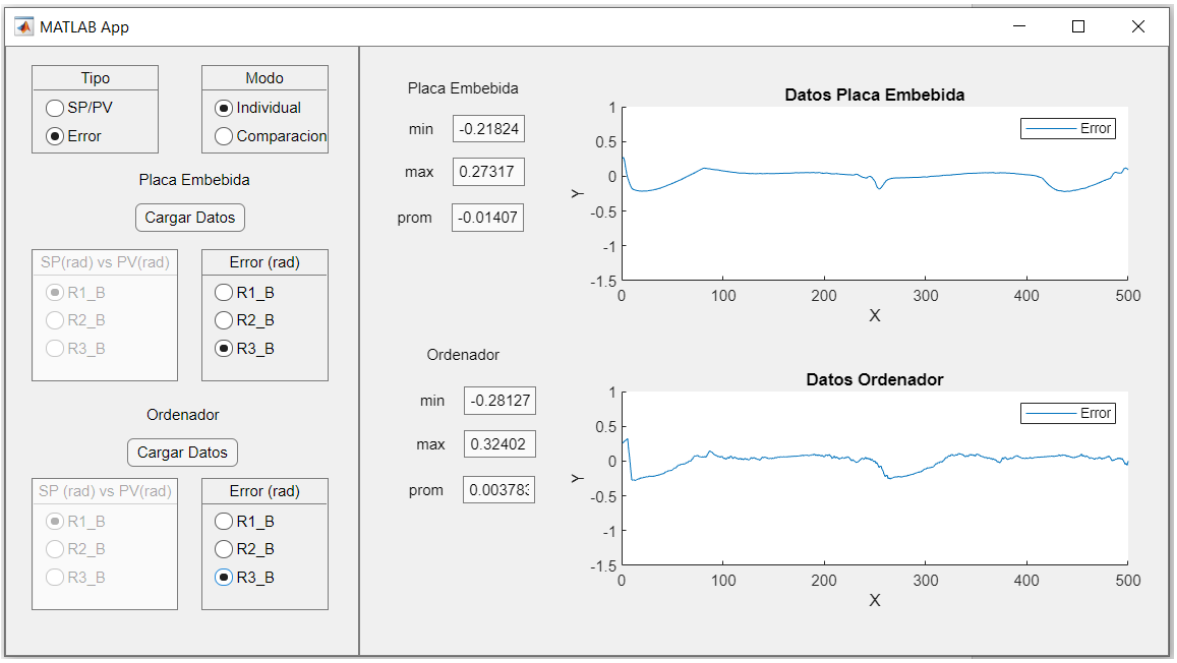


**Figura 3.11.** Representación del error en la articulación 3.
Fuente: Propia.

Las Figuras de trayectoria (3.9 - 3.11) del robot delta indica comportamiento del error de posición por el controlador PID, la cual fue implementado en Orocos y en el ToolBox de Matlab. Así mismo, se puede apreciar que el error promedio angular de posición en las articulaciones R1_B y R3_B de la placa embebida con Orocos es menor al error obtenido utilizando MatLAB con ROS ToolBox, evidenciando; que el controlador implementado en el paquete RTT de Orocos presenta un mejor desempeño.

### 3.1.3.2 Datos de posición cartesiana

Como parte del análisis de datos, se realiza una comparación de las posiciones cartesianas alcanzadas por el robot delta sobre las posiciones cartesianas de la trayectoria deseada, para lo cual se han almacenado 500 puntos en la tarjeta de desarrollo y en el ordenador.

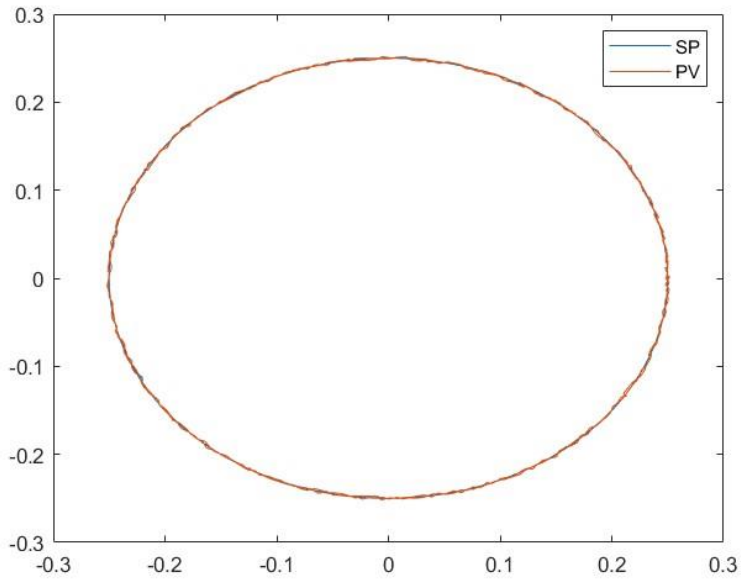


**Figura 3.12.** Posición cartesiana SP vs PV en la tarjeta de desarrollo.
Fuente: Propia.

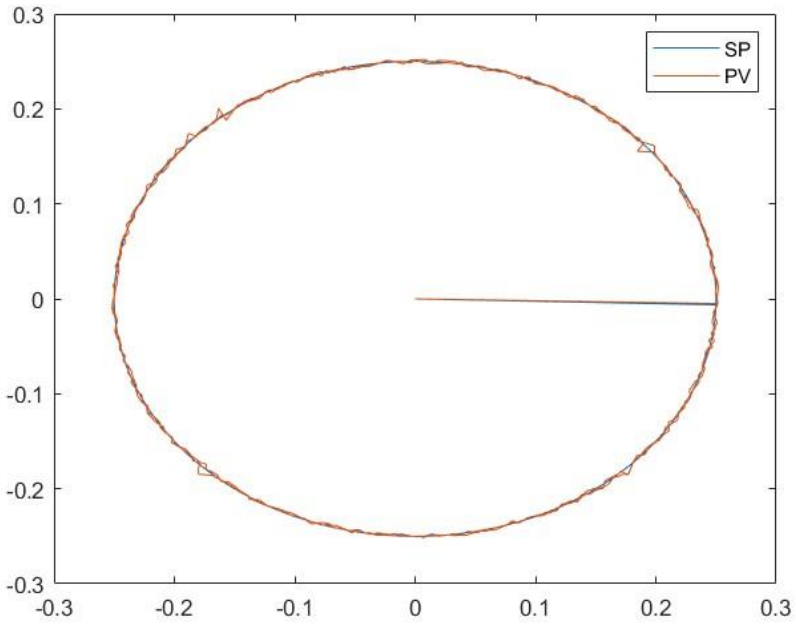


**Figura 3.13.** Posición cartesiana SP vs PV en ordenador.
Fuente: Propia.

En la Figura 3.12 y Figura 3.13 se puede apreciar el seguimiento del punto de referencia ubicado en la plataforma móvil del robot delta sobre la trayectoria planificara con una secuencia de puntos; con estos datos se calculó los errores promedio en los tres ejes coordenados detallados a continuación:

**Tabla 3.5.** Error promedio posición cartesiana en tres ejes.

| | **Error (m)** | | |
|---|---|---|---|
| **Ejes** | **x** | **y** | **z** |
| Tarjeta de desarrollo | 0.00003019 | -0.00003620 | 0.016124 |
| Ordenador | -0.00010273 | -0.00001272 | 0.026404 |

### 3.1.3.3 Índice de desempeño

Para analizar el desempeño del controlador se utilizarán métricas que ayuden a medir cuantitativamente el error [22]. En la Tabla 3.6 se puede apreciar que los errores de posición cartesiana obtenidas con el controlador de posición implementado son pequeños, por ello; se opta analizar el desempeño utilizando las métrica del Error Absoluto Máximo (EAM), Error Absoluto Medio (MAE) y la raíz cuadra del error cuadrático medio (RMSE), no se seleccionó el error cuadrático medio (MSE) debido a que el error es pequeño y este métrica da mucho peso a los errores grandes para conocer su comportamiento. Así mismo, se opta utilizar la RMSE para ver el error en las unidades de las posiciones.

**Tabla 3.6.** Métrica de medición del Error utilizando OROCOS en tarjeta de desarrollo embebida.

| | EAM [mm] | MAE | RMSE [mm] |
|---|---|---|---|
| Error_pos_X | 1.991 | 0.993 | 1.147 |
| Error_pos_Y | 1.995 | 1.009 | 1.164 |
| Error_pos_Z | 46.187 | 20.571 | 25.50 |

**Tabla 3.7.** Métrica de medición del Error utilizando Matlab – Toolbox ROS

| | EAM [mm] | MAE | RMSE [mm] |
|---|---|---|---|
| Error_pos_X | 9.544 | 1.30 | 1.63 |
| Error_pos_Y | 9.335 | 1.31 | 1.69 |
| Error_pos_Z | 453.266 | 28.49 | 63.59 |

La Tabla 3.6 y Tabla 3.7 indican el desempeño del sistema de control implementando en la tarjeta de desarrollo con Orocos y Matlab. Se puede apreciar que el controlador implementado en Matlab tiene un EAM en el eje Z mayor al EAM del eje Z controlado con Orocos, es decir que tiene un pico máximo de error correspondiente a 407.07mm de diferencia. De la misma manera, el RMSE obtenido por el control implementado en Matlab, tiene una diferencia de 38.09mm.

El error absoluto medio MAE nos indica la métrica de todos los residuos correspondientes a la trayectoria, se puede observar en Tabla 3.6 la precisión correspondiente para cada eje con sus respectivos controles. La precisión del control desarrollado en la tarjeta tiene mejor precisión en comparación al control de Matlab.

# 4 CONCLUSIONES Y RECOMENDACIONES

## 4.1 CONCLUSIONES

- Del estudio bibliográfico de artículos científicos y tesis, se puede analizar el estado actual del uso del middleware OROCOS aplicado para el control de robots, siendo el punto de partida que permitió aclarar los conceptos para planificar el diseño, simulación e implementación del sistema.

- La simulación del robot VREP se logró utilizando los parámetros físicos y geométricos del modelo mecánico del robot delta, sin embargo, se requirió de la instalación de la interfaz de programación de aplicaciones API para interactuar con el nodo de comunicación ROS donde se enlaza con el controlador de OROCOS.

- Para este proyecto la librería de OROCOS RTT (Real Time Toolkit) garantizó la ejecución periódica del controlador PID de posición implementado en la tarjeta de desarrollo embebida, lo que conllevo a un índice de desempeño EAM (Error Absoluto Máximo) en x:1.991, y:1.995, z:46.187 menor a x:9.544, y:9.335, z:453.266 del controlador implementado en Matlab.

- En este proyecto, se propone un sistema de control para un robot delta invertido con la finalidad de realizar un análisis comparativo de los errores angulares, posición cartesiana e índices de desempeño utilizando el middleware OROCOs con la tarjeta embebida de desarrollo y Matlab utilizando el ROS Toolbox ejecutado en un ordenador, de lo cual se pudo determinar que se tuvo un mejor índice de desempeño con OROCOS. Además, con el controlador implementado en OROCOS se realizó un ensayo y análisis de las prestaciones para el robot según la norma UNE-EN ISO 9283 para determinar la exactitud en posicionamiento.

- La precisión de control de posición para el robot delta desde el controlador implementado con Orocos en la tarjeta embebida tiene mejor desempeño en comparación al control ejecutado con Matlab. Las métricas de medición de error calculadas prueban el desempeño del control.

## 4.2 RECOMENDACIONES

- Este proyecto deja planteado las matemáticas del control y del robot, siendo pilares fundamentales para el perfeccionamiento del sistema de control, indudablemente se recomienda a futuro experimentar con la tarjeta y cuantificar su comportamiento con cada complejidad de control desarrollado, desde un PID robusto hasta un

control utilizando Inteligencia Artificial, esta medida ayudará a futuro a conocer el límite del middleware OROCOS tanto en control como comunicación hacia el simulador V-REP utilizando una tarjeta de desarrollo embebido, además, ayudará a mejorar sus conocimientos en Linux, ROS y OROCOS.

- Se conoce a base de otros experimentos que utilizar un entorno virtual ayuda a diseñar un futuro robot, teniendo en cuenta esto, se recomienda realizar una tesis para desarrollar un robot y experimentar el sistema de control desarrollado con un prototipo real.

# 5 REFERENCIAS BIBLIOGRÁFICAS


[1] J. I. Cazalilla, “Implementación basada en el middleware OROCOS de controladores dinámicos para un robot paralelo,” Universidad Politécnica de Valencia, 2012.

[2] M. Vallés, J. I. Cazalilla, Á. Valera, V. Mata, and Á. Page, “Implementación basada en el middleware OROCOS de controladores dinámicos pasivos para un robot paralelo,” *Revista Iberoamericanda de Automática e Informática industrial 10*, pp. 96–103, 2013, [Online]. Available: http://dx.doi.org/10.1016/j.riai.2012.11.009

[3] E. Estévez, A. S. García, J. G. García, and J. G. Ortega, “Aproximación Basada en UML para el Diseño y Codificación Automática de Plataformas Robóticas Manipuladoras,” *RIAI - Revista Iberoamericana de Automatica e Informatica Industrial*, vol. 14, no. 1, 2017, doi: 10.1016/j.riai.2016.11.001.

[4] D. H. Martillo and E. L. Zambrano, “Diseño de aplicaciones de sistemas embebidos basados en tecnología raspberry-pi y odroid-u3,” Universidad Politécnica Salesiana, 2015.

[5] J. Coronado, “Desarrollo de aplicaciones embebidas de control en robots móviles,” Universidad Politécnica de Valencia, 2013.

[6] A. Elkady and T. Sobh, “Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography,” *Journal of Robotics*, vol. 2012, 2012, doi: 10.1155/2012/959013.

[7] P. Soetens, T. Issaris, H. Bruyninckx, S. Joyeux, and R. Smits, “Orocos Project documentacion,” *Orocos Project documentation*, 2020. https://docs.orocos.org/ (accessed Aug. 31, 2021).

[8] J. I. Cazalilla, “Diseño e implementación de un sistema de control de robots mediante la ingeniería del software basada en componentes. Aplicación a un robot paralelo de 3DOF,” Universidad Politécnica de Valencia, 2017.

[9] E. (Grupo de investigación del L. de M. y R. Ceballos, M. (Grupo de investigación del L. de M. y R. Rodriguez, J. L. (Grupo de I. B. Paredes, and P. (Departamento de T. y D. Vargas, “Desarrollo de un robot de rehabilitación pasiva para la articulación de la muñeca mediante la implementación de un microcontrolador Arduino UNO,” 2016.

[10] J. Camarero, “Mejora de la Funcionalidad de ROS mediante el Middleware DDS,” Madrir, 2015. [Online]. Available: https://www.researchgate.net/publication/282947850

[11] MathWorks, “Ros Toolbox.” https://www.mathworks.com/products/ros.html (accessed Apr. 29, 2022).

[12] M. Díaz *et al.*, “Aplicación de los Robots Paralelos To cite this version : HAL Id : hal-01907282,” p. 27, 2018.

[13] R. P. Foundation, “Raspberry,” *Raspberry Pi*, 2021. https://www.raspberrypi.org/ (accessed Sep. 30, 2021).

[14] BeagleBoard. org Foundation, “BeagleBoard.” https://beagleboard.org/black (accessed Apr. 17, 2022).

[15] P. Cárdenas. Peña, C., E. Martínez, “Optimización dimensional de un robot paralelo tipo delta basado en el menor consumo de energía,” *Ciencia e Ingeniería Neogranadina*, vol. 21, pp. 73–88, 2011.

[16] Y. Lou, G. Liu, and Z. Li, “A General Approach for Optimal Design of Parallel Manipulators,” *IEEE Transactions on Automation science …*, vol. X, no. X, pp. 1–16, 2005.

[17] E. Courteille, D. Deblaise, and P. Maurine, “Design optimization of a delta-like parallel robot through global stiffness performance evaluation,” *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, no. November 2009, pp. 5159–5166, 2009, doi: 10.1109/IROS.2009.5353906.

[18] F. Álvarez, “Diseño y simulación de un controlador para un péndulo invertido virtual ejecutando el middleware Orocos e implementación del controlador en una raspberry pi con comunicación al simulador,” Escuela Politécnica Nacional, Quito, 2019.

[19] C. Pardo, “Controlador PID,” *https://www.picuino.com/es/control-pid.html*, 2013.

[20] J. Calvo, R. Ferreiro, Á. Alonso, A. Piñon, F. Pérez, and H. Alaiz, “METODO GRÁFICO DIDÁCTICO PARA MATLAB DE AJUSTE DE REGULADORES PID EN CADENA CERRADA.”

[21] H. Pardo, “Ensayo y Análisis de las prestaciones de un robot industrial de seis ejes según la norma UNE-EN ISO 9283,” 2010.

[22] S. Castaño, “Índices de Desempeño,” *https://controlautomaticoeducacion.com/control-realimentado/indices-de-desempeno/#Criterio_de_desempeno_ejemplo*.

# ANEXOS