DESARROLLO DE UNA INTERFAZ PARA LA TELEOPERACIÓN DE UN ROBOT PARALELO DELTA TIPO 3RRR

Trabajo Fin de Grado

Autor:
Luis Manuel Zambrano Caballero
Tutor/es:
Ph.D Pedro Fabián Cárdenas Herrera
Ph.D César Augusto Peña Cortés

18 de Junio de 2020

DESARROLLO DE UNA INTERFAZ PARA LA TELEOPERACIÓN DE UN ROBOT PARALELO DELTA TIPO 3RRR

Trabajo de Grado - Pasantía de Investigación

Autor
Luis Manuel Zambrano Caballero

Tutor/es
Ph.D Pedro Fabián Cárdenas Herrera
Universidad Nacional de Colombia
Ph.D César Augusto Peña Cortés
Universidad de Pamplona

Pamplona, 18 de Junio de 2020

Resumen

La siguiente pasantía de investigación da a conocer el análisis cinemático (directo e inverso) y el desarrollo de una interfaz para la manipulación de un robot paralelo Delta tipo 3RRR por teleoperación. La estructura del robot paralelo delta cuenta con una plataforma fija y una móvil que se unen mediante cadenas cinemáticas independientes, su configuración les otorga ventajas en condiciones de rigidez, velocidad, precisión e inercia en movimiento.

Se realizó un estudio detallado de ROS, de interfaces con Python, de la configuración de los servomotores *dynamixel* y de las herramientas matemáticas de robots paralelos tipo delta, para afianzar conocimientos.

Durante el desarrollo de la interfaz y herramientas de software para el comando y teleoperación del robot paralelo delta, se implementaron las interfaces tanto local como remota bajo el ambiente de ROS y Python. Se evidenció un inconveniente en la transmisión de datos, debido a la alta tasa de muestreo del robot falcon, saturando el canal de comunicación que bloqueaba a los servomotores. Adicionalmente se desarrolló un paquete que contiene toda la información necesaria para facilitar el acceso a los avances desarrollados durante la pasantía de investigación.

Esta pasantía de investigación fue desarrollada en la **Universidad Nacional de Colombia** bajo la asesoría del Ph.D Pedro Fabián Cárdenas Herrera.

Palabras Clave: ROS, Teleoperación, Robótica paralela, Interfaz, Robot *falcon*.

Abstract

The following research internship introduces kinematic analysis (direct and inverse) and the development of an interface for manipulating a Delta type 3RRR parallel robot by teleoperation. The structure of the parallel delta robot has a fixed and a mobile platform that are joined by independent kinematic chains, its configuration gives them advantages in conditions of rigidity, speed, precision and inertia in movement.

A detailed study of ROS, of interfaces, was carried out With Python, the configuration of the dynamixel servo motors and the mathematical tools of parallel delta robots, to strengthen knowledge.

During the development of the interface and software tools for the command and teleoperation of the parallel delta robot, the interfaces were implemented both Local as well as remote under the ROS and Python environment. A problem in data transmission was evident, due to the high sampling rate of the falcon robot, saturating the communication channel that blocked the servomotors. Additionally, a package was developed that contains all the necessary information to facilitate access to the advances developed during the research internship.

This research internship was developed at the National University of Colombia under the advice of Ph.D Pedro Fabián Cárdenas Herrera.

Key Words: ROS, Teleoperation, Parallel Robotics, Interface, Robot *falcon*.

Agradecimientos

Le agradezco a Dios, a mi tía Yaneth Muñoz, a mi abuela Carmen Díaz Villazón, a mis padres, familiares y compañeros de estudio. Siempre me brindaron su apoyo de manera incondicional, impulsándome siempre a seguir adelante. Gracias por confiar y creer en mi.

Al Ph. Dr. Pedro Fabián Cárdenas, Ph. Dr. César Peña, a los asesores de mi trabajo de grado, a la profesora Yara Oviedo y a los diferentes profesores de la **Universidad de Pamplona**; por su tiempo, por compartir sus conocimientos y ayudarme a crecer a nivel personal y profesional.

A todas aquellas personas que siempre me motivaron para poder cumplir esta meta. Por que a veces solo es necesario una conversación o una palabra para uno motivarse y creer en si mismo.

En lo profundo del inconsciente humano hay una gran necesidad de un universo lógico que tenga sentido. Pero el universo siempre está un paso más allá de la lógica.

Frank Herbert.

El hombre ignorante se maravilla de lo excepcional; el hombre sabio se maravilla de lo común; la mayor maravilla de todas es la regularidad de la naturaleza.

G. D. Boardman.

Aquel que no sabe y no sabe que no sabe es un idiota; evítalo. Aquel que no sabe y sabe que no sabe es un niño; edúcalo. Aquel que sabe y no sabe que sabe está dormido; despiértalo. Aquel que sabe y sabe que sabe es un sabio; síguelo.

Provervio Persa

Tres hombres son mis amigos: el que me estima, el que me detesta y al que le soy indiferente. El que me estima me enseña a apreciar; el que me detesta me enseña a protegerme; al que le soy indiferente me enseña a confiar en mí mismo.

J. E. Dinger

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1. Introducción

El campo de aplicación y estudio de la robótica paralela y de teleoperación esta teniendo unos avances sin precedentes en áreas como la industria, la medicina, el entretenimiento, entre otos campos. La robótica paralela y el concepto de teleoperación son un poco complejo, por eso el objetivo de este proyecto es acercar un poco estos conceptos a los estudiantes o interesados, sobre todo a aquellos que quieran irse por el campo de la robótica y quieran profundizar en esta área.

Una herramienta fundamental para la robótica y este proyecto, es ROS. Un *framework* que facilita la ejecución de proyectos robóticos. En la actualidad existen muchos entornos de desarrollo, lenguajes de programación, entre otras herramientas, lo cual genera una diversificación en la manera en que los desarrolladores abordan un proyecto relacionado con robótica, Debido a esta diversidad y que hay muchas herramientas y diferentes tipos de robots, se ha tenido la necesidad de crear un entorno o ambiente en donde todos los interesados en esta área puedan interactuar de manera más práctica. Es aquí donde aparece ROS. Un potente meta-sistema operativo que proporciona los instrumentos necesarios para la realización de proyectos robóticos.

Se comenzará con una breve introducción a ROS, sus comandos, métodos de comunicación y herramientas básicas, seguido de las especificaciones de los motores dynamixel MX-64, como configurarlos, controlarlos desde ROS y una breve descripción la tarjeta que se van a utilizar para la comunicación motores dynamixel MX-64 -PC , información que se encuentra en los anexos del libro. Teniendo los conocimientos básicos con respecto a ROS y los hardwares se abordó el proyecto, el cuál consistía de dos objetivos: el primero fue teleoperar el robot delta con las teclas direccionales del teclado y el segundo fue teleoperarlo con un robot háptico (Nonint – *Falcon*).

1.1. Justificación

En la actualidad la robótica paralela y la teleoperación se están convirtiendo en un tema de interés en el campo industrial y de investigación debido a su eficiencia y alta velocidad de trabajo, que reduce considerablemente los tiempos de ejecución de labores, sobre todo en aquellas que son repetitivas.

Es por ello que esta pasantía de investigación tiene como finalidad desarrollar una herramienta que permita la manipulación de un robot paralelo tipo delta mediante el ambiente de ROS y Python. Contribuyendo además en la ampliación de conocimientos respecto al tema ya que durante un curso básico no se alcanzan a profundizar. Conllevando a la necesidad de implementar metodologías que permitan abarcar temáticas de interés en la nueva generación de estudiantes investigadores.

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar una interfaz y un conjunto de herramientas de software para el comando y la teleoperación de un robot paralelo tipo Delta.

1.2.2. Objetivos Específicos

- Construir una base para el Robot Paralelo Delta.
- Analizar los modelos de cinemática directa e inversa del robot paralelo Delta 3RRR y la configuración de los servomotores para un mejor control.
- Realizar el algoritmo para la teleoperación local y la teleoperación remota, Mediante ROS y Python.
- Incorporar un dispositivo (Robot *Novint Falcón*) para la teleoperación remota del robot paralelo delta.
- Diseñar una interfaz que me permita comunicar y controlar al robot paralelo delta.

2. Estado del arte

La robótica es un área multidisciplinaria que ha tenido un gran auge en los últimos tiempos. Debido al acelerado crecimiento y a la alta demanda tecnológica que se ha venido presentando, surge la necesidad de crear sistemas inteligentes y autónomos que sean capaces de realizar tareas que resultan complejas y peligrosas para el ser humano. Los robots manipuladores son considerados como los primeros robots desarrollados, también llamados robots de primera generación, siendo estos mecanismos multifuncionales que se caracterizan por tener sistemas de control sencillos y realizar secuencias fijas o variables.

2.1. Historia de la Robótica

Desde tiempos memorables se tienen relatos de autómatas o mecanismos automáticos creados por el hombre. Pero, no fue sino hasta 1921 cuando por primera vez se incorpora el término *Robot*, en una obra llamada *Rossum's Universal Robots* escrita por el famoso escritor **Karel Capek** [2]. Etimológicamente el termino robot proviene de la palabra checa *robota*, que significa *trabajo forzado* o *esclavitud*, en la actualidad toda máquina capaz de realizar labores o acciones de manera autónoma o por teleoperación es considerada un robot.

Figura 2.1: Gallo de Estraburgo.
Fuente: Pajares Correa.

La *Clepsidra, órgano de agua* de Ctesibius, año 270 a.C. y el *Teatro automático* de Herón de Alejandría año 62 d. C. son de los primeros autómatas o mecanismos de los cuáles se tienen conocimientos, el primero se utilizaba para la producción de los primeros relojes y órganos de agua, el segundo consistía de un teatro automático donde las figuras cambiaban de posición, los pájaros cantaban, entre otros movimientos, ambos eran de aplicación neumática. De los autómatas antiguos que aún se conservan están, el *Gallo de la catedral de Estraburgo* de origen desconocido año 1352, (Figura 2.1) [3]. El caballero mecánico y el León mecánico de Leonardo Da Vinci (Figura 2.2) que se encuentran en el museo de ciencia de Florencia [4].

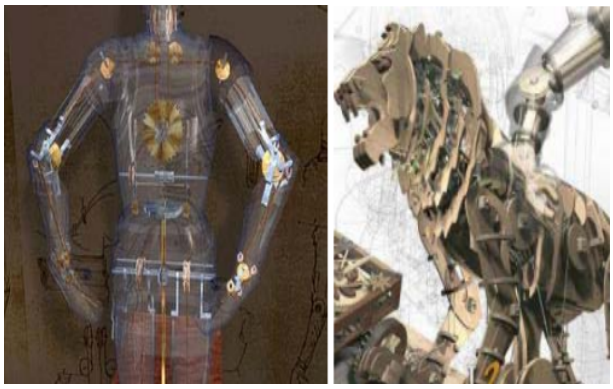


Figura 2.2: Caballero Mecánico y León Mecánico de Leonardo Da Vinci.
Fuente: Cerveró Meliá et al.

Existen muchas definiciones para el término robot, La más aceptada es la definición proporcionada por la **RIA** (*Robot Institute of America*, por sus siglas en inglés): *"Un robot es un manipulador multifuncional reprogramable diseñado para mover herramientas, piezas, materiales o dispositivos, a través de movimientos programados para la ejecución de un sin número de tareas*" [5].

Hoy en día existe una gran variedad de robots, lo que ha hecho que su clasificación sea compleja, la Tabla 2.1 muestra una descripción general de como se pueden clasificar los robots:

Tipos de Robots		
Móviles	Terrestres, Acuáticos, Aéreos	AIBO
Humanoides	Diseño Complejo	ASIMO, SOPHIA
Industriales	Brazos Robóticos	SCARA, PUMA

Tabla 2.1: Clasificación de los Robots

.

El término robótica fue utilizado por primera vez por el escritor Isaac Asimov en una obra llamada *Yo Robot* en 1942, estableciendo las leyes de la robótica [6] [7]:

- Un robot no hará daño a un ser humano o por su inacción permitir que un humano sufra daños.
- Un robot debe cumplir con todas las órdenes dadas por un humano, pero no aquellas que entren en conflicto con la primera ley.
- Un robot debe proteger su propia existencia, siempre y cuando no entre en conflicto con la primera y segunda ley.

Los avances en la robótica se han visto influenciados por la alta demanda y rápido crecimiento tecnológico que sé está viviendo desde mediados del siglo XX hasta la actualidad. Algunos avances han sido:

- En 1937 se construyó el primer robot industrial accionado por un motor eléctrico.
- En 1956 *Unimation*, es la primera empresa en construir un robot.[8]
- En 1961 Victor, desarrolla un robot de 6 ejes, llamado robot de *Stanford*, (Figura 2.3). [9]

Figura 2.3: Robot Stanford
Fuente: Burgar et al..

- Con la revolución industrial se reprodujeron en masas sistemas automáticos inteligentes, que desplazaron al hombre de sus labores, lo cual ha atraído problemas éticos.
- En la década de los 70 muchas empresas entraron al campo de la robótica. Entre ellas, *General Electric y General Motors.*

- En 1978 se presenta al Robot SCARA (Figura 2.4), Construido por el profesor Hiroshi Makino de la universidad Yamanashi en Japón, donde introdujo un nuevo concepto. [10]

Figura 2.4: Robot *SCARA*.
Fuente: Izaguirre et al.

- En 1973 una empresa alemana, construyó el primer robot electromecánico, *FAMULUS*.
- En la década de los 80 se comienzan a utilizar microcontroladores, reduciendo los altos costos que demandaban los materiales y equipos, disminuyendo consideradamente el tamaño dimensional.
- En 1985 el robot *PUMA 560*, fue utilzado para poder incrustar una aguja en el celebro, en 1988 el robot World First extrajo fragmentos de tejido humano, (figura 2.5) [11].

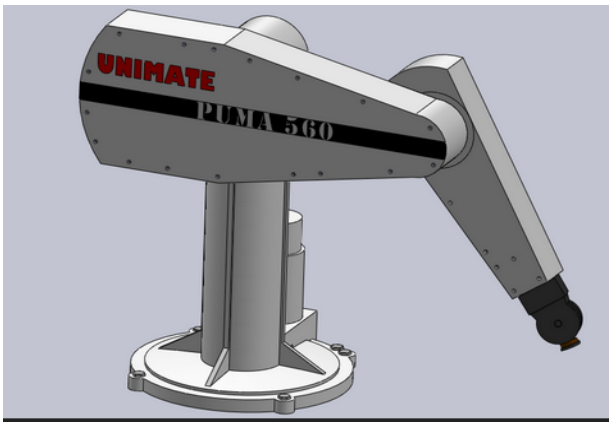


Figura 2.5: Robot PUMA.
Fuente: Vidal and Laura.

- En el 2010. Honda lanza la última versión de su robot ASIMO, un humanoide capaz de realizar tareas domésticas (Figura 2.6), al igual que *Romba*. [13]

Figura 2.6: Robot Humanoide *ASIMO*.
Fuente: Ng-Thow-Hing et al.

- En el 2015 se presenta a *Sophia*, una humanoide capaz de realizar gestos y articular una conversación básica (Figura 2.7). [14]

Figura 2.7: Robot Humanoide SOPHIA.
Fuente: Barón and Pinilla.

2.2. Robots Manipuladores

Existen dos tipos de robots manipuladores, los robots seriales y los robots paralelos (Figura 2.8). Los robots seriales son los más utilizados en la industria, presentan una configuración sencilla, donde sus articulaciones se unen de manera secuencial empezando por la base hasta el efector final del robot, para ir desde la base hasta el efector

final del robot solo hay un camino [16]. Los robots paralelos son unos mecanismos de cinemática cerrada, donde su base se une con una plataforma móvil por medio de varios caminos, cada uno con cinemáticas independientes, su estudio y análisis es más complicado que el de los robots seriales.

Figura 2.8: Robots: Serial vs Paralelo.
Fuente: Autor.

2.3. Robots Seriales

Es la configuración más común para robots industriales y están diseñados como una serie de eslabones conectados consecutivamente, teniendo una estructura análoga a un brazo humano, comenzando por la base hasta llegar al efector final (Figura 2.9).

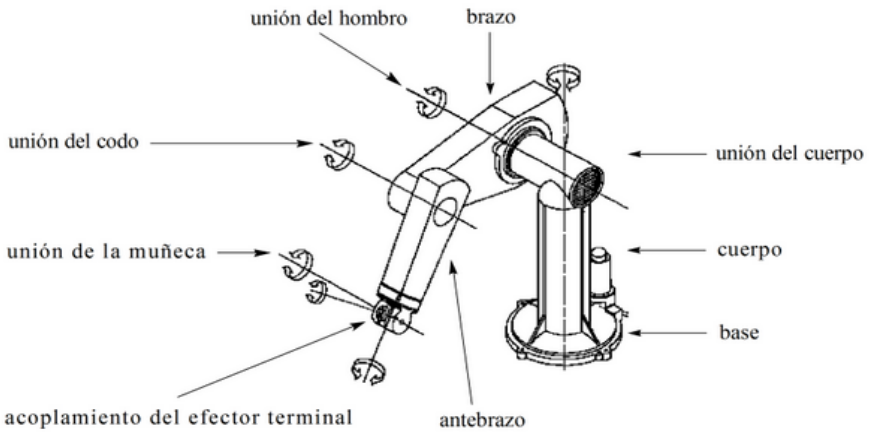


Figura 2.9: Estructura de un robot serial.
Fuente: Tsai.

2.3.1. Configuraciones más Frecuentes

- **Robot Cartesiano:** Tiene 3 articulaciones prismáticas, precisión elevada, velocidad alta y espacio de trabajo amplio.
- **Robot Cilíndrico:** Tiene 1 articulación rotacional y 2 prismáticas.
- **Robot Esférico o polar:** Tiene 2 articulaciones rotacionales y 1 prismática.
- **Robot SCARA:** Tiene 2 articulaciones rotacionales y 1 prismática. Para tareas en la dirección del eje vertical
- **Robot Antropomórfico:** Tiene 3 articulaciones rotacionales. Es el más versátil y su espacio de trabajo es una esfera.

2.3.2. Elementos de un Robot Serial

Las principales partes que conforman un robot serial, también conocido como manipulador serial son:

- Sensores.
- Controlador.
- Efector final o herramienta.
- Elementos motrices o actuadores.

Controlador

Son los dispositivos encargados de realizar los cálculos, procesar la información y regular los movimientos del robot.

Sensores

Son los que permiten recibir información, para luego procesarlo y hacer que el robot realice una función en específico:

- **Presencia:** Inductivo, Capacitivo, Efecto Hall, ultrasonido, óptico.
- **Posición:** Potenciómetros, Resolver, Encoders.
- **Velocidad:** Tacogeneradores.
- **Fuerza:** Galgas extensiométricas.

Actuadores

- **Neumáticos e Hidráulicos:** Cilindros y motores.
- **Eléctricos:** motores DC, AC y Motores paso a paso.

Efector final

Es el elemento que interacciona con el entorno:

- Electroimán, ventosa, pinzas de agarre y pinza de presión (al vacio).

2.4. Historia de los Robots Paralelos

Mucho antes de que se comenzará a hablar de robots paralelos, aparecieron los primeros indicios y obras relacionadas con las estructuras mecánicas de los robots paralelos. Uno de los primeros que se las ingenió para construir una plataforma de movimiento paralela fue James E. Gwinnett. El mecanismo fue estructurado, pero nunca fue culminado [17]. En 1942 W. L. V Pollard patentó un robot paralelo (*Position- Controlling-Apparatus*), este sistema se diseñó para pintar los automóviles (Figura 2.10) [18]. Gough y Stewart diseñaron un robot paralelo, la plataforma es conocida como *Stewart*, tiene 6 grados de libertad y fue diseñada como simulador de vuelo. Este robot fue el primer prototipo de mecanismo paralelo con seis actuadores prismáticos e hidráulicos (Figura 2.11) [19]. En 1967 Cappel diseño y construyó un simulador de vuelo utilizando el mismo principio de la plataforma de Gough, con un sistema y diseño más eficiente (Figura 2.12) [20].

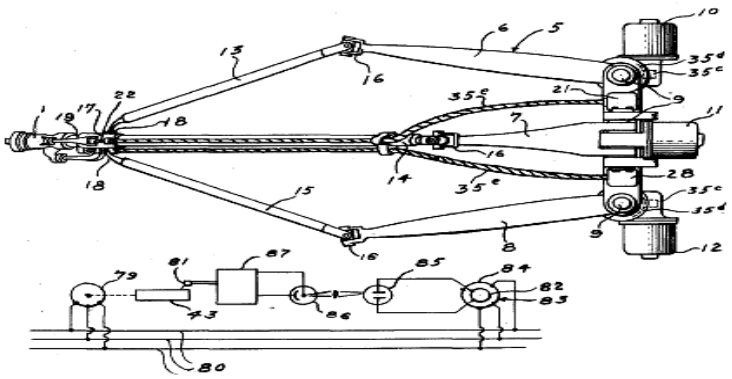


Figura 2.10: Prototipo del robot paralelo de V. Pollard.
Fuente: Gudiño-Lau et al.

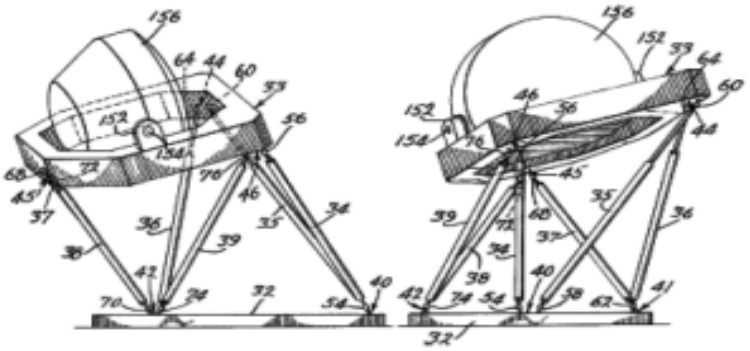


Figura 2.11: Simulador de vuelo, plataforma Steward.
Fuente: Guimaraes and Steven.

2.5. Definición de los robots paralelos

Una definición estandarizada para los robots paralelo es: "*Aquellos robots en los que el extremo final está unido a la base por más de una cadena cinemática independiente*" [21]. Un robot paralelo es considerado un mecanismo de cadena cinemática cerrada en el cuál una plataforma móvil se encuentra unida a una base fija. Esta configuración otorga a los robots paralelos ciertas ventajas sobre los robots seriales en términos de rigidez, velocidad, precisión e inercia en movimiento. En el caso de los robots paralelos las técnicas de análisis cinemático y diferencial de movimiento deben ser abordados con metodologías diferentes a los robots seriales. Su estudio conlleva a entender situaciones particulares como la pérdida de control debido a singularidades locales, limitaciones debido a su geometría y restricciones mecánicas. Una gran desventaja de los robots paralelos con respecto a los seriales, es su reducido espacio de trabajo. [22]

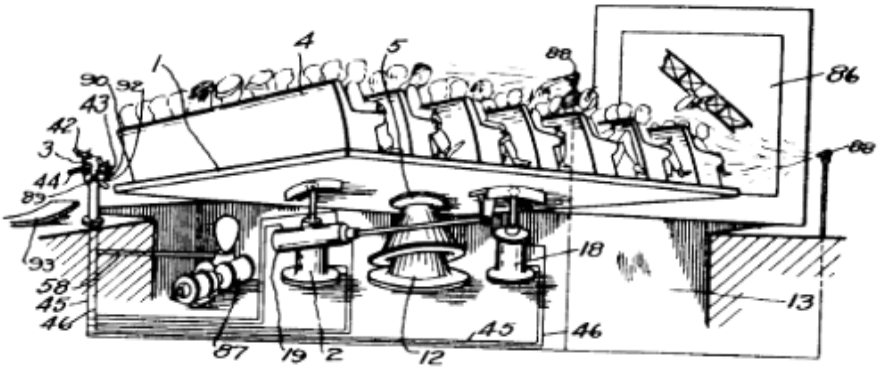


Figura 2.12: Robot Paralelo diseñado por R. Clavel.
Fuente: Kim.

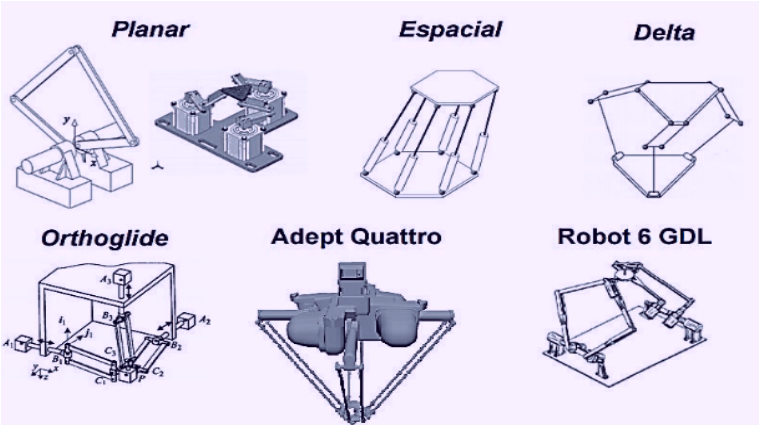


Figura 2.13: Tipos de robots paralelos.
Fuente: Fernández Yánez and Sotomayor Reinoso.

2.5.1. Tipos de robots paralelos

Dependiendo del movimiento que pueden realizar se clasifican en planares (2D) o espaciales (3D). De acuerdo a esto, la figura 2.13 muestra una clasificación general de los robots paralelos. Donde los robots más conocidos son los robots espaciales *delta y el hexápodo*. [23]

Figura 2.14: Robot para procedimiento ortopédico.
Fuente: Fernández Yánez and Sotomayor Reinoso.

2.5.2. Aplicaciones de los robots paralelos

Como se mencionó anteriormente, una de las principales aplicaciones de los robot paralelos es la simulación de vuelos o de movimientos [1]. Los robots paralelos *hexápodos* son utilizados para operaciones de traumatologías y ortopédicas (Figura 2.14) [12].

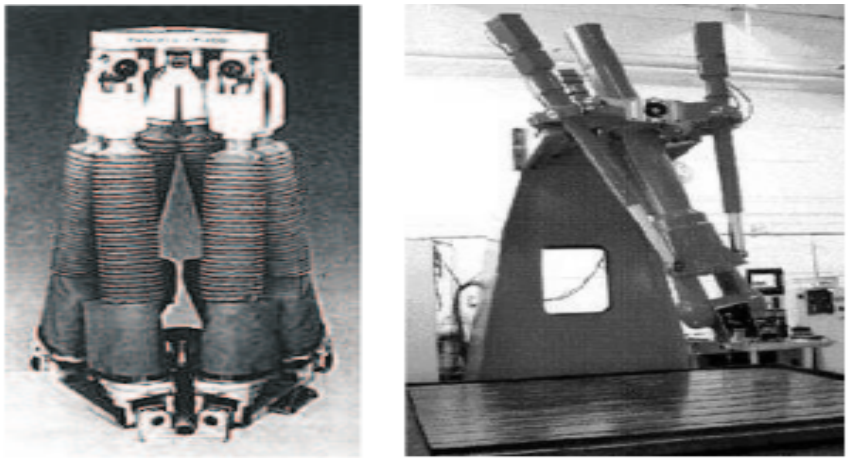


Figura 2.15: Robots Paralelos: FANUC y ABB 340.
Fuente: Barón and Pinilla.

Un robot paralelo muy utilizado para aquellas aplicaciones industriales donde se quiere levantar grandes volúmenes de carga es el robot *FANUC*. El robot *ABB 340* es muy utilizado para operaciones que requieran trabajar a altas velocidades (Figura 2.15) [15].

Para trabajos de difícil acceso o lugares remotos está *Remo I*, un robot paralelo de tipo submarino no tripulado, su estructura esta basada en la plataforma de *Stewart-Gough* (Figura 2.16) [24].

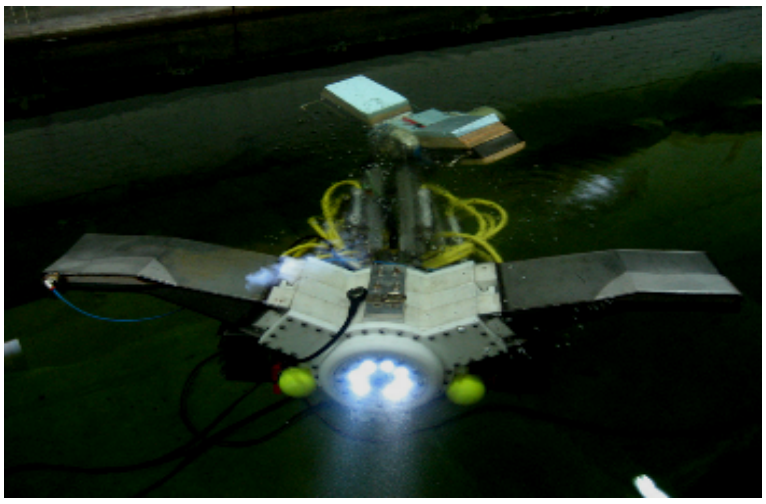


Figura 2.16: Robot paralelo acuático: Remo I.
Fuente: Álvarez et al.

2.6. Robot Delta

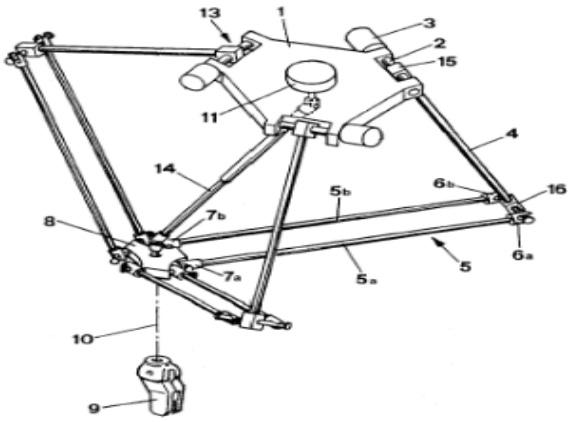


Figura 2.17: Robot Delta
Fuente: Martínez Macancela

Uno de los robots paralelos más comunes o utilizados en la industria es el robot paralelo Delta (RPD). Tiene 3 grados de libertad constituido por dos bases unidas por tres cadenas cinemáticas cerradas, la base superior se encuentra fija y para su construcción se pueden utilizar actuadores rotacionales o lineales según el trabajo o la aplicación (Figura 2.17). Uno de los más importantes pioneros de los robots paralelos tipo delta fue Reymond Clavel, dicho robot tenía forma simétrica, espacial y estaba compuestos por 3 eslabones paralelos equivalentes [25]. Los robots paralelos delta de 3GDL, tienen 3 articulaciones actuadas y las otras son articulaciones pasivas.

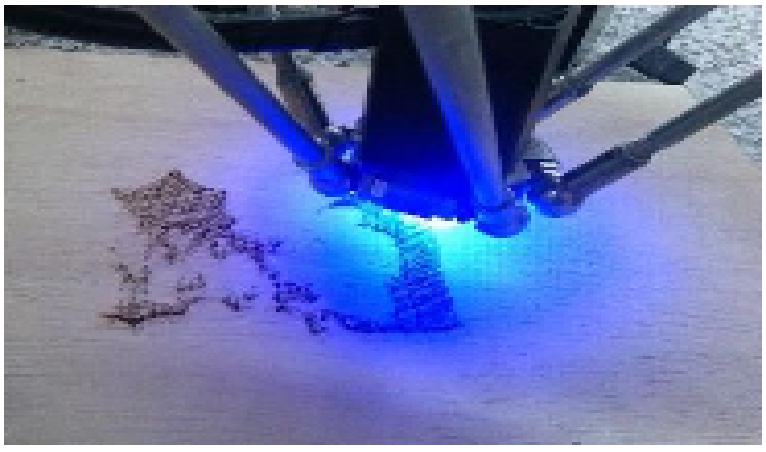


Figura 2.18: Cortadora láser CNC con estructura Delta.
Fuente: Mendoza and Zapata.

2.6.1. Aplicaciones de los Robots Delta

Una empresa sueca rediseña al robot delta para que pueda cargar un microscopio potente (Figura 2.19) [27]. Su principal aplicación es para cirugías y tratamientos quirúrgicos, años después vendieron la patente.

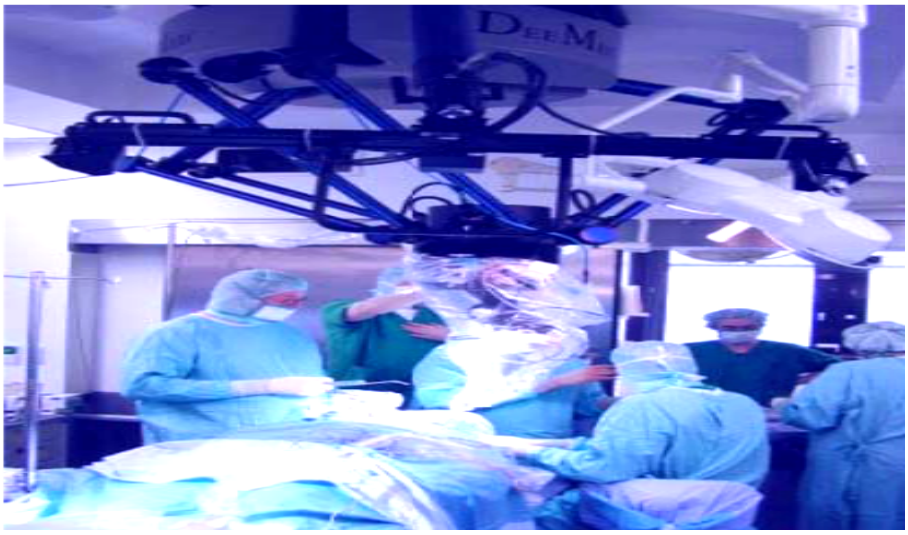


Figura 2.19: Robot Delta para aplicaciones quirúrgicas.
Fuente: Rueda Florez et al.

Los robots tipo delta son muy utilizados en el campo de la **impresión 3D** (Figura 2.20) [28].

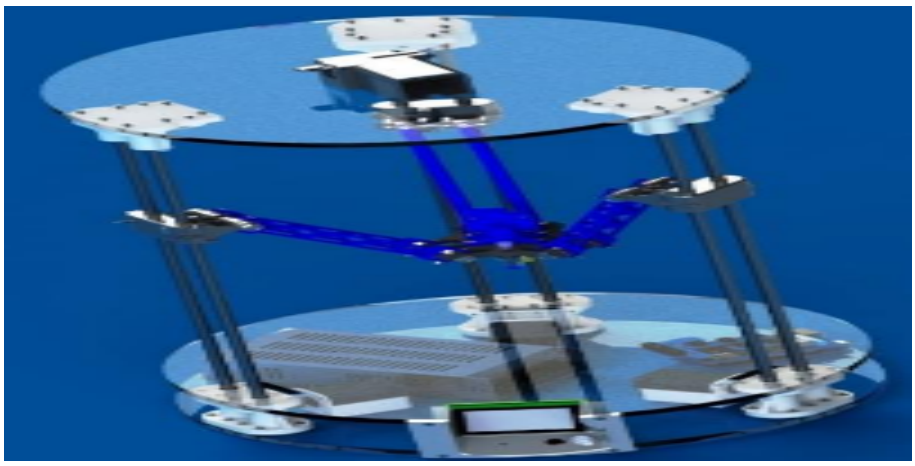


Figura 2.20: Impresora 3D con estructura Delta.
Fuente: Celi et al.

También es muy utilizado como una cortadora Láser **CNC**, (Figura 2.18) [26].

2.7. Robots de Teleoperación

Una definición de la NASA para los robots teleoperados es: *"dispositivos robóticos con brazos manipuladores y sensores con cierto grado de movilidad, controlados remotamente por un operador humano de manera directa o mediante un ordenador"* [29]. En ésta definición entrarían entre otros, los robots dedicados a cuidados médicos *(Robot quirúrgico Da Vinci)* (Figura 2.21) [30], educación, domésticos, uso en oficinas, ambientes peligrosos, aplicaciones espaciales, submarinas y en la agricultura.

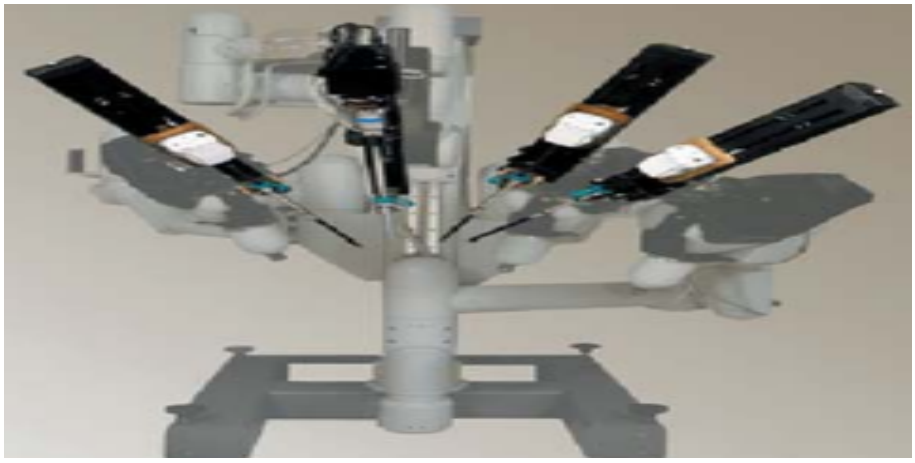


Figura 2.21: Esquema del Robot Da Vinci.
Fuente: Villavicencio Mavrich.

Sin embargo, esta definición de robots de servicio excluye a los robots telemanipuladores (*control remoto)*, ya que estos no se mueven mediante el control de un ordenador, sino que están controlados de manera directa por un operador humano. Un sistema teleoperado se compone principalmente de una estación de tele operación (dispositivo maestro), un sistema de comunicación (interfaz) que le va a permitir al usuario interactuar con la interfaz y un dispositivo esclavo, el esclavo puede ser un manipulador o un robot móvil (Figura 2.22) [31].

2.7.1. Elementos en la teleoperación

Los elementos básicos que conforman un sistemas de teleoperación son (Figura 2.25:

Dispositivo Maestro: Encargado de llevar el control y la manipulación del dispositivo teleoperado [32].

En el campo de la teleoperación hay 2 tipos de dispositivos maestros:

- **Dispositivo maestro de bajo nivel**: Cuando se útiliza un dispositivo que genera comandos para controlar directamente los servos del robot, por ejemplo:

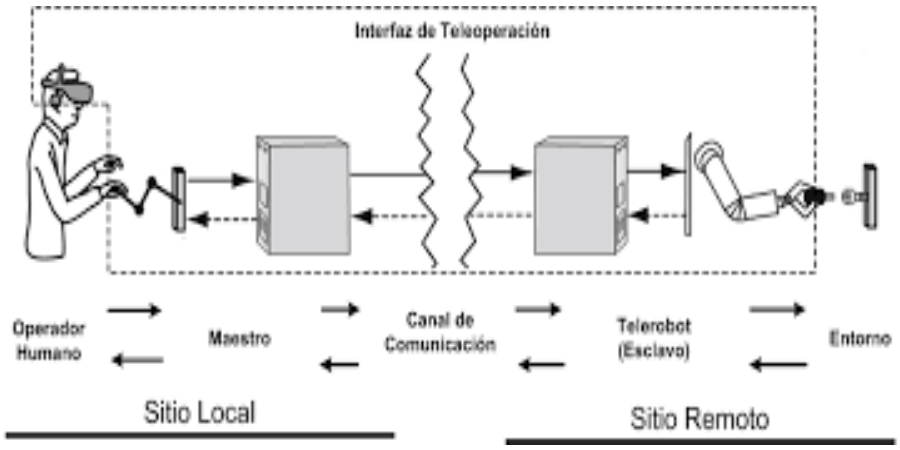


Figura 2.22: Sistema general de Teleoperación.
Fuente: Villavicencio Mavrich.

1. **Robot Novint (Falcon)**: Robot háptico activo de 3 grados de libertad (Figura 2.23), es un dispositivo que permite simular el tacto en un mundo virtual, permitiendo sentir objetos, superficies y fuerzas virtuales. Sus 2 principales aplicaciones son, los videojuegos y las aplicaciones tecnológicas. Fue creado por la empresa Novint Technologies, Inc.[33] [34].

Figura 2.23: Robot Haptico (Falcon).
Fuente: Karbasizadeh et al.

2. **Dispositivo Maestro similar**: Es cuando se utiliza un robot con una arquitectura igual a la del dispositivo esclavo. Donde el esclavo imita los movimientos del maestro.
3. **Interfaces o trajes biométricos corporales**: Se basa en la detección de marcas que se limitan a posiciones específicas del cuerpo humano, normalmente son trajes que se utilizan para la teleoperación de robots humanoides, codifican la información y la envían a los actuadores del dispositivo teleoperado (Figura 2.24) [35] [36].

Figura 2.24: Teleoperación con traje biométrico.
Fuente: Gálvez Cobo.

- **Dispostivos maestro de alto nivel**: Es cuando se le envían ordenes, ya el robot maestro está programado para seguir secuencias y patrones, por ejemplo:
    1. **Reconocimiento visual o de voz**: En este caso el operador contará con una cámara o con un micrófono respectivamente, para poder enviarle comandos (órdenes) visuales ó de voz al robot.

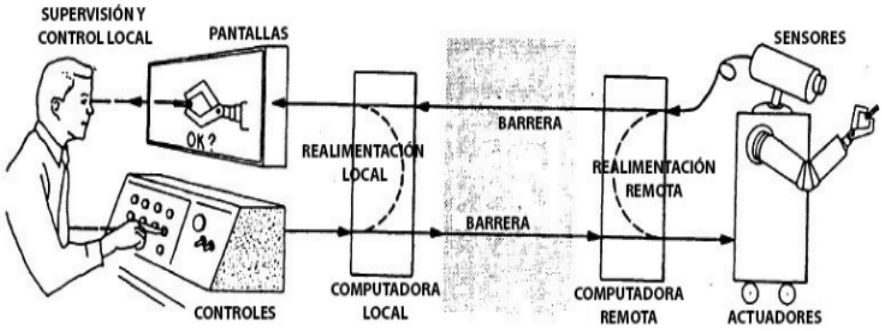


Figura 2.25: Esquema general de Teleoperación.
Fuente: Escolano and Minguez.

Dispositivo teleoperado (Esclavo): Es el elemento controlado por el dispositivo maestro, puede ser un vehiculo, un robot o sistemas inteligentes [38].

Interfaz: Es por donde el dispositivo maestro y esclavo se comunican y el maestro puede controlar al esclavo.

Sistema de control y comunicación: El primero consiste en módulos que permiten el procesamiento de los datos, el segundo hace referencia a los módulos que permiten el envío y la recepción de los datos [37].

2.7.2. Aplicaciones

Las principales aplicaciones de los robots teleoperados se presentan en entornos insalubres, como lo son:

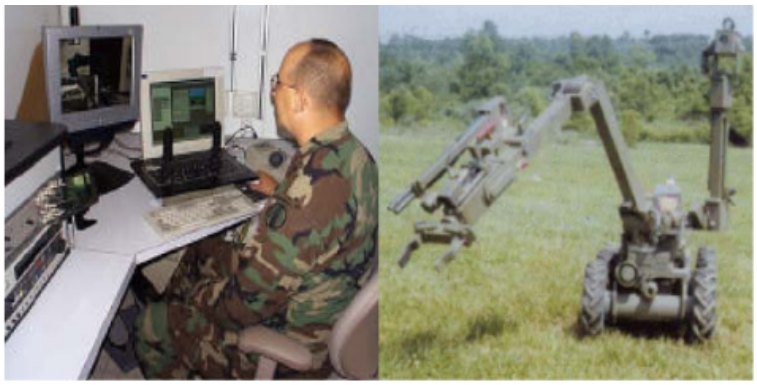


Figura 2.26: Robot de desactivación de minas.
Fuente: Correa.

- **Minas Subterránea**: La teleoperación es muy utilizada en ámbitos militares, cuyo objetivo es exponer a los robots para la desactivación de minas en lugar de poner en peligro la vida de las personas (Figura 2.26) [39].

Figura 2.27: Robot Teleoperado espacial y marino.
Fuente: Correa.

- **Fondo Marino**: Cuando se requiere trabajar en profundidades muy altas donde el cuerpo humano se expone a las presiones elevadas ejercidas por el agua y la poca luz que llega (Figura 2.27).
- **Plantas Nucleares**: De hecho fue en este campo donde surgió la teleoperación, la idea era evitar el contacto o la exposición directa con las sustancias radioactivas y limpiar lugares contaminados.
- **Espacio Exterior**: Surge de la necesidad de no enviar personas al espacio y exponerlas a la radiación solar (Figura 2.28) [40].

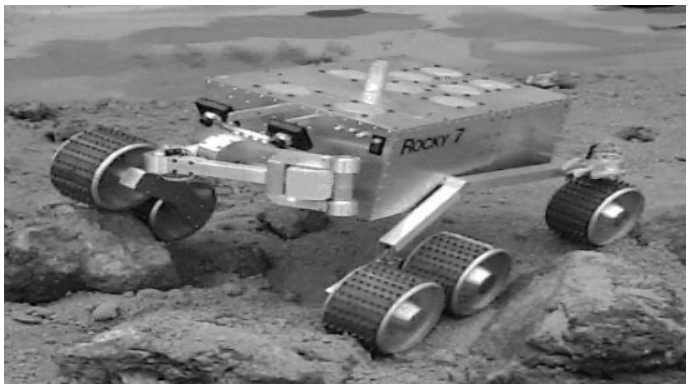


Figura 2.28: Robot Teleoperado Espacial: Rocky 7.
Fuente: Volpe et al.

- **Medicina**: Se utilizan para la realización de operaciones quirúrgicas de rutina, el robot quirúrgico más famoso es el robot *Da Vinci* (Figura 2.29) [41].

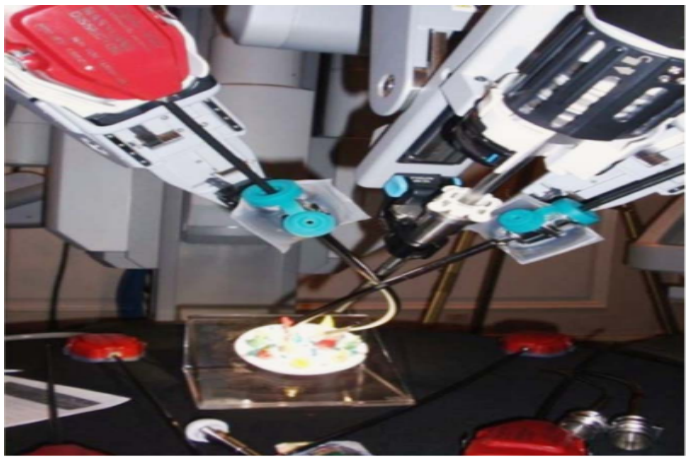


Figura 2.29: Robot Teleoperado Medico: Robot Da Vinci.
Fuente: Valero et al.

3. Análisis de Objetivos y Metodología

En este apartado se realizará una descripción del proceso que se ha llevado a cabo para el desarrollo de la pasantía de investigación, al igual que la metodología empleada.

3.1. Metodología

Metodología exploratoria: Como primera etapa, se profundizará en los diferentes temas de interés para adquirir destreza y llevar a cabo el desarrollo del proyecto (Figura 3.1).

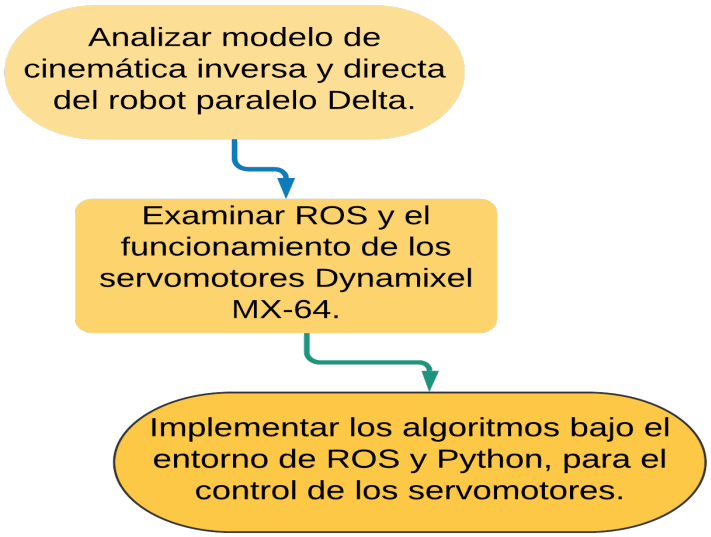


Figura 3.1: Metodología.
Fuente: Autor

Después se realizará la implementación de la teleoperación local, la cuál consistirá en mover al robot paralelo delta en su espacio de trabajo con las teclas direccionales del teclado. Esta parte contará un nodo publicador (Figura 3.2) que lanzará automáticamente la interfaz y enviará tres valores (Px, Py, Pz), correspondiente a la posición a la cual debe dirigirse el robot paralelo delta (dispositivo esclavo) quien estará en él nodo subscriptor (Figura 3.3) recibiendo, procesando y dirigiéndose a la posición recibida.

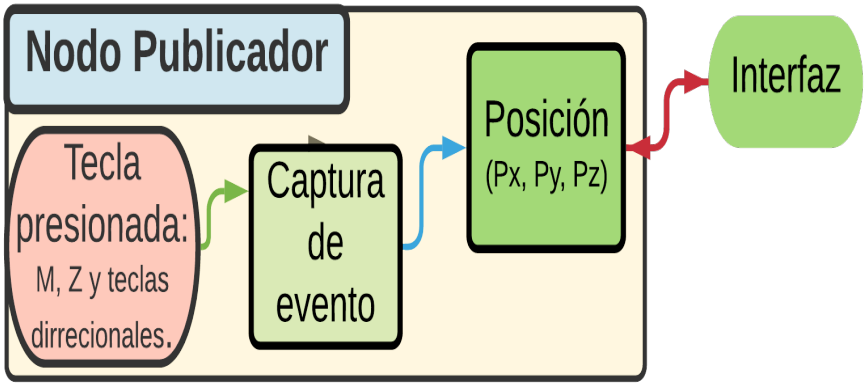


Figura 3.2: Nodo Publicador, Teleoperación local.
Fuente: Autor.

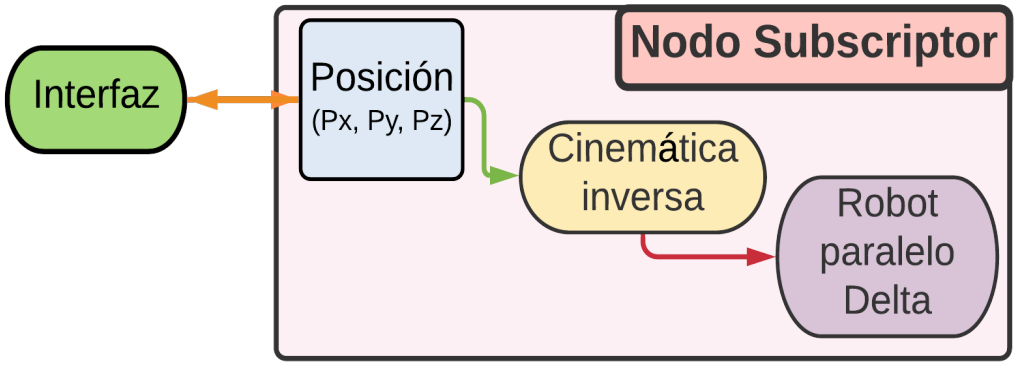


Figura 3.3: Nodo Subscriptor, Teleoperación local
Fuente: Autor.

Posteriormente, se desarrollará la teleoperación remota. Que consiste en manipular el robot paralelo delta por medio de un dispositivo maestro (Robot *falcon*), a travez de un nodo publicador se capturará la posición del efector final de dicho robot y se publicará. Se hará un nodo publicador - subscriptor intermedio para regular la frecuencia de muestreo del robot *falcon*, ya que es muy alta y satura la recepción de los datos. Después se implementará el nodo suscriptor, que al igual que en la teleoperación local será el robot paralelo delta recibiendo y dirigiéndose a la posición recibida. La interfaz mostrará los datos recibidos del falcon, los valores de los motores y la posición del robot paralelo delta (Figura 3.4).

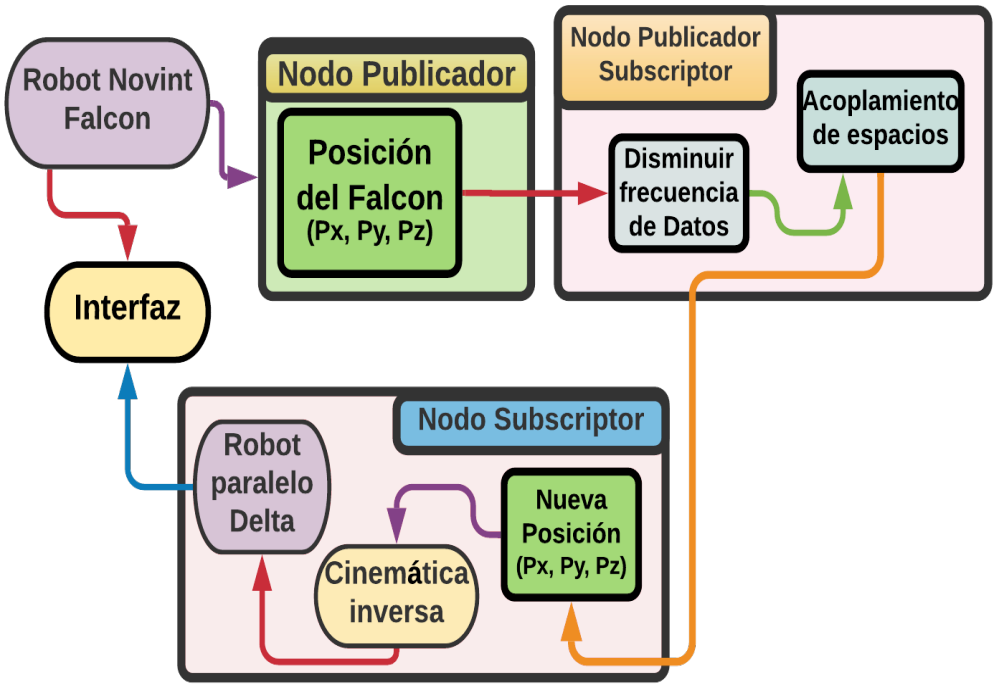


Figura 3.4: Nodos de la teleoperación remota.
Fuente: Autor.

3.1.1. Alcance del Proyecto

El objetivo de este proyecto de investigación es lograr la teleoperación tanto local como remota de un robot paralelo Delta tipo 3RRR ya diseñado y construido.

La teleoperación local contará con una interfaz que le permite a un usuario u operador enviarle datos provenientes del teclado (teclas direccionales) para mover al robot en su espacio de trabajo. La teleoperación remota también contará con una interfaz y tendrá como dispositivo maestro al robot háptico *Falcon*.

Para el dispositivo maestro se configurará el entorno de trabajo para que pueda ser reconocido y se pueda comunicar con el sistema de control, como solo es necesario la posición del efector final del falcon (dispositivo maestro), se utilizará un paquete que contiene dicha información.

3.2. Requisitos de Ingeniería

En esta sección se describe cada una de las herramientas que se utilizaron para el desarrollo del proyecto. La configuración básica que tiene el PC es la siguiente:

Sistema Operativo: Linux - Ubuntu 18.0
Distribución: ROS - Melodic Morenia. Para más información (Ver-Anexo A.1) [1].

Figura 3.5: Logotipo de la distribución de ROS-Melodic.
Fuente: ROS.org.

3.2.1. Servomotores Dynamixel MX-64

Estos servomotores cuentan con librerias compatibles con *ROS, Arduino, Linux, MacOS*. Hay softwares o paquetes que permiten configurarlos y controlarlos. Hay muchos tipos de servomotores Dynamixel (Figura 3.6), en el presente proyecto se desarrolló con servomotores Dynamixel MX-64. Para más información de estos servomotores ver (Anexo A.2) [2].

3.2.2. Descarga y Configuración de los servomotores por librerias

En una terminal ejecutar las siguientes líneas:

- Descargar librerias y dependencias:

Código 3.1: Descargar librerias

```
1 git clone https://github.com/ROBOTIS-GIT/dynamixel-workbench.git
2 git clone https://github.com/ROBOTIS-GIT/dynamixel-workbench-msgs.git
3 git clone https://github.com/ROBOTIS-GIT/DynamixelSDK.git
```

[1] http://wiki.ros.org/melodic/Installation
[2] http://emanual.robotis.com/docs/en/dxl/mx/mx-64/

Figura 3.6: Tipos de motores Dynamixel.
Fuente: Robotys Dynamixel.

- Configurar librerias y dependencias:

Código 3.2: Configurar librerias

```
cd ~/dynamixel-workbench/dynamixel_workbench_toolbox/examples
mkdir -p build && cd build
cmake ..
make
```

Descarga y Configuración de los servomotores por Software

Los servomotores dynamixel se pueden configurar por librerías, pero es mucho más práctico hacerlo por softwares: Dynamixel Wizard, es por defecto el software utilizado para la configuración de los parámetros de los servomotores dynamixel (Figura 3.7).

- **Dinamixel Wizard**:

Para descargar el archivo: `https://www.robotis.com/service/download.php?no=1671`

Luego ejecutar las siguentes lineas en una terminal:

Código 3.3: Descargar librerias de Dinamixel Wizard

```
sudo chmod 775 DynamixelWizard2Setup_x64
./DynamixelWizard2Setup_x64
```

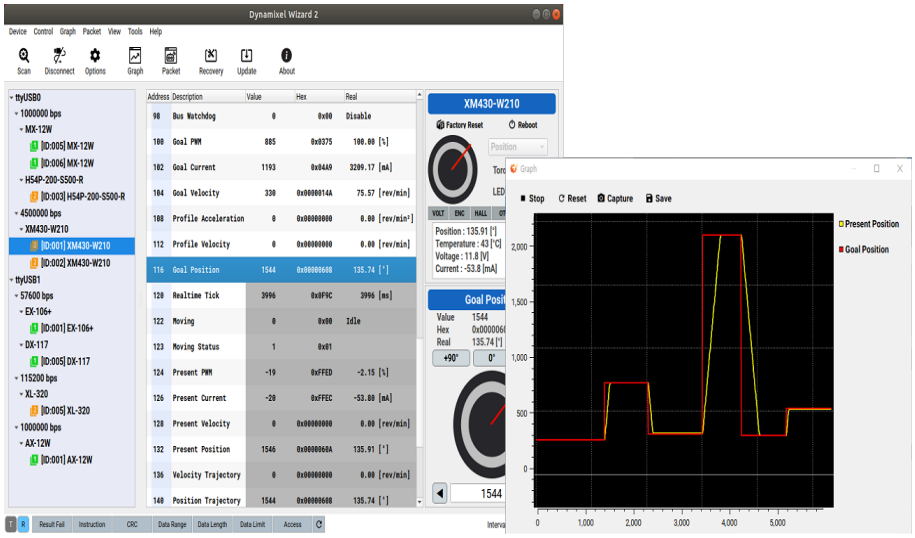


Figura 3.7: Dynamixel Wizard.
Fuente: Robotis e-Manual.

3.2.3. U2D2 y Tarjeta de comunicación HUB

Estas tarjetas permiten la comunicación de los servomotores con el dispositivo de control, en este caso el PC.

3.2.4. U2D2

Convertidor de comunicación USB de pequeño tamaño, que permite controlar y manipular los diferentes servomotores Dynamixels, la (Figura 3.8) muestra un esquema general de como conectar el PC y los servomotores dynamixel por medio del convertidor U2D2[3] y la (Figura 3.9) muestra los diferentes tipos de conectores (pines), normalmente los servomotores dynamixel trabajan con pines TTL.

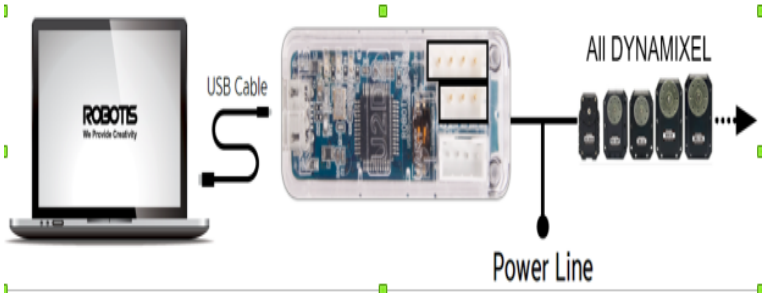


Figura 3.8: Convertidor U2D2.
Fuente: Robotis e-Manual.

[3] http://emanual.robotis.com/docs/en/parts/interface/u2d2/

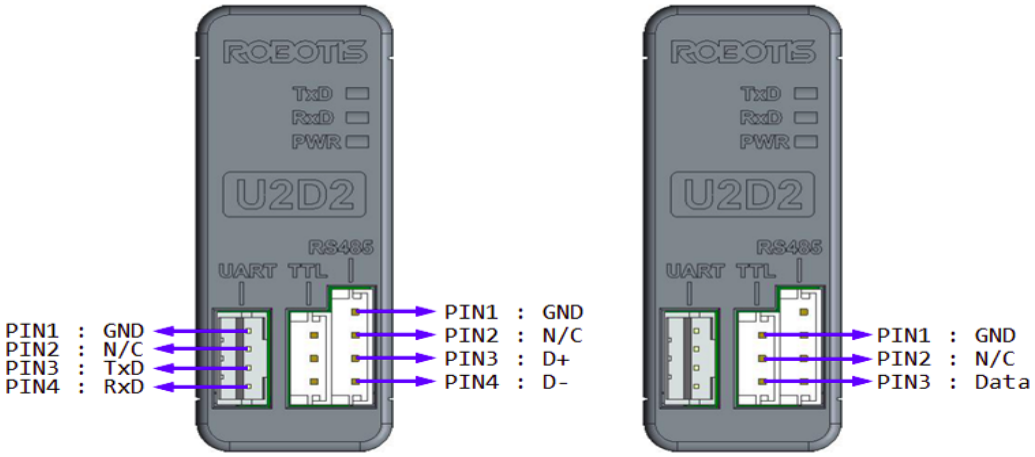


Figura 3.9: Pines o Conectores.
Fuente: Robotis e-Manual.

3.2.5. Tarjeta HUB

Figura 3.10: Convertidor U2D2.
Fuente: Robotis e-Manual.

La tarjeta HUB[4] (Figura 3.10), permite la integración de varios dispositivos entre sí, lo cual la hace muy versátil. Los servomotores dynamixels traen un bus de comunicación que les pertmite conectarse en serie, facilitando en gran medida el cableado, esta tarjeta se puede conectar a la fuente de alimentación por medio de adaptador AC/DC (Figura 3.11) o con cables pelados por el bloque de terminales de tornillos.

[4] `https://www.trossenrobotics.com/6-port-ax-mx-power-hub`

Figura 3.11: Adaptador, fuente de alimentación.
Fuente: Autor.

Esta tarjeta es compatible con los siguientes Dynamixels:

- RX-24F
  RX-28
  RX-64
- MX-28R
  MX-106R
  MX-64
- EX-106

3.2.6. Robot Novint Falcon

Es el dispositivo maestro de la teleoperación remota como se mencionó anteriormente. En la (Figura 3.12) se puede observar el comportamiento de los movimientos por ejes. En la siguiente dirección encontrará información detallada de como hacer la instalación de un paquete que contiene un conjunto de herramientas que facilitan el uso del robot *Novint Falcon*: `https://github.com/jcorredorc?tab=repositories`.

3.2.7. Configuración del Paquete Novint Falcon

El codigo 3.4 muestra como configurar las librerias[5] y *Drivers*[6] que permiten que el robot *Novint falcon* pueda comunicarse con el entorno y espacio de trabajo de ROS y

[5] `https://github.com/jcorredorc?tab=repositories`
[6] `https://github.com/libnifalcon/libnifalcon`

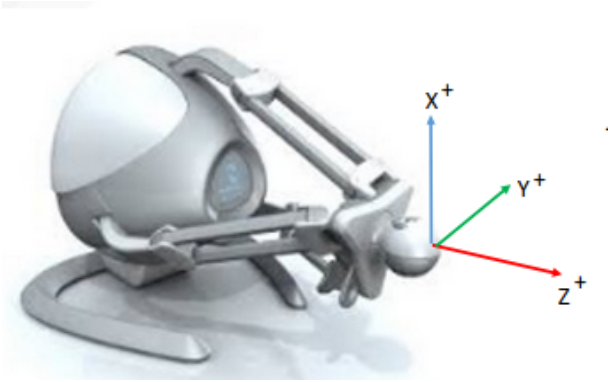


Figura 3.12: Ejes del Novint Falcon.
Fuente: Autor.

el PC. La (Figura 3.13) muestra la topología general del paquete, donde *src* contiene todos los nodos, en *msg* se pueden encontrar los tipos de mensajes que se utilizan y *Node* contiene al nodo principal *falcon_node*, el cual se debe de ejecutar siempre de primero para que el entorno de **ROS** pueda reconocer al robot *falcon*: (*rosrun ros_falcon falcon_node*).

En una terminal ejecutar los siguientes comandos:

Código 3.4: Configurar librerias del Falcon

```
cd catkin_ws/src
git clone https://github.com/jcorredorc/ros_falcon.git
git clone https://github.com/libnifalcon/libnifalcon.git
cd ros_falcon/
git clone https://github.com/libusb/libusb.git òsudo apt−get install libusb−1.0−0
sudo cp udev_rules/99−udev−novint.rules /etc/udev/rules.d
```

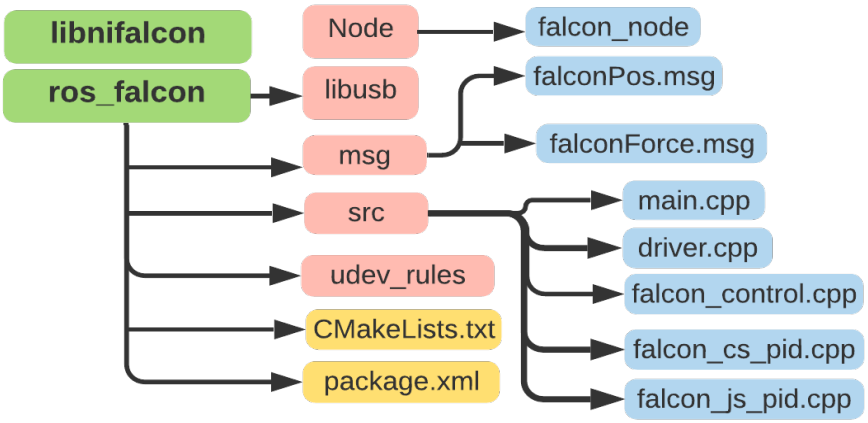


Figura 3.13: Topología del paquete ros_falcon.
Fuente: Autor.

4. Diseño y resolución del trabajo realizado

4.1. Desarrollo y diseño de la Arquitectura

En este apartado se describen los pasos llevados a cado durante el desarrollo del proyecto. El proyecto consta de 2 partes principales, la primera es la teleoperación local y la segunda la teleoperación remota.

Figura 4.1: Esquema de la Teleoperación Local.
Fuente: Autor.

La Figura 4.1 muestra el esquema general de la teleoperación local. El PC y el robot paralelo delta se comunican por medio del convertidor U2D2 y la tarjeta hub, las ID de los servomotores son [1, 2, 3] respectivamente y la posición del robot delta se estará mostrando en la interfaz.

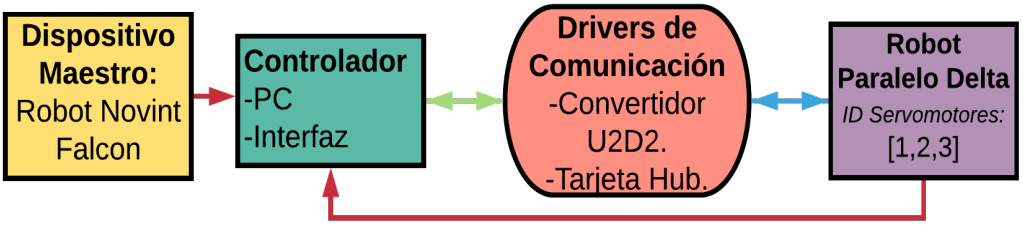


Figura 4.2: Esquema de Teleoperación Remota.
Fuente: Autor.

La Figura 4.2 muestra el esquema general de la teleoperación remota. El dispositivo maestro es el robot falcon, el cual estará publicando y enviando su posición a los controladores (interfaz y PC) y se comunicará con el robot paralelo delta por medio de los drivers de comunicación. La posición de ambos robots se estará mostrando en la interfaz.

4.1.1. Configuración de los Servomotores

Los servomotores hay que adaptarlos y configurarlos de tal manera, que los datos que reciban puedan ser interpretados (Tabla 4.1). También hay que establecer una posición de referencia en cada uno de los servomotores de tal manera que ellos lo puedan interpretar como su posición cero, a dicho proceso se le conoce como *caracterización de los motores*.

Para caracterizar los servomotores hay que tener en cuenta como quedan montados dentro la estructura mecánica del robot paralelo delta, se toman dos posiciones del servomotor como referencia, se analiza el valor real que tiene el servomotor en las dos referencias tomadas, se calcula el factor de resolución (RM) y se mira hacia donde gira el servomotor (Giro a favor o en contra de las manecillas del reloj), esto último es para ver si se le suma o se le resta el valor enviado al valor de referencia. Donde $RM = |(ValorReal1 - ValorReal2)/(ValorRef.1 - ValorRef.2)|$. La Figura 4.3 muestra la configuración del ID y baudios del servomotor 1, la configuración de los otros servomotores es la misma, solo hay que cambiar ID=1, por ID=2 para el servomotor 2 Y ID=3 para el servomotor 3.

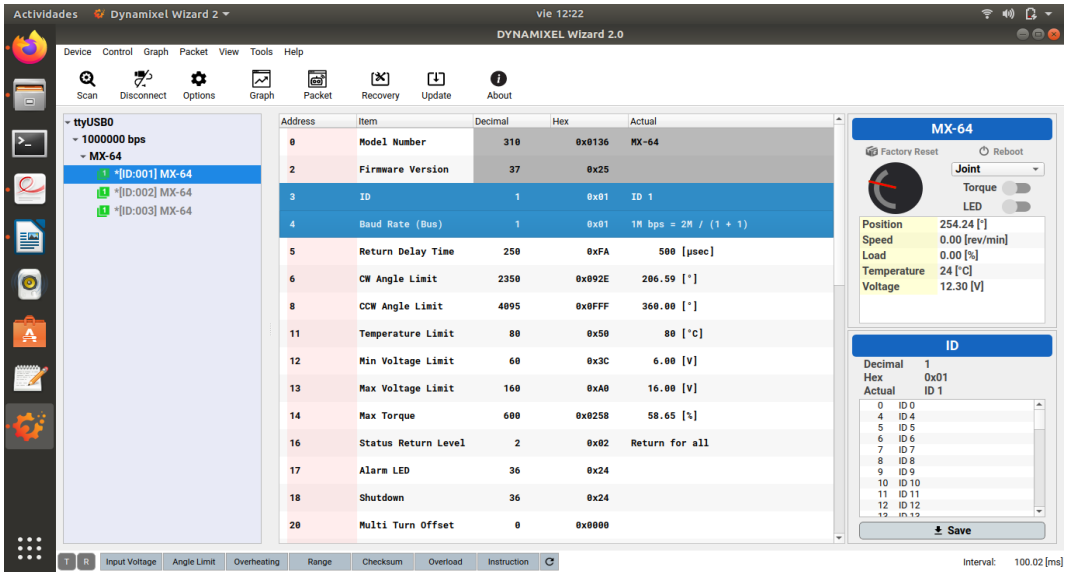


Figura 4.3: Configuración del servomotor1.
Fuente: Autor.

Factor de resolución de los servomotores

Para el servomotor1 los valores que se tomaron como referencias fueron: 3800 y 2820, el primero valor indica la posición que se quiere tomar como 0° y el segundo valor indica la posición que se quiere tomar como 90° para el servomotor1. Se hace una división entre la diferencia de los valores de referencia tomados y la diferencia de los valores asignados a cada referencia, el mismo procedimiento se hace para el servomotor2 y servomotor3. La Tabla 4.2 muestra un resumen de los valores de referencias de los servomotores y el factor de resolución de cada uno de ellos.

Servomotor1: $FR = |(3800 - 2820)/(0 - 90)| = 10,8888$

Servomotor2: $FR = |(3050 - 1980)/(0 - 90)| = 11,8888$

Servomotor3: $FR = |(2350 - 1320)/(0 - 90)| = 11,4444$

Configuración de los Servomotores		
ID Servo-motor	**Baud Rate**	**Rango de Operación Max-Min**
1	1000000	2350 - 4095
2	1000000	1800 - 3450
3	1000000	1180 - 2600

Tabla 4.1: Configuración de los Motores

Parámetros de los Servomotores					
ID Motor	**Valor Ref. 1**	**Valor Real1**	**Valor Ref. 2**	**Valor Real2**	**Resolución (RM)**
1	0	3800	90	2820	10,888
2	0	3050	90	1980	11,888
3	0	2350	90	1320	11,444

Tabla 4.2: Caracterización de los Servomotores

Los servomotores reciben valores de 0 a 4095, si el valor recibido no se encuentra entre este rango o dentro del rango de operación asignado, los servomotores no realizan ninguna acción y devuelven **False**, indicando que el dato recibido es erróneo. Para resolver el problema hay que hallar una expresión que permita convertir los valores recibidos a los valores que pueden recibir los servomotores, como la conversión da un número flotante hay que convertirlo al entero más próximo.

Motor1: $3800 - FR(PosicinDeReferencia)$

Ejemplo: Si se quiere que el motor 1 gire 45° a partir de su posición de referencia: $3800 - 10,88(45) = 3310$, se le envía al motor **3310**, el cual es interpretado como 45°.

Motor2: $3050 - FR(posicionDeReferencia)$

Ejemplo: Si se quiere que el motor 2 gire 60° a partir de su posición de referencia: $3050 - 11,88(60) = 2337$, se le envía al motor **2337**, el cual es interpretado como 60°.

Motor3: $2350 - FR(PosicionDeReferencia)$

Ejemplo: Si se quiere que el motor 3 gire 45° a partir de su posición de referencia: $2350 - 11,44(45) = 1385$, se le envía al motor **1835**, el cuál es interpretado como 45°.

4.1.2. Cinemática del Robot Paralelo Delta

A continuación se analiza la cinemática directa e inversa del robot paralelo Delta 3RRR, este robot presenta una configuración simétrica, solo será necesario resolver el problema de la cinemática para una sola cadena cinemática:

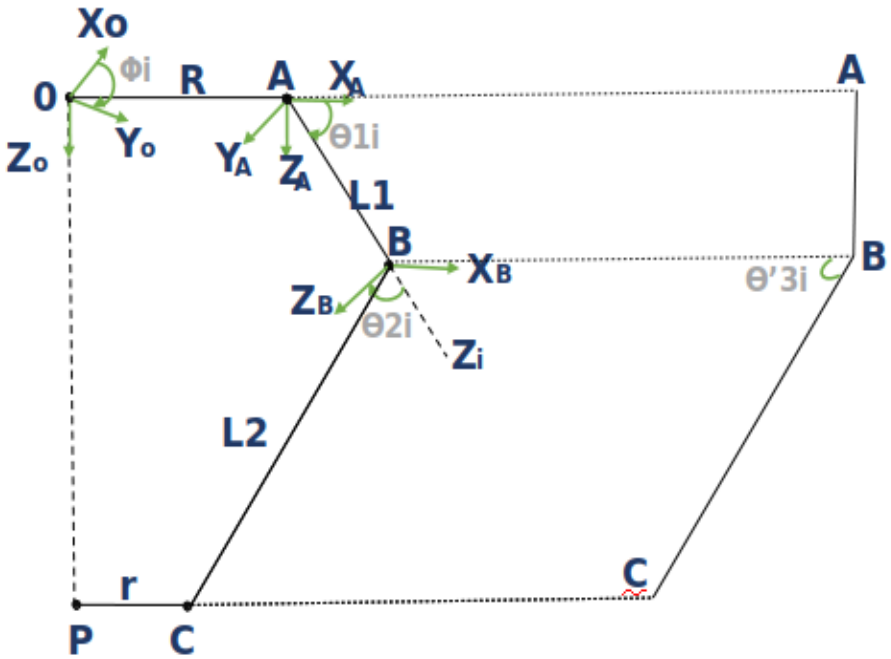


Figura 4.4: Vista Frontal y Lateral, Robot RPD
Fuente: Autor.

La Figura 4.4 muestra la vista frontal y lateral de una de las cadenas cinemáticas. **R** es la distancia que hay desde el sistema de origen **0** hasta el punto **A**, el cuál es una articulación accionada (servomotor1). **L1** es la distancia que hay desde la articulación accionada hasta la articulación pasiva (rotacional, punto **B**). **L2** es la distancia que hay desde la articulación pasiva hasta el punto **C**, la cual es una articulación rotacional y **r** es la distancia que hay desde el punto **C** hasta el punto **P** (*posición del efector final*). θ_{1i} es un ángulo accionado (valor del servomotor1), θ_{2i} y θ_{3i} son ángulos pasivos.

4.1.3. Cinemática Inversa

El problema de la cinemática inversa consiste en hallar el valor de las articulaciones $[\theta_{11}, \theta_{12}, \theta_{13}]$, dada la posición espacial $[P_x, P_y, P_z]$ del efector final del robot delta:

$$[P_x, P_y, P_z] \rightarrow \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \\ \theta_{31} & \theta_{32} & \theta_{33} \end{bmatrix}.$$

4 ángulos de interés:

- ϕ Distancia de los brazos respecto al sistema **0**, $(0°, 120°, 240°)$
- $\theta_{1i} \rightarrow$ única variable actuada (Servomotor Dynamixel MX-64).
- $(\theta_{2i}, \theta_{3i}) \rightarrow$ variables sub-actuadas de la junta universal.

De la Figura 4.4 se tiene: $\overline{OP} = \overline{OA} + \overline{AB} + \overline{BC} + \overline{CP}$ y

$$L_2 \cos(\theta_{31}) = P_y \cos(\phi_i) - P_x \sin(\phi_i) \tag{4.1}$$

Para simplificar los cálculos tomamos como referencia inicial al punto A y como referencia final al punto B:

$$\overline{AB} + \overline{BC} = \overline{OP} - \overline{OA} - \overline{CP} \tag{4.2}$$

Tomando $\phi_i = 0$, en términos de matrices queda:

$$\begin{bmatrix} L_1 \cos(\theta_{11}) \\ 0 \\ L_1 \sin(\theta_{11}) \end{bmatrix} + \begin{bmatrix} L_2 \sin(\theta_{31}) \cos(\theta_{11} + \theta_{21}) \\ L_2 \cos(\theta_{31}) \\ L_2 \sin(\theta_{31}) \sin(\theta_{11} + \theta_{21}) \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} - \begin{bmatrix} R \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} \tag{4.3}$$

$$\overline{AB} + \overline{BC} = \begin{bmatrix} L_1 \cos(\theta_{11}) + L2 \cos(\theta_{11} + \theta_{21}) \sin(\theta_{31}) \\ L_2 \cos(\theta_{31}) \\ L_1 \sin(\theta_{11}) + L_2 \sin(\theta_{31}) \sin(\theta_{11} + \theta_{21}) \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} + \begin{bmatrix} r - R \\ 0 \\ 0 \end{bmatrix} \tag{4.4}$$

De la ecuación 4.3 se puede obtener θ_{31}:

$$P_y = L_2 cos(\theta_{31}) \rightarrow cos(\theta_{31}) = P_y/L_2 \rightarrow \theta_{31} = \arccos(P_y/L_2) \tag{4.5}$$

Si se eleva cada término de la ecuación 4.4 al cuadrado y se suman, se obtienen los valores de θ_{21} y θ_{11}.

$$[L_1 \cos(\theta_{11}) + L2 \cos(\theta_{11} + \theta_{21}) \sin(\theta_{31}) L_2 \cos(\theta_{31})]^2 + ... \tag{4.6}$$
$$...(L_2 \cos(\theta_{31})^2 + [L_2 \sin(\theta_{31}) \sin(\theta_{11} + \theta_{21})]^2 = (P_x + r - R)^2 + P_y^2 + P_z^2$$

Después de una serie de artilujios matemáticos se obtiene:

$$C_x^2 + C_y^2 + C_z^2 = L_1^2 + L_2^2 + 2L_1 L_2 \sin(\theta_{31}) \cos(\theta_{21}) \tag{4.7}$$

Donde:
$C_x = P_x \cos(\phi_1) + P_y \sin(\phi_1) + r - R$
$C_y = P_y \cos(\phi_1) - P_x \sin(\phi_1)$
$C_z = P_z$
$A = -(C_x + C_y + C_z + L_2 \cos(\theta_{31}) + L_2 \sin(\theta_{31}) \sin(\theta_{21}) + L_2 \sin(\theta_{31}) \cos(\theta_{21}))$
$B = 2(L_1 - L_2 \sin(\theta_{31}) \sin(\theta_{21}) + L_2 \sin(\theta_{31}) \cos(\theta_{21}))$
$C = L_1 + L_2 \sin(\theta_{31}) \sin(\theta_{21}) + L_2 \sin(\theta_{31}) \cos(\theta_{21})) + L_2 \cos(\theta_{31})$

De la ecuación 4.7 se obtiene: θ_{21} y θ_{11}:

$$\theta_{21} = \arccos((C_x^2 + C_y^2 + C_z^2 - L_1^2 - L_2^2)/(2L_1 L_2 \sin(\theta_{31})) \tag{4.8}$$

$$\theta_{11} = 2 \arctan((-B \pm \sqrt{B^2 - 4AC})/2A) \tag{4.9}$$

Los otros valores articulares se pueden hallar aplicando el mismo procedimiento, solo hay que cambiar $\phi_i = 0$, por $\phi_i = 120°$ y luego por $\phi_i = 240°$

4.1.4. Cinemática Directa

El problema de la cinemática directa consiste en hallar la posición del efector final $[P_x, P_y, P_z]$, a partir del valor de sus articulaciones $[\theta_{11}, \theta_{21}, \theta_{31}]$.

De la ecuación 4.4 se tiene que:

$$P_x = R - r + L_2 \sin(\theta_{31}) \cos(\theta_{11} + \theta_{21}) + L_1 \cos(\theta_{11}) \tag{4.10}$$

$$P_y = L_2 \cos(\theta_{31}) \tag{4.11}$$

$$P_z = L_2 \sin(\theta_{31}) \sin(\theta_{11} + \theta_{21}) \tag{4.12}$$

Para la implementación del código de la cinemática Inversa y Directa, hay que tener en cuenta con que tipo de configuración se va a trabajar, si codo arriba o codo abajo, como se va a trabajó con codo arriba se tomó el signo positivo de la ecuación 4.9.

La Tabla 4.3 y la Tabla 4.3 resumen los parámetros y dimenciones del Robot Paralelo Delta (RPD):

Parámetros del RPD		
Eje	**Mínimo(mm)**	**Máximo (mm)**
X	-100	100
Y	-150	180
Z	-338	-188

Tabla 4.3: Rango de Movilidad por eje del RPD.

Dimensiones del RPD		
	Longitud(mm)	**Longitud((cm)**
L1	140	14
L2	280	28
R	60	6
r	35	3,5

Tabla 4.4: Dimensiones físicas del RPD.

4.2. Implementación

En este apartado se desarrollará la codificación necesaria para el funcionamiento y cumplimiento de cada uno de los objetivos planteados.

4.2.1. Teleoperación local

La teleoperación local tiene la siguiente estructura:

- **Dispositivo Maestro:** Teclado del PC, teclas direccionales, las teclas (Z y M).
- **Dispositivo Esclavo:** Robot Paralelo Delta (RPD).

- **Interfaz:** Implementada con *Python*, permite ver las posiciones de los servomotores y la posición del robot en el espacio de la tarea.
- **Comunicación:** ROS (nodos), Tarjeta hub para reducir el cableado y la tarjeta U2D2 para el intercambio de datos entre el PC-interfaz-RPD y ROS.

Crear Paquete (ROS y Python)

Teniendo la cinemática inversa del robot y conociendo los aspectos básicos del funcionamiento de los servomotores, se realiza el primer objetivo del proyecto: teleoperar el robot paralelo delta con las teclas direccionales, M y Z del PC.

Los pasos a seguir son:

- Crear un paquete, con sus respectivas dependencias.
- Crear un nodo publicador, contará con una interfaz y va a capturar los eventos del teclado, convertirlos en los datos que reciben los motores y publicará la posición del efector final del robot delta.
- Crear un nodo subscriptor (robot delta), va a obtener la posición del nodo publicador y se moverá a la posición recibida.

Crear Paquete bajo el entorno de ROS
Ejecutar los siguientes comandos en una terminal:

Código 4.1: Nombre del paquete: proyecto_delta

```
1 cd catkin_ws/src
2 catkin_create_pkg proyecto_delta roscpp rospy std_msgs dynamixel_workbench_controllers
```

Automáticamente se crean los siguientes archivos: CMakeList.txt y Package.xml. Los cuáles se tienen que modificar, dejandolos como se muestran en (Anexos: A.4 y A.5).

Nodo publicador (Interfaz): Este nodo se comportará como el dispositivo maestro y es el encargado de publicar la posición a la que el dispositivo esclavo (robot delta) se tiene que mover (Figura 4.5). Se crea una carpeta con el nombre **src** dentro del paquete creado, en donde se guardan todos los nodos creados.

Si se presiona la tecla **Z** el robot sube (eje $Z \uparrow$), si es la tecla **M** el robot baja (eje $Z \downarrow$). La tecla ($\leftarrow$) hace que se mueva hacia la Izquierda (eje $X \downarrow$) y la tecla ($\rightarrow$), hacia la Derecha (eje $X \uparrow$). La tecla ($\downarrow$) hace que se mueva hacia atrás (eje Y$\downarrow$) y la

tecla (↑), hacia delante (eje Y ↑). La (Tabla 4.5) resume el comportamiento del robot dependiendo de la tecla que se presione y el código (ver - Anexos: A.6) la implentación del nodo publicador.

Eje	Disminuye	Aumenta
X	←	→
Y	↓	↑
Z	M	Z

Tabla 4.5: Movilidad del robot delta vs Tecla presionada.

Característica del nodo publicador:

- Nombre del archivo: *publicando_coordenadas.py*.
- Nombre del Nodo: *pub_posicion*.
- Nombre del topic: *coordenadas*.
- Recibe: Teclas presionadas.
- Publica: una terna (Px, Py, Pz).

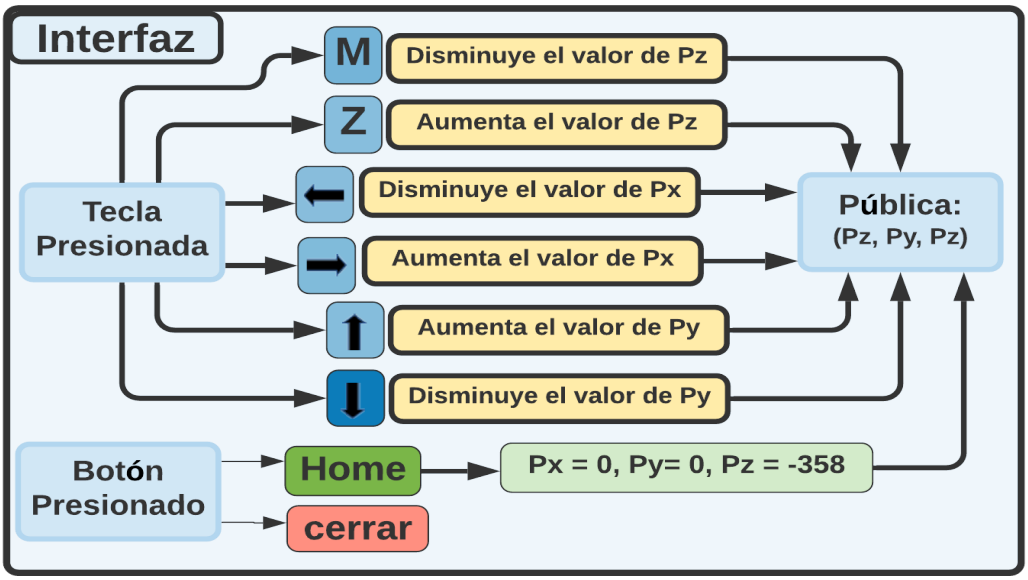


Figura 4.5: Esquema de la Interfaz de la Teleoperación Local.
Fuente: Autor.

Interfaz: Es uno de los elementos más importantes en un sistema de teleoperación, permite ver el estado del proceso ó si hay fallas. La interfaz cuenta con dos botones, uno que permite enviar al robot a una posición establecida (HOME) y otro que permite cerrar la interfaz. Al correr (ejecutar) el nodo publicador, este lanza automáticamente la interfaz mostrada en la (Figura 4.6). Para ejecutar el nodo publicador no es necesario tener conectado los servomotores, pero al lanzar el nodo subscriptor si deben de estar conectados.

Los botones **HOME** y **cerrar** así se presionen, no realizan ninguna acción de inmediato. Al presionarlos muestran respectivamente un mensaje de advertencia, preguntandole al usuario si quieren enviar el robot a la posición **HOME** establecida (0, 0, -358) ó si quieren cerrar la interfaz - nodo publicador (Figura 4.6).

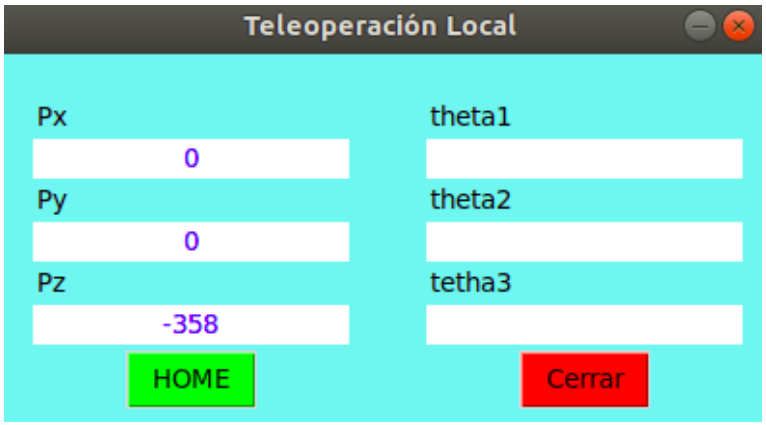


Figura 4.6: Interfaz de la Teleoperación Local.
Fuente: Autor.

Nodo Subscriptor (Robot Paralelo Delta): En este nodo estará el robot paralelo delta (dispositivo esclavo), quien se subscribirá al nodo publicador para recibir la terna que este pública y se moverá hasta llevar a la posición recibida. Este nodo (ejecutable) se guarda en la carpeta **src** del paquete creado. La implementación del nodo subscriptor lo puede ver en (Anexos: A.7). La (Figura 4.7) muestra el esquema general del nodo subscriptor.

Característica del nodo Subscriptor:

- Nombre del archivo: *pos_coordenadas.py*.
- Nombre del Nodo: *tele_local*.
- Nombre del topic: *coordenadas*.
- Recibe: una terna (Px, Py, Pz).

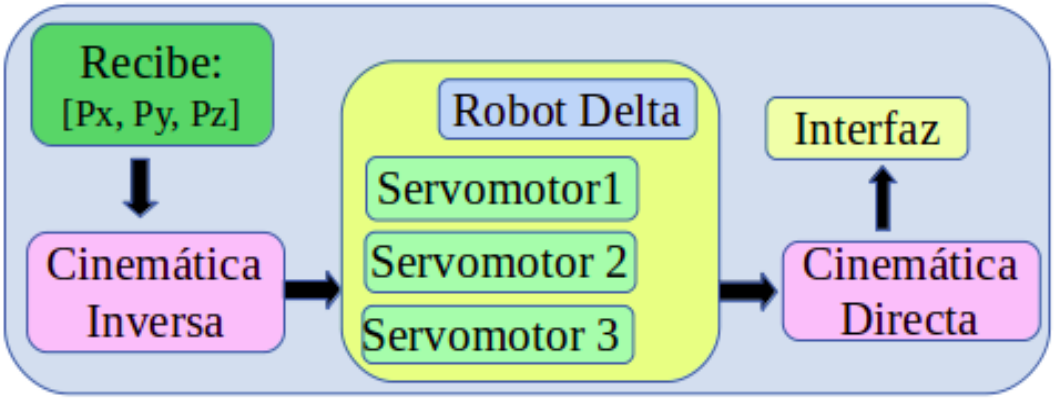


Figura 4.7: Esquema del Nodo Subscriptor.
Fuente: Autor.

La (Figura 4.8), muestra la arquitectura del paquete y los archivos creados. Los documentos que están en color azul son los nodos (ejecutables) de la teleoperación local, el archivo *ik.py* contiene la cinemática inversa y directa del robot delta.

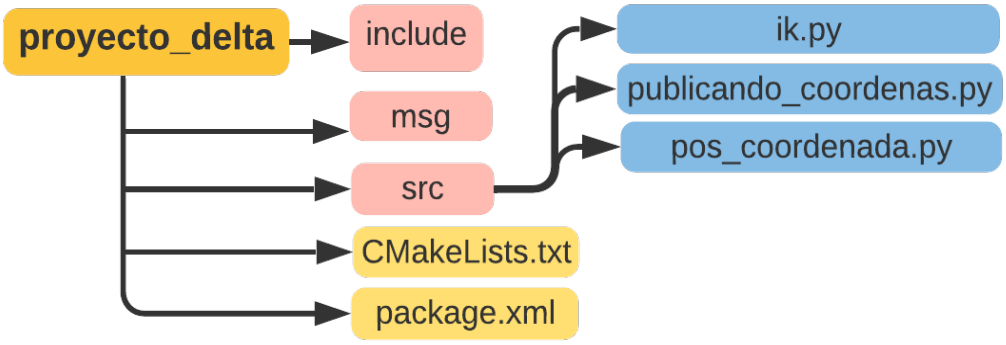


Figura 4.8: Topología del paquete creado.
Fuente: Autor.

4.2.2. Teleoperación Remota

La teleoperación remota tiene la siguiente estructura:

- **Dispositivo Maestro:** Robot Novint Falcon.
- **Dispositivo Esclavo:** Robot ParaLelo Delta (RPD).
- **Interfaz:** Implementada en *Python*.

- **Comunicación:** ROS (nodos), Tarjeta hub para reducir el cableado y la tarjeta U2D2 para el intercambio de datos entre el Falcon-(PC-Interfaz-ROS)-Robot delta.

Los pasos a seguir son:

- Utilizar el paquete creado.
- Crear nodos, para obtener la posición del falcon, adaptar los datos y enviárselos a los servomotores.
- Crear un nodo subscriptor, Va a ir a la posición que reciba del robot falcon.

Nodo Publicador-Subscriptor (Interfaz): Este nodo se suscribe al tópico **falconPos** que trae el paquete *ROSfalcon* para recibir la posición del falcon, realiza el acoplamiento de espacios de trabajo y publica los valores que van a recibir los servomotores (robot delta). También lanza automáticamente la interfaz de comunicación y visualización del proceso (Figura 4.9). En el (Anexo: A.8) se puede ver la implementación del nodo publicador-subscriptor).

- Nombre del archivo: *pos_falcon.py*.
- Nombre del Nodo: *tele_remota*.
- Nombre del topic: *falconPos*.
- Recibe: Posición del falcon.
- Publica: Valores que van a recibir los servomotores y muestra la interfaz.

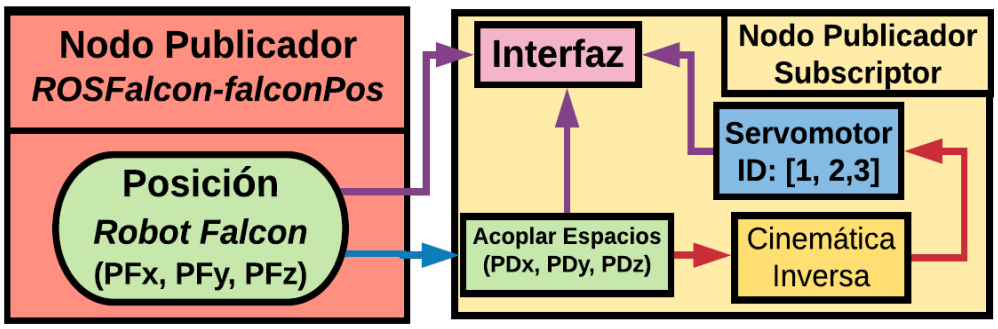


Figura 4.9: Nodo Publicador-Subscriptor.
Fuente: Autor.

Interfaz: Mostrará la posición del robot falcon, los valores que estan recibiendo cada uno de los motores y el ángulo en el que se encuentran, también la posición del robot paralelo delta (Figura 4.10). También tiene un botón para cerrar la interfaz.

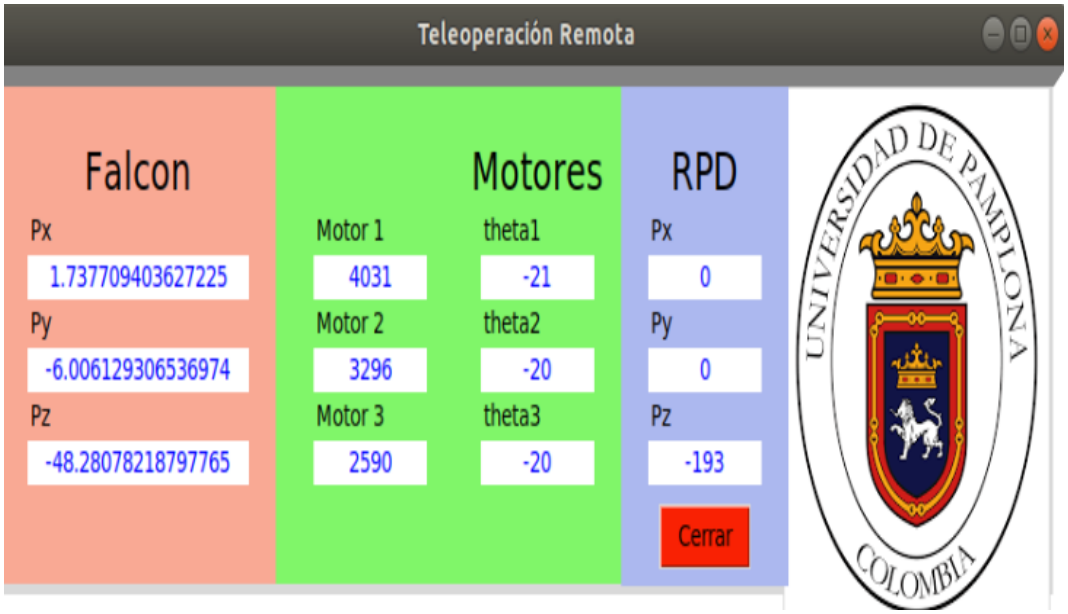


Figura 4.10: Interfaz de la Teleoperación Remota.
Fuente: Autor.

4.2.3. Parámetros del Robot Novint Falcon.

Teniendo los parámetros dimensionales del robot paralelo delta y del robot háptico *Novint Falcon*, hay que realizar un acople de los espacios de trabajo, como el espacio de trabajo del robot *Novint Falcon* es muy pequeño hay que realizar un escalamiento o mapeo, para darle un mayor rango de movilidad al robot paralelo Delta. La Tabla 4.6 resume las dimensiones espaciales del robot *falcon*.

Las ecuaciones 4.13 y 4.14 muestran el rango de valores del robot *falcon* y del robot paralelo delta respectivamente:

$$\triangle falcon = (maxfalcon - minfalcon) \tag{4.13}$$

$$\triangle delta = (maxdelta - mindelta) \tag{4.14}$$

Con los rangos de valores se calcula el factor de escala. Como se requiere ampliar el espacio de trabajo, se divide el rango del robot delta entre el rango del robot *falcon*:

$$factordeescala = (\triangle delta / \triangle falcon) \tag{4.15}$$

El nuevo valor del robot delta (V_{delta}) se obtiene mediante la ecuación 4.16. Donde *mindelta* es el valor mínimo del robot delta, *minfalcon* es el valor mínimo del robot

falcon y (V_{falcon}) es el valor del robot *falcon* que se quiere mapear:

$$V_{delta} = mindelta + (V_{falcon} - minfalcon) * factordeescala) \tag{4.16}$$

Parámetros del Novint Falcon		
Eje	**Mínimo(mm)**	**Máximo (mm)**
X	-52	56
Y	-53	51
Z	-50	-50

Tabla 4.6: Rango de Movilidad del Novint Falcon.

4.2.4. Escalamiento del eje X

La Tabla 4.3 da los rangos de valores para cada eje del robot delta y la Tabla 4.6 da los rangos de valores para cada eje del robot *falcon*.

Con la ecuación 4.17 se calcula el valor del rango para el eje X del robot falcon:

$$\triangle falconX = 56 - (-52) = 108 \tag{4.17}$$

Con la ecuación 4.18 se calcula el valor del rango para el eje X del robot delta:

$$\triangle deltaX = 100 - (-100) = 200 \tag{4.18}$$

El factor de escala para el eje X ($factorescalaX$) se calcula con la ecuacion 4.19:

$$factorescalaX = deltaX/falconX = 1,852 \tag{4.19}$$

Con la ecuación 4.20 se calcula el valor que toma la coordenada **X** del robot delta paralelo (PD_X) dependiendo del valor de la coordenada **X** del robot *falcon* (PF_X):

$$PD_X = -100 + (PF_X - (-52)) * 1,852 \tag{4.20}$$

Por *ejemplo*; si $PF_X = 10$ mm, entonces el valor que va a recibir el robot delta es:

$PD_X = -100 + (10 - (-52)) * 1,852 = \underline{14,82mm}$

4.2.5. Escalamiento del eje Y

Con la ecuación 4.21 se calcula el valor del rango para el eje Y del robot *falcon*:

$$\triangle falconY = 51 - (-53) = 104 \tag{4.21}$$

Con la ecuación 4.22 se calcula el valor del rango para el eje Y del robot delat:

$$\triangle deltaY = 180 - (-150) = 330 \tag{4.22}$$

El factor de escala para el eje Y ($factorescalaY$) se calcula con la ecuacion 4.23:

$$factorescalaY = deltaY/falconY = 3,173 \tag{4.23}$$

Con la ecuación 4.24 se calcula el valor que toma la coordenada **Y** del robot delta paralelo (PD_Y) dependiendo del valor de la coordenada **Y** del robot *falcon* (PF_Y):

$$PD_Y = -150 + (PF_Y - (-53)) * 3,173 \tag{4.24}$$

Por ejemplo; si $PF_Y = 20$ mm, entonces el valor que va a recibir el robot delta es:

$PDy = -150 + (20 - (-53)) * 3,173 = \underline{81,629}$

4.2.6. Escalamiento del eje Z

Con la ecuación 4.25 se calcula el valor del rango para el eje Z del robot *falcon*:

$$\triangle falconZ = 50 - (-50) = 100 \tag{4.25}$$

Con la ecuación 4.26 se calcula el valor del rango para el eje Z del robot delta:

$$\triangle deltaZ = -188 - (-338) = 150 \tag{4.26}$$

El factor de escala para el eje Z ($factorescalaZ$) se calcula con la ecuacion 4.27:

$$factorescalaZ = deltaZ/falconZ = 1,5 \tag{4.27}$$

Con la ecuación 4.28 se calcula el valor que toma la coordenada **Z** del robot delta paralelo (PD_Z) dependiendo del valor de la coordenada **Z** del robot *falcon* (PF_Z):

$$PD_z = -338 + (PF_Z - (-50)) * 1,5 \tag{4.28}$$

Por ejemplo; si PF_Z = -30 mm, entonces el valor que va a recibir el robot delta es:

$DP_Z = -338 + (-30 - (-50)) * 1,5 = \underline{-308mm}$

Característica del Nodo Subscriptor: Este nodo recibe los valores del nodo anterior (*tele_remota*) y se los envía de manera directa a los servomotores. En el (Anexo: A.9) puede ver la implementación del nodo subscriptor (Figura 4.11).

- Nombre del archivo: *sub_falcon.py*.
- Nombre del Nodo: *subsfalcon*.

- Nombre del topic: *ir_pos*.
- Recibe: Posición del falcon.
- Publica: Valores que van a recibir los motores.

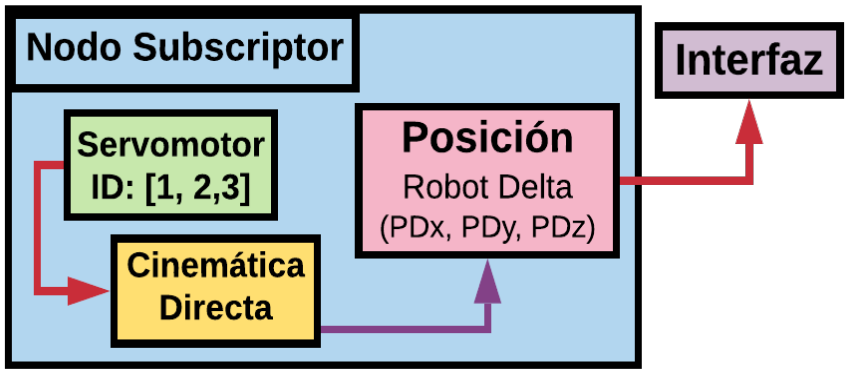


Figura 4.11: Nodo Subscriptor de la Teleoperación Remota.
Fuente: Autor.

La (Figura 4.12) muestra la la arquitectura del paquete completo, el cual guarda los archivos de la teleoperación local y de la teleoperación remota. En la siguiente dirección encontrará información más detallada: `https://github.com/IngLuisZambrano/Teleoperacion_falcon`.

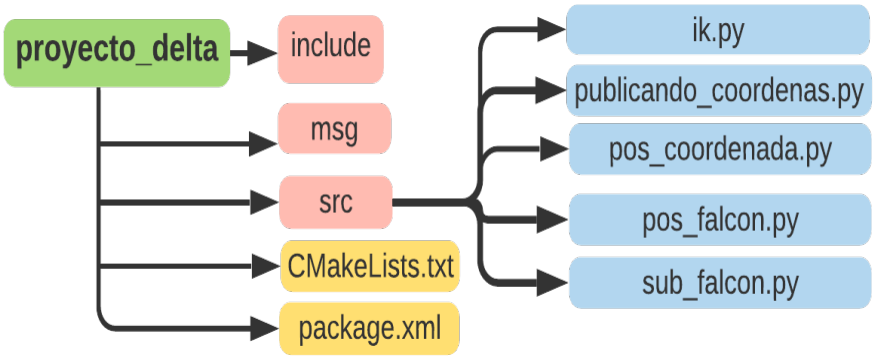


Figura 4.12: Topología del Paquete completo.
Fuente: Autor.

5. Resultados

5.1. Base del Robot Paralelo Delta

Las (Figuras 5.1 y 5.2) muestran el robot paralelo delta con el que se desarrolló la pasantía de investigación. El robot delta tiene 3 servomotores Dynamixel MX-64, la estructura esta hecho en aluminio.

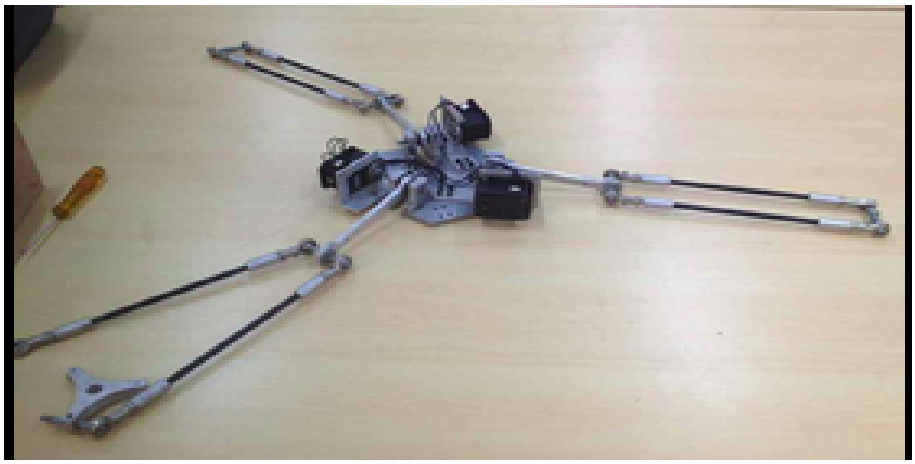


Figura 5.1: Prototipo del Robot delta.
Fuente: Autor.

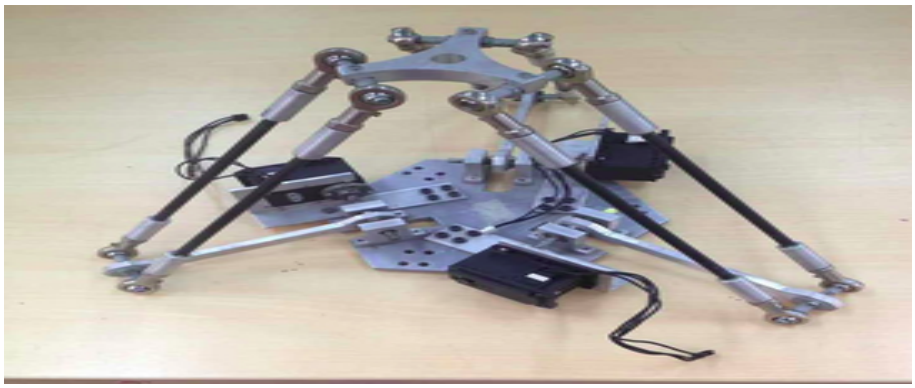


Figura 5.2: Armando el Robot Paralelo Delta.
Fuente: Autor.

La (Figura 5.3) muestra la base que se construyó como soporte para el robot paralelo delta, esta hecha en *madera*.

Figura 5.3: Base del RPD
Fuente: Autor.

5.2. Interfaz Teleoperación Local.

Esta interfaz cuenta con 2 botones, uno para cerrar la comunicación y otro con una configuración predeterminada que enviá al robot a una posición predeterminada **(HOME)** (Figura 5.4). Además muestra la posición del efector final del robot paralelo delta y la posición angular de los servomotores:

(a) Botón HOME. **(b)** Botón Cerrar.

Figura 5.4: Mensajes de advertencias.
Fuente: Autor.

La (Figura 5.5) muestra el esquema general de conexión de la teleoperación local. Donde */pub_posicion* es el nodo que publica la posición (Px, Py, Pz) a la cual se tiene que mover el robot paralelo delta, el nodo */tele_local* se subscribe al nodo */pub_posicion* mediante el tema (*topic* en inglés) */coordenadas* para recibir lo que sé está publicando, (*rosout* es el nodo maestro que se encarga de gestionar la comunicación y controlar el intercambio de datos.

Topic: Es un método de comunicación que utiliza **ROS** para la transmisión de datos o información.

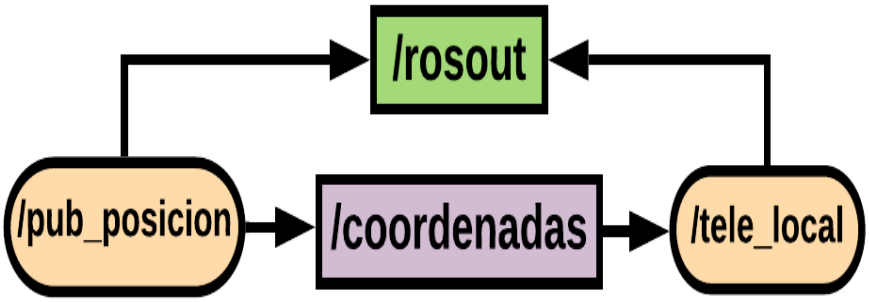


Figura 5.5: Esquema de Conexión de la Teleoperación Local.
Fuente: Autor.

5.3. Interfaz Teleoperación Remota.

Esta interfaz muestra un poco más de información. En ella se puede observar la posición del robot *falcon*, la posición angular y el valor que se le enviá a los servomotores, y la posición del efector final del robot paralelo delta (Figura 4.10).

La (Figura 5.6) muestra el esquema general de conexión de la teleoperación remota. Donde *ROSfalcon* es el nodo que esta publicando la posición (PFx, PFy, PFz) del robot *falcon*, el nodo */tele_remota* se subscribe a ese nodo publicador mediante el tema */falconPos* para poder obtener la posición y publicar una nueva terna (PDx, PDy, PDz), el nodo */subs_falcon* se subscribe al nodo */tele_remota* mediante el tema */ir_pos* para recibir la nueva posición y enviarsela al robot paralelo delta, */rosout* es el nodo maestro.

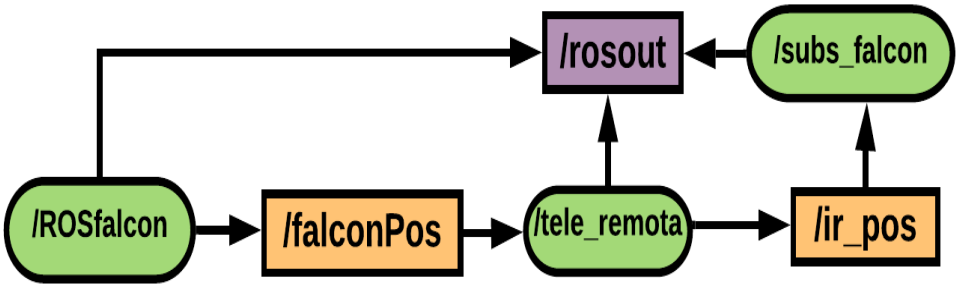
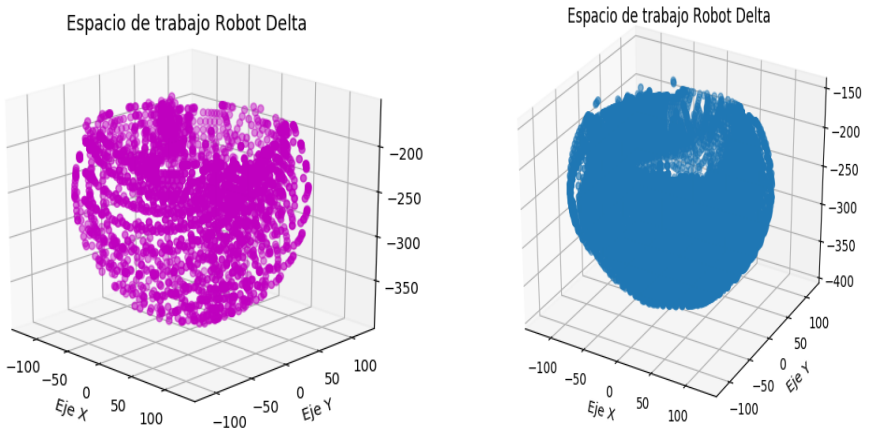


Figura 5.6: Esquema de Conexión de la Teleoperación Remota.
Fuente: Autor.

5.3.1. Espacio de Trabajo del Robot Paralelo Delta.

La (Figura 5.7) muestra el espacio de trabajo del robot delta, el cual fue hallado sin tener en cuenta sus restricciones mecánicas, solo se tuvieron en cuenta las longitudes de los eslabones y el rango de movilidad del por cada eje del robot. En el A.1) está la implementación del espacio de trabajo del robot delta.

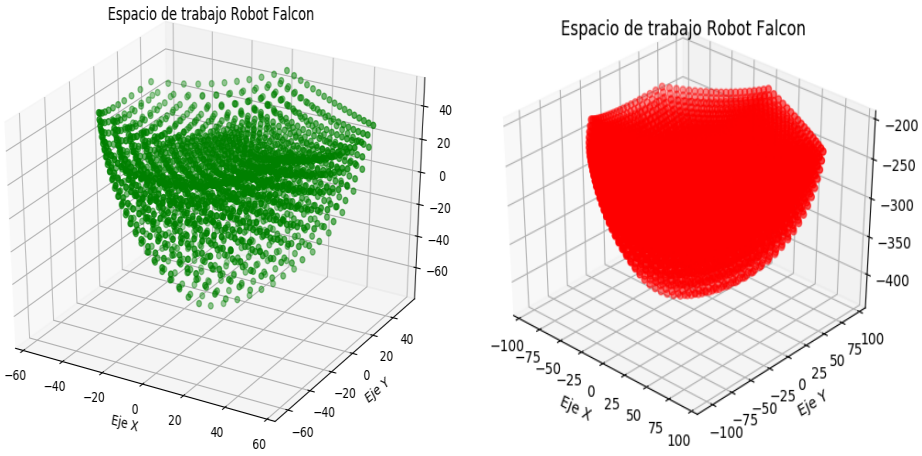


(a) 8000 puntos. **(b)** 10000 puntos.

Figura 5.7: Espacio de trabajo del Robot Delta.

5.3.2. Espacio de Trabajo del Robot *falcon*.

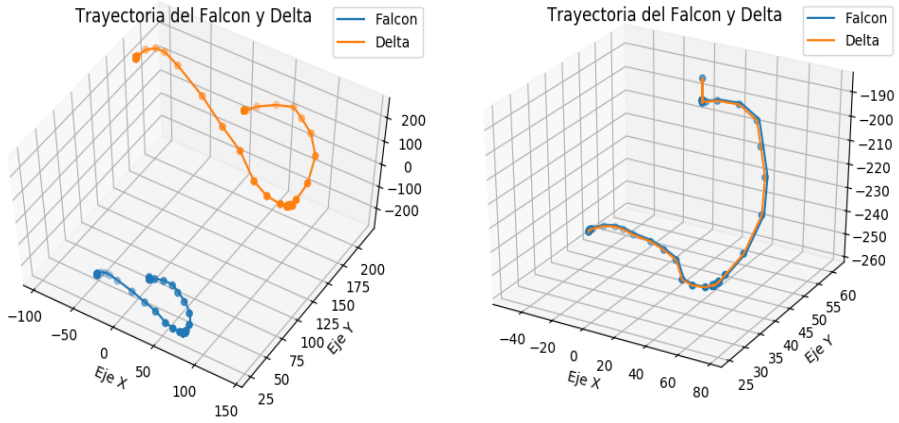
El espacio de trabajo del robot *falcon* (Figura 5.8), se encontró con el mismo procedimiento utilizado para hallar el espacio de trabajo del robot delta. En el (Anexo: A.2) se encuentra la implementación del espacio de trabajo del robot *falcon*.



(a) Sin Escala. **(b)** Con Escala.

Figura 5.8: Espacio de trabajo del Robot Novint *Falcon*.
Fuente: Autor.

Es muy importante el acoplamiento de los espacios de trabajo entre el robot delta y el robot *falcon* para tener un mejor control.

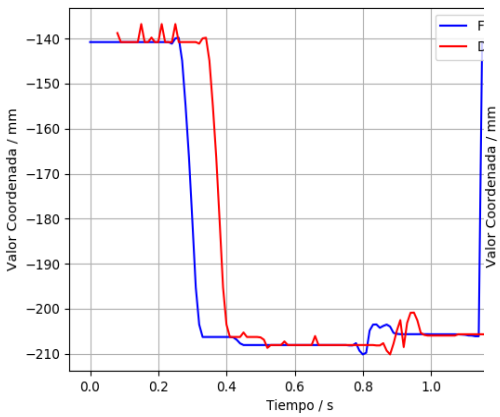


(a) Sin acoplamiento. **(b)** Con acoplamiento.

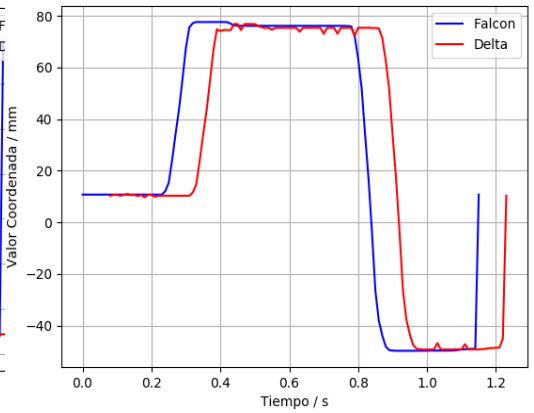
Figura 5.9: Trayectoria 1.
Fuente: Autor.

La (Figura 5.9) muestra una trayectoria descrita por el robot *falcon* y que el robot delta tiene que seguir. En la (**parte a**) se puede ver el comportamiento de ambos robots sin acoplamiento del espacio de la tarea y en la (**parte b**) se puede ver como mejor notablemente la trayectoria descrita por ambos robots con el acoplamiento de espacios de la tarea.

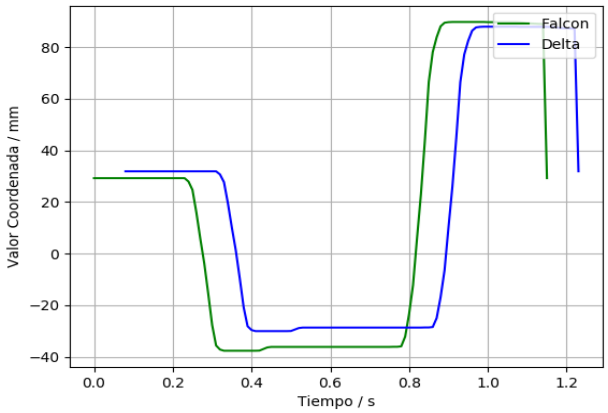
En la (Figura 5.10), se puede observar como varia la posición del robot *falcon* y del robot delta a lo largo de cada uno de los ejes, también se puede detallar como hay un retardo de tiempo por parte del robot delta cuando sigue la trayectoria descrita por el robot *falcon*.



(a) Visualización del Eje X.



(b) Visualización del Eje Y.



(c) Visualización del Eje Z.

Figura 5.10: Trayectoria 1.
Fuente: Autor.

En la (Figura 5.11) se puede observar otra curva realizada por el robot falcon y que el robot delta siguió de manera muy precisa, con buenos resultados. En la (Figura 5.12) se puede mirar el comportamiento de cada uno de los ejes, donde el retardo de tiempo es muy pequeño, el robot *falcon* se movio a una velocidad promedio.

La (Figura 5.13) muestra otra trayectoria realizada a mayor velocidad por el robot *falcon*. Se puede observar como disminuye la precisión por parte del robot paralelo delta para seguir dicha trayectoria.

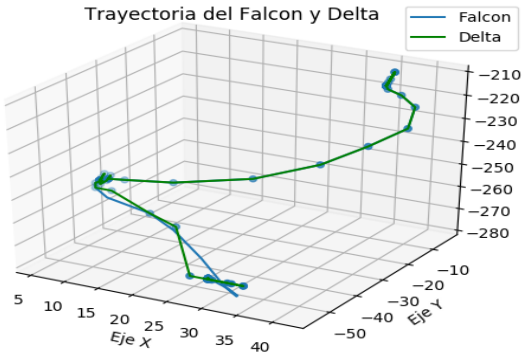
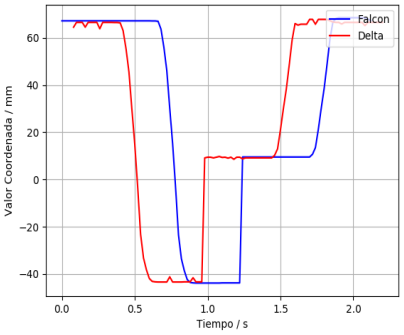
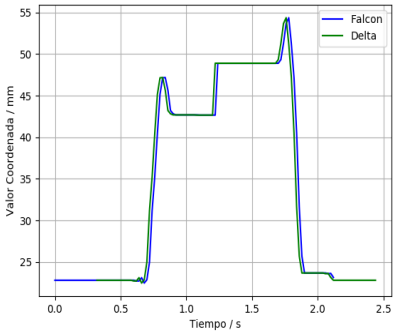


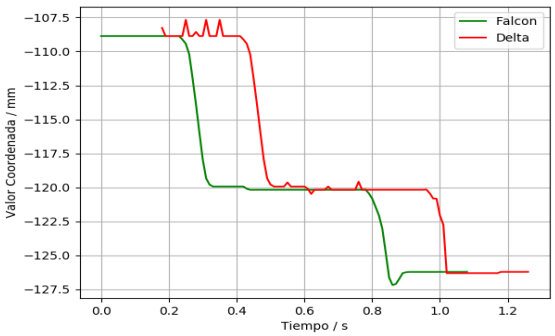
Figura 5.11: Trayectoria 2.
Fuente: Autor.



(a) Visualización del Eje X.



(b) Visualización del Eje Y.



(c) Visualización del Eje Z.

Figura 5.12: Trayectoria 2.
Fuente: Autor.

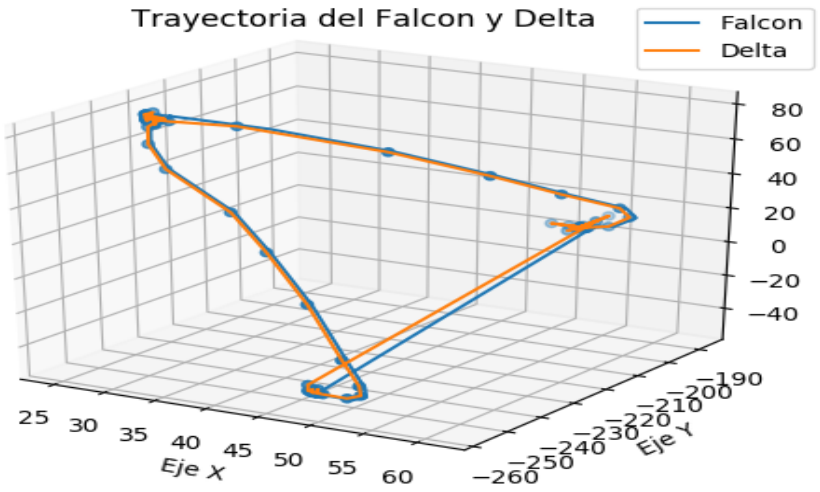
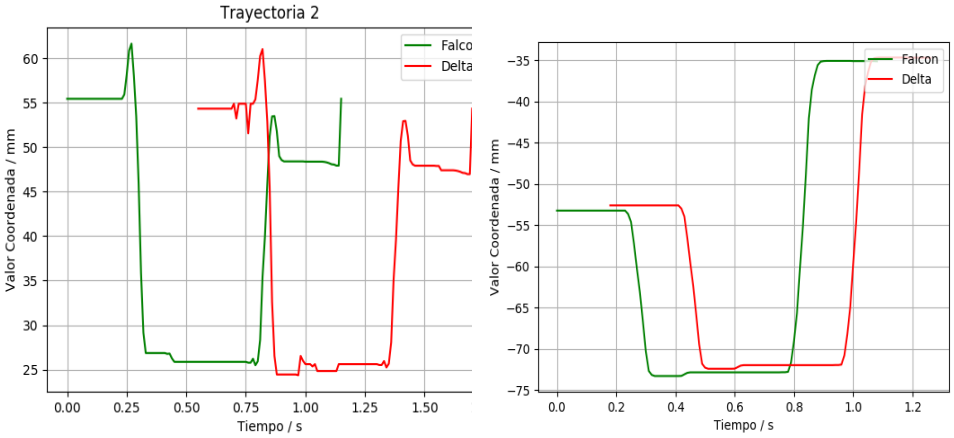


Figura 5.13: Trayectoria 3, Falcon a mayor velocidad.
Fuente: Autor.



(a) Visualización del Eje X. **(b)** Visualización del Eje Y.

Figura 5.14: Velocidad del falcon vs Presición.
Fuente: Autor.

6. Conclusiones y vías futuras

6.1. Conclusiones

Al momento de llevar a cabo la interfaz de la teleoperación remota se presentaron inconvenientes respecto a la comunicación, debido a que el robot *falcon* presentaba una alta tasa de muestreo que saturaba el canal de comunicación y ocasionaba que los servomotores se bloquearán. Para solucionar dicho problema se restringió la cantidad de datos que los servomotores iban a recibir, dificultad que se intensificaba cuando se movía al robot *falcon* más rápido, por que el robot delta se demoraba más en realizar la trayectoria descrita.

Para obtener mejores resultados en cuanto a la comunicación y transmisión de los datos se implementaron dos interfaces, una para la teleoperación local y otra para la teleoperación remota, para poder manipular de manera independiente al robot delta, también para poder simplificar la programación de las interfaces y evitar problemas por tener varios dispositivos conectados al mismo PC.

El sistema de conexión eléctrico (servomotores y tarjetas de comunicación) en términos generales mostraron un excelente desempeño, ya que facilitó el no tener que utilizar mucho cableado, simplificando él envió de los datos al robot delta. Por otro lado, no se tuvieron problemas de compatibilidad con la versión de ROS (Melodic) utilizada.

Adicionalmente, por medio del análisis cinemático se implementó un algoritmo en Python para obtener el espacio de trabajo del robot delta y del robot *falcon*, el cuál consistía en dibujar un espacio arbitrario (semi-esfera), tomar puntos al azar y por medio de las cinemáticas ver si esos puntos pertenecían o no al espacio de trabajo (*nube de puntos*) de los robots.

Durante el desarrollo de la tesis de investigación se realizaron todas las configuraciones y pruebas de funcionamiento, creando un paquete que contiene toda la información necesaria para facilitar el acceso a lo desarrollado durante la pasantía de investigación.

6.2. Vías futuras

- Analizar de manera más de detallada la cinemática y dinámica del robot paralelo delta, para poder utilizarlo en aplicaciones que requieran de un alto control y que el tiempo de retardo sea mínimo.
- Estudiar a fondo otros tipos de robots paralelos y desarrollar un paquete o herramientas para la simulación de dichos robots, ya que muchas veces estos son de difícil acceso. Lo cual facilitaría el estudio y análisis por parte de personas interesadas con la robótica paralela y la teleoperación.
- Poder realizar una teleoperación remota a larga distancia o desde dos computadoras diferentes.
- Diseñar mi propio prototipo del Robot Paralelo Delta, para seguir de manera independiente y más practica con la investigación.

Bibliografía

[1] Eduardo Izaguirre, Luis Hernández, Ernesto Rubio, Pablo J Prieto, and Arian Hernández. Control desacoplado de plataforma neumática de 3-gdl utilizada como simulador de movimiento. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, 8(4):345–356, 2011.

[2] FM Sánchez Martín, F Millán Rodríguez, J Salvador Bayarri, J Palou Redorta, F Rodríguez Escovar, S Esquena Fernández, and H Villavicencio Mavrich. Historia de la robótica: de arquitas de tarento al robot da vinci (parte i). *Actas Urológicas Españolas*, 31(2):69–76, 2007.

[3] Brian David Pajares Correa. Diseño del sistema mecánico de un equipo para rehabilitación de la muñeca usando mecanismos paralelos.

[4] Ernesto Cerveró Meliá, Pablo Ferrer Gisbert, and Salvador F Capuz-Rizo. El diseño basado en analogías en la obra de leonardo da vinci. 2018.

[5] Fernando Reyes. *Robótica-Control de robots manipuladores*. Alfaomega grupo editor, 2011.

[6] Álvaro Augusto Vejarano Anzola. Cine: Yo robot. *Revista Policía y Seguridad Pública*, pages 421–425, 2014.

[7] Alfonso Muñoz Corcuera. Esclavos y superhombres: la ética en los relatos de isacc asimov. 2009.

[8] David Marshall and Christina Bredin. Historia de un éxito. *Revista ABB*, 2(2008): 56–62, 2008.

[9] Charles G Burgar, Peter S Lum, Peggy C Shor, HF Machiel Van der Loos, et al. Development of robots for rehabilitation therapy: The palo alto va/stanford experience. *Journal of rehabilitation research and development*, 37(6):663–674, 2000.

[10] Jaime Rolando Heredia Velasteguí. Diseño e implementación de dos controladores para los brazos robóticos: Antropomórfico y scara, utilizando microcontroladores atmel avr´ s para el laboratorio de robótica de la universidad politécnica salesiana campus sur. B.S. thesis, 2013.

[11] FM Sánchez-Martín, F Millán Rodríguez, J Salvador-Bayarri, V Monllau Font, J Palou Redorta, H Villavicencio Mavrich, and P Jiménez Schlegl. Historia de la robótica: de arquitas de tarento al robot da vinci.(parte ii). *Actas Urológicas Españolas*, 31(3):185–196, 2007.

[12] Valdez Vidal and Laura. Sensorización de un robot paralelo para aplicaciones médicas. 2018.

[13] Victor Ng-Thow-Hing, Jongwoo Lim, Joel Wormer, Ravi Kiran Sarvadevabhatla, Carlos Rocha, Kikuo Fujimura, and Yoshiaki Sakagami. The memory game: Creating a human-robot interactive scenario for asimo. In *2008 IEEE/RSJ international conference on intelligent robots and systems*, pages 779–786. IEEE, 2008.

[14] Jesús Retto. Sophia, first citizen robot of the world. *ResearchGate https://www. researchgate. net*, pages 2–9, 2017.

[15] Katherin Duarte Barón and Carlos Borrás Pinilla. Generalidades de robots paralelos. *Visión electrónica*, (1):102–112, 2016.

[16] Lung-Wen Tsai. *Robot analysis: the mechanics of serial and parallel manipulators*. John Wiley & Sons, 1999.

[17] José Serracín, Iveth Moreno, Tirone Vásquez, and Isaac Bonilla. Prototipo de robot paralelo delta para fortalecer el proceso educativo a nivel superior. In *Memorias de Congresos UTP*, pages 155–160, 2017.

[18] Jorge Alberto Gudiño-Lau, Janeth Alcalá-Rodríguez, Henrry Narrarro, Daniel Velez-Díaz, Saida Charre-Ibarra, et al. Diseño y modelo cinemático de un robot delta para el diagnóstico y rehabilitación. *XIKUA Boletín Científico de la Escuela Superior de Tlahuelilpan*, 6(11), 2018.

[19] Anchante Guimaraes and Cromwell Steven. Modelación y simulación dinámica del mecanismo paralelo tipo plataforma de stewart-gough usado en un simulador de marcha. 2014.

[20] Han-Sung Kim. Development of a new 6-dof parallel-type motion simulator. *Journal of The Korean Society of Manufacturing Technology Engineers*, 19(2):171–177, 2010.

[21] Jamith Bermúdez Galvis, Omar Andrés Muñoz Monsalve, et al. Fabricación y control de posición de un robot paralelo tipo deltacon3 grados de libertad. 2014.

[22] José Serracín, Iveth Moreno, Tirone Vásquez, and Isaac Bonilla. Prototipo de robot paralelo delta para fortalecer el proceso educativo a nivel superior. In *Memorias de Congresos UTP*, pages 155–160, 2017.

[23] Lorena Alejandra Fernández Yánez and Luisa Fernanda Sotomayor Reinoso. Análisis cinemático inverso y directo del robot paralelo. Master's thesis, Quito, 2016., 2016.

[24] César Álvarez, Roque Saltaren, Rafael Aracil, and Cecilia García. Concepción, desarrollo y avances en el control de navegación de robots submarinos paralelos: El robot remo-i. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, 6(3):92–100, 2009.

[25] Jorge Esteban Martínez Macancela. Desarrollo de un robot delta para aplicaciones de manipulación de objetos y visión artificial mediante el sensor kinect. B.S. thesis, 2018.

[26] Diana Tumbaco Mendoza and Wilmer Quimbita Zapata. Diseño y construcción de un prototipo de robot delta con implementación de un cortador láser cnc utilizando la plataforma robotic operating system (ros) para la elaboración de artículos publicitarios.

[27] Jhonattan Didier Rueda Florez et al. Metodologia para el diseño de un robot paralelo industrial tipo delta. 2013.

[28] Ricardo Celi, Ana Sempértegui, Derlin Morocho, David Loza, Darwin Alulema, and Mariela Proaño. Study, design and construction of a 3d printer implemented through a delta robot. In *2015 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pages 717–722. IEEE, 2015.

[29] Miguel Hernando Gutiérrez. *Arquitectura de control, planificación y simulación para teleprogramación de robots*. PhD thesis, Tesis Doctoral. Universidad Politécnica de Madrid. Madrid, 2002.

[30] H Villavicencio Mavrich. Cirugía laparoscópica avanzada robótica da vinci: origen, aplicación clínica actual en urología y su comparación con la cirugía abierta y laparoscópica. *Actas Urológicas Españolas*, 30(1):1–12, 2006.

[31] José Marıa Sabater Navarro. Desarrollo de una interfaz kinestésica paralela y experimentación en control de sistemas hápticos y teleoperados. *Departamento de Ingeniería de Sistemas Industriales*, page 220, 2003.

[32] Santiago Alfaro Ballesteros. Sistema de teleoperación mediante una interfaz natural de usuario. Master's thesis, 2012.

[33] Steven Martin and Nick Hillier. Characterisation of the novint falcon haptic device for application as a robot manipulator. In *Australasian Conference on Robotics and Automation (ACRA)*, pages 1–9. Citeseer, 2009.

[34] Nima Karbasizadeh, Mojtaba Zarei, Ali Aflakian, Mehdi Tale Masouleh, and Ahmad Kalhor. Experimental dynamic identification and model feed-forward control of novint falcon haptic device. *Mechatronics*, 51:19–30, 2018.

[35] Juan Domingo Gálvez Cobo. Teleoperación del robot nao mediante dispositivos móviles android. B.S. thesis, Universidad Carlos III de Madrid, 2012.

[36] Juan Sebastián Méndez Rubiano et al. Sistema robótico teleoperado por captura de movimiento. B.S. thesis, Bogotá-Uniandes, 2010.

[37] Carlos Escolano and Javier Minguez. Sistema de teleoperación multi-robot basado en interfaz cerebro-computador. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, 8(2):16–23, 2011.

[38] J Tafur, C Peña, Cecilia Garcia, and R Aracil. Implementación de una plataforma experimental para un sistema de teleoperación robótica en tiempo real. *Revista Iberoamericana de Sistemas, Cibernética e Informática Sistemas, Cibernética e Informática*, 7(1):69–74, 2010.

[39] Alexander Cerón Correa. Sistemas robóticos teleoperados. *Ciencia e Ingeniería Neogranadina*, 15:62–72, 2005.

[40] Richard Volpe, J Balaram, Timothy Ohm, and Robert Ivlev. The rocky 7 mars rover prototype. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'96*, volume 3, pages 1558–1564. IEEE, 1996.

[41] R Valero, YH Ko, S Chauhan, O Schatloff, A Sivaraman, RF Coelho, F Ortega, KJ Palmer, Rafael Sánchez-Salas, H Davila, et al. Cirugía robótica: Historia e impacto en la enseñanza. *Actas urológicas españolas*, 35(9):540–545, 2011.

[42] Isaac Asimov. *I, robot.* Spectra, 2004.

A. Anexo I

A.1. ROS

Es un sistema meta-operativo de código abierto que brinda de manera análoga los servicios que puede ofrecer un sistema operativo clásico. Se ha convertido en una de las herramientas más usadas en aplicaciónes robóticas.

A.1.1. Conceptos Importantes de ROS

Conceptos de ROS	
	Definición
master	Se comporta como un servido, gestiona la comunicacion entre nodos.
Nodos	Es un ejecutable, es la unidad más pequeña que se puede procesar en ROS
Package	Es la unidad básica de ROS, contiene toda la configuración para compilación de los nodos y la comunicación con otros paquetes.
topics	Es un mecanismo de comunicación para el intercambio de mensajes de manera asíncrona.
Service	Es un mecanismo de comunicación para el intercambio de mensajes de manera síncrona.
Action	Es un mecanismo de comunicación para el intercambio de mensajes de manera Asíncrona, es una combinación de topics y services
Parameters	Es un mecanismo de comunicación que utiliza variables globales para el intercambio de mensajes.

Tabla A.1: Conceptos Básicos

A.1.2. Comandos Importantes de ROS

Comandos de ROS	
	Definición
roscd	Moverse al directorio principal de ROS.
rosls	Mostar los archivos que contiene un paquete.
roscp	Copiar un archivo en otro paquete.
rosed	Editar un archivo o ejecutable.
roscore	(master) Lanzar el gestionador para la comunicación de paquetes.
rosrun	Ejecutar nodos.
roslaunch	Ejecutar varios nodos.
rosclear	Borrar archivos de registros de ROS.
rosnode	Muestra los nodos activos.
rostopic	Muestra información de un topico.
rosservice	Muestra información de un servicio.
rosmsg	Examina información de un archivo tipo message ROS.
rossrv	Examina información de un archivo tipo service ROS.
catkin create pkg	Crear automáticamente un paquete.
catkin make	Actualizar y compilar el espacio de trabajo de ROS.
rospack	Ver información detallada de un paquete.
rqt	Herramienta que permite ver la conexión de los nodos.
roswft	Ver y examinar el estado de ROS.
rosbag	Grabar y Reproducir un mensaje de ROS.
rosd	Consultar el índice del directorio de ROS.
rosversion	Mirar la versión de ROS.
rosversion	Mirar la versión de ROS.

Tabla A.2: Comandos Básicos

A.2. Dynamixel Motors

A continuación se muestran las principales áreas de control, características y parámetros de los motores dynamixels:

Especificaciones	
ítem	**Especificación**
MCU	ARM CORTEX-M3 (72 [MHz], 32Bit)
Position Sensor	Contactless absolute encoder (12Bit, 360 [°]) Maker : ams(www.ams.com), Part No : AS5045
Motor	Coreless(Maxon)
Baud Rate	8,000 [bps] 4.5 [Mbps]
Control Algorithm	PID control
Resolution	4096 [pulse/rev]
Backlash	20 [arcmin] (0.33 [°])
Operating Mode	Joint Mode (0 360 [°]) Wheel Mode (Endless Turn)
Weight	165 [g]
Dimensions (W x H x D)	40.2 x 61.1 x 41 [mm]
Gear Ratio	200:1
Stall Torque	5.5 [Nm](at 11.1 [V], 3.9 [A])
No Load Speed	58 [rev/min] (at 11.1 [V])
ID	254 ID (0 253)
Feedback	Position, Temperature, Load, Input Voltage, etc
Material	Full Metal Gear, Engineering Plastic(Front, Middle, Back), 1 Metal(Front)
Standby Current	100 [mA]
Input Voltage	10.0 14.8 [V] (Recomendado: 12.0 [V])
Operating Temperature	-5 +80[°C]
Protocol Type	TTL

Tabla A.3: Especificaciones básicas.

Especificaciones		
ítem	**Acceso**	**Valor inicial**
Model Number	R	310
ID	RW	1
Baud Rate	RW	34
CW Angle Limit	RW	250
CWW Angle Limit	RW	4095
Temperature Limit	RW	80
Min Voltage Limit	RW	60
Max Voltage Limit	RW	240
Max Torque	RW	1023
Alarm LED	RW	36
Shutdown	RW	36

Tabla A.4: Control de área EEPROM.

Especificaciones		
ítem	**Acceso**	**Valor inicial**
Torque Enable	RW	0
LED	RW	0
D Gain	RW	0
I Gain	RW	0
P Gain	RW	32
Goal Position	RW	-
Moving Speed	RW	-
Present Position	R	-

Tabla A.5: Control de área RAM.

A.3. Espacio de trabajo

A continuación se muestra el código para hallar el espacio de trabajo de los robots:

A.3.1. Robot Paralelo Delta

Código A.1: Espacio de Trabajo - Robot delta

```
1 #!/usr/bin/env python
2 import xlrd
3 from ik import *
4 import numpy as np
5 from pylab import *
6 import matplotlib.pyplot as plt
7 from mpl_toolkits.mplot3d import *
8
```

```
fig = plt.figure()
ax1 = fig.gca(projection = '3d')

xx = []
yy = []
zz = []

for i in range(-100,101,10):
    for j in range(-150,181,10):
        for k in range(-340,-181,10):
            rr = inverse(i,j,k)
            if rr[3] == 1:
                rf = forward(i, j, k)
                xx.append(rf[1])
                yy.append(rf[2])
                zz.append(rf[3])

ax1.scatter(xx,yy,zz, color = "g")
plt.title("Espacio de trabajo Robot Delta")
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.show()
```

A.3.2. Robot Novint *Falcon*

Código A.2: Espacio de Trabajo - Robot *falcon*

```
#!/usr/bin/env python
import xlrd
from ik import *
import numpy as np
from pylab import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import *

fig = plt.figure()
ax1 = fig.gca(projection = '3d')

xx = []
yy = []
zz = []

for i in range(-52,56,5):
    for j in range(-53,51,5):
        for k in range(-50,50,6):
            rr = inverse(i,j,k)
            if rr[3] == 1:
                rf = forward(i, j, k)
                xx.append(rf[1]*0.5)
                yy.append(rf[2]*0.5)
                zz.append(rf[3]*0.9-100)

ax1.scatter(xx,yy,zz, color = "r")
plt.title("Espacio de trabajo Robot Falcon")
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.show()
```

A.4. Códigos

A.4.1. Cinemática Directa e Inversa del robot Delta

Código A.3: Nombre del paquete: proyecto_delta

```
#!/usr/bin/env python
# coding: utf-8

# Original code from
# http://forums.trossenrobotics.com/tutorials/introduction-129/delta-robot-kinematics-3276/
# License: MIT

import math

# Specific geometry for bitbeambot:
# http://flic.kr/p/cYaQah
e = 35.0
f = 60.0
re = 280.0
rf = 140.0

# Trigonometric constants
s = 165*2
sqrt3 = math.sqrt(3.0)
pi = math.pi
sin120 = sqrt3 / 2.0
cos120 = -0.5
tan60 = sqrt3
sin30 = 0.5
tan30 = 1.0 / sqrt3

# Forward kinematics: (theta1, theta2, theta3) -> (x0, y0, z0)
# Returned {error code,theta1,theta2,theta3}
def forward(theta1, theta2, theta3):
    x0 = 0.0
    y0 = 0.0
    z0 = 0.0

    t = (f-e) * tan30 / 2.0
    dtr = pi / 180.0

    theta1 *= dtr
    theta2 *= dtr
    theta3 *= dtr

    y1 = -(t + rf*math.cos(theta1) )
    z1 = -rf * math.sin(theta1)

    y2 = (t + rf*math.cos(theta2)) * sin30
    x2 = y2 * tan60
    z2 = -rf * math.sin(theta2)

    y3 = (t + rf*math.cos(theta3)) * sin30
    x3 = -y3 * tan60
    z3 = -rf * math.sin(theta3)

    dnm = (y2-y1)*x3 - (y3-y1)*x2

    w1 = y1*y1 + z1*z1
    w2 = x2*x2 + y2*y2 + z2*z2
    w3 = x3*x3 + y3*y3 + z3*z3
```

```
57
58      # x = (a1*z + b1)/dnm
59      a1 = (z2-z1)*(y3-y1) - (z3-z1)*(y2-y1)
60      b1= -( (w2-w1)*(y3-y1) - (w3-w1)*(y2-y1) ) / 2.0
61
62      # y = (a2*z + b2)/dnm
63      a2 = -(z2-z1)*x3 + (z3-z1)*x2
64      b2 = ( (w2-w1)*x3 - (w3-w1)*x2) / 2.0
65
66      # a*z^2 + b*z + c = 0
67      a = a1*a1 + a2*a2 + dnm*dnm
68      b = 2.0 * (a1*b1 + a2*(b2 - y1*dnm) - z1*dnm*dnm)
69      c = (b2 - y1*dnm)*(b2 - y1*dnm) + b1*b1 + dnm*dnm*(z1*z1 - re*re)
70
71      # discriminant
72      d = b*b - 4.0*a*c
73      if d < 0.0:
74          return [1,0,0,0] # non-existing povar. return error,x,y,z
75
76      z0 = -0.5*(b + math.sqrt(d)) / a
77      x0 = (a1*z0 + b1) / dnm
78      y0 = (a2*z0 + b2) / dnm
79      print (x0,y0,z0)
80      return [0,x0,y0,z0]
81  rs = forward(30,60, 45)
82  #print "{:.4f}, {:.4f}, {:.4f} ".format(rs[1], rs[2], rs[3])
83  # Inverse kinematics
84  # Helper functions, calculates angle theta1 (for YZ-pane)
85  def angle_yz(x0, y0, z0, theta=None):
86      y1 = -0.5*0.57735*f # f/2 * tg 30
87      y0 -= 0.5*0.57735*e # shift center to edge
88      # z = a + b*y
89      a = (x0*x0 + y0*y0 + z0*z0 + rf*rf - re*re - y1*y1) / (2.0*z0)
90      b = (y1-y0) / z0
91
92      # discriminant
93      d = -(a + b*y1)*(a + b*y1) + rf*(b*b*rf + rf)
94      if d<0:
95          return [1,0] # non-existing povar. return error, theta
96
97      yj = (y1 - a*b - math.sqrt(d)) / (b*b + 1) # choosing outer povar
98      zj = a + b*yj
99      theta = math.atan(-zj / (y1-yj)) * 180.0 / pi + (180.0 if yj>y1 else 0.0)
100
101     return [0,theta] # return error, theta
102
103 def inverse(x0, y0, z0):
104     theta1 = 0
105     theta2 = 0
106     theta3 = 0
107     status = angle_yz(x0,y0,z0)
108
109     if status[0] == 0:
110         theta1 = status[1]
111         status = angle_yz(x0*cos120 + y0*sin120,
112                                     y0*cos120-x0*sin120,
113                                     z0,
114                                     theta2)
115     if status[0] == 0:
116         theta2 = status[1]
117         status = angle_yz(x0*cos120 - y0*sin120,
118                                     y0*cos120 + x0*sin120,
119                                     z0,
120                                     theta3)
```

```
121     theta3 = status[1]
122     return [theta1,theta2,theta3]
123 rr = inverse(0, 0, -419.907)
124 print rr
125 #print "{:.4}, {:.4}, {:.4}".format(rr[0], rr[1], rr[2])
```

A.4.2. CMakeList.txt

Código A.4: CMakeLists.txt

```
1  cmake_minimum_required(VERSION 2.8.3)
2  project(proyecto_delta)
3
4  find_package(catkin REQUIRED COMPONENTS
5    message_generation
6    dynamixel_workbench_controllers
7    roscpp
8    rospy
9    std_msgs
10 )
11
12 # catkin_python_setup()
13
14 ################################################
15 ## Declare ROS messages, services and actions ##
16
17 add_message_files(
18    FILES
19    posicion.msg
20 )
21
22 ## Generate added messages and services with any dependencies listed ↩
     ↪ here
23 generate_messages(
24    DEPENDENCIES
25    std_msgs
26 )
27
28 ###################################
29 ## catkin specific configuration ##
30
31 catkin_package(
32 # INCLUDE_DIRS include
33 # LIBRARIES proyecto_delta
34 # CATKIN_DEPENDS dynamixel_workbench_controllers roscpp rospy std_msgs
```

```
# DEPENDS system_lib
)

############
## Build ##

include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME}
# src/${PROJECT_NAME}/proyecto_delta.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ↩
    ↪ ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake ↩
    ↪ context
## The recommended prefix ensures that target names across packages don↩
    ↪ 't collide
# add_executable(${PROJECT_NAME}_node src/proyecto_delta_node.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following↩
    ↪  renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg ↩
    ↪ someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME ↩
    ↪ node PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}↩
    ↪ _EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
```

```
# ${catkin_LIBRARIES}
# )

##############
## Install ##
##############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# install(PROGRAMS
# scripts/my_python_script
# DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/↩
    ↪ building_executables.html
# install(TARGETS ${PROJECT_NAME}_node
# RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark libraries for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/↩
    ↪ building_libraries.html
# install(TARGETS ${PROJECT_NAME}
# ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
# LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
# RUNTIME DESTINATION ${CATKIN_GLOBAL_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
# DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
# FILES_MATCHING PATTERN "*.h"
# PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
# # myfile1
# # myfile2
# DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
```

```
# )

##############
## Testing ##
##############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_proyecto_delta.cpp)
# if(TARGET ${PROJECT_NAME}-test)
# target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

A.4.3. package.xml

Código A.5: package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>proyecto_delta</name>
  <version>0.0.0</version>
  <description>The proyecto_delta package</description>

  <maintainer email="luis@todo.todo">luis</maintainer>

  <license>TODO</license>

  <!-- <url type="website">http://wiki.ros.org/proyecto_delta</url> -->

  <author email="lumazaca17@hotmail.com">Luis Zambrano</author>

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>dynamixel_workbench_controllers</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>dynamixel_workbench_controllers</↩
      ↪ build_export_depend>
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>dynamixel_workbench_controllers</exec_depend>
  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>

  <export> </export>
</package>
```

A.4.4. Teleoperación Local - Nodo Pulicador

Código A.6: Nodo Publicador - Interfaz

```
#!/usr/bin/env python
# coding: utf−8
import rospy
from ik import ∗
import tkMessageBox
from Tkinter import ∗
from proyecto_delta.msg import posicion

global x, y, z, delta, deltaz

def principal():
    global x, y, z, delta, deltaz
    pub = rospy.Publisher('coordenadas', posicion, queue_size=10)
    rospy.init_node('pub_posicion', anonymous=True) #inicia el nodo publicador
    rate = rospy.Rate(10) #Hz
    vp = Tk()
    vp.title("Teleoperación Local")
    vp.geometry("400x200+0+0")
    vp.resizable(0, 0)
    # Configuración de Frame1
    frame1 = Frame(vp, width = 400, height = 400 )
    frame1.pack(side = "left", anchor = "n")
    frame1.config(bg="#86f1ef", cursor="hand2")
    canvas = Canvas(vp, width = 400, height = 400)
    canvas.pack()
    # Configuración de Barra de Menùs
    #barraMenu = Menu(vp)
    #vp.config(menu = barraMenu, width = 10, height = 2)
    #ayudaMenu = Menu(barraMenu)
    #barraMenu.add_cascade(label = "Ayuda", menu = ayudaMenu)

    tc3 = Label(frame1, bg = "#86f1ef")
    tc3.grid(row = 0, column = 0, padx = 2, pady = 1)
    tc4 = Label(frame1, bg = "#86f1ef")
    tc4.grid(row = 14, column = 0, padx = 2, pady = 1)

    te1 = Label(frame1, text = "Px", bg = "#86f1ef", font=("Comic Sans MS", 10))
    te1.grid(row = 3, column = 0, sticky = "w", pady = 0, padx = 20)
    te1.config(cursor = "pirate")
    td1 = Label(frame1, bg = "#ffffff", width = 20 )
    td1.grid(row = 4, column = 0, sticky = "e", pady = 1, padx = 20)
    td1.config(fg = "blue", justify = "center")

    te1r = Label(frame1, text = "theta1", bg = "#86f1ef", font=("Comic Sans MS", 10))
    te1r.grid(row = 3, column = 1, sticky = "w", pady = 0, padx = 20)
    te1r.config(cursor = "pirate")
    td1r = Label(frame1, bg = "#ffffff", width = 20 )
    td1r.grid(row = 4, column = 1, sticky = "e", pady = 1, padx = 20)
    td1r.config(fg = "blue", justify = "center")

    te2 = Label(frame1, text = "Py", bg = "#86f1ef", font=("Comic Sans MS", 10))
    te2.grid(row = 6, column = 0, sticky = "w", pady = 0, padx = 20)
    te2.config(cursor = "pirate")
    td2 = Label(frame1, bg = "#ffffff", width = 20 )
    td2.grid(row = 7, column = 0, sticky = "e", pady = 1, padx = 20)
    td2.config(fg = "blue", justify = "center")

    te2r = Label(frame1, text = "theta2", bg = "#86f1ef", font=("Comic Sans MS", 10))
    te2r.grid(row = 6, column = 1, sticky = "w", pady = 0, padx = 20)
```

```
60  te2r.config(cursor = "pirate")
61  td2r = Label(frame1, bg = "#ffffff", width = 20 )
62  td2r.grid(row = 7, column = 1, sticky = "e", pady = 1, padx = 20)
63  td2r.config(fg = "blue", justify = "center")
64
65  te3 = Label(frame1, text = "Pz", bg = "#86f1ef", font=("Comic Sans MS", 10))
66  te3.grid(row = 9, column = 0, sticky = "w", pady = 0, padx = 20)
67  te3.config(cursor = "pirate")
68  td3 = Label(frame1, bg = "#ffffff", width = 20 )
69  td3.grid(row = 10, column = 0, sticky = "e", pady = 1, padx = 20)
70  td3.config(fg = "blue", justify = "center")
71
72  te3r = Label(frame1, text = "tetha3", bg = "#86f1ef", font=("Comic Sans MS", 10))
73  te3r.grid(row = 9, column = 1, sticky = "w", pady = 0, padx = 20)
74  te3r.config(cursor = "pirate")
75  td3r = Label(frame1, bg = "#ffffff", width = 20 )
76  td3r.grid(row = 10, column = 1, sticky = "e", pady = 1, padx = 20)
77  td3r.config(fg = "blue", justify = "center")
78
79  x = 0
80  y = 0
81  z = -358
82  delta = 5
83  deltaz = 5
84  td1["text"] = x
85  td2["text"] = y
86  td3["text"] = z
87  motores = posicion()
88  motores.x = x
89  motores.y = y
90  motores.z = z
91  rospy.loginfo("Estoy publicando en el topic...")
92
93  def codigoboton1():
94      s = tkMessageBox.askyesno(message="¿Desea continuar?", title="Título")
95      if s == False:
96      vp.destroy()
97
98  def codigoboton2():
99  global x, y, z
100     t = tkMessageBox.askyesno(message="¿Desea enviar a Home al Robot?", title="Título")
101     if t == True:
102     motores.x = 0
103         motores.y = 0
104             motores.z = -358
105             td1["text"] = 0
106             td2["text"] = 0
107             td3["text"] = -358
108     x, y, z = [0, 0, -358]
109         pub.publish(motores)
110     return t
111
112
113 boton1= Button(frame1, text = "Cerrar", command = codigoboton1,
114     bg = "#f92103")
115 boton1.grid(row = 12, column = 1, pady = 2, padx = 4)
116
117 boton2= Button(frame1, text = "HOME", command = codigoboton2,
118     bg = "#36e60f")
119 boton2.grid(row = 12, column = 0, pady = 2, padx = 4)
120
121
122 def Capturarevento(event):
123 global x,y,z, delta, deltaz
```

```
124 if z < −410:
125     deltaz = 0.1
126
127 if z > −400:
128         deltaz = 5
129
130 if z >= −419.9:
131     if event.keysym == 'Up':
132         x = x + delta
133         td1["text"] = round(x, 4)
134         td2["text"] = round(y, 4)
135         td3["text"] = round(z, 4)
136         ri = inverse(x, y, z)
137         td1r["text"] = round(ri[0], 4)
138         td2r["text"] = round(ri[1], 4)
139         td3r["text"] = round(ri[2], 4)
140                 motores.x = x
141                 motores.y = y
142             motores.z = z
143             pub.publish(motores)
144     elif event.keysym == 'Down':
145         x = x − delta
146         td1["text"] = round(x, 4)
147         td2["text"] = round(y, 4)
148         td3["text"] = round(z, 4)
149         ri = inverse(x, y, z)
150         td1r["text"] = round(ri[0], 4)
151         td2r["text"] = round(ri[1], 4)
152         td3r["text"] = round(ri[2], 4)
153         motores.x = x
154                 motores.y = y
155             motores.z = z
156             pub.publish(motores)
157     elif event.keysym == 'Left':
158         y = y − delta
159         td1["text"] = round(x, 4)
160         td2["text"] = round(y, 4)
161         td3["text"] = round(z, 4)
162         ri = inverse(x, y, z)
163         td1r["text"] = round(ri[0], 4)
164         td2r["text"] = round(ri[1], 4)
165         td3r["text"] = round(ri[2], 4)
166         motores.x = x
167                 motores.y = y
168             motores.z = z
169             pub.publish(motores)
170     elif event.keysym == 'Right':
171         y = y + delta
172         td1["text"] = round(x, 4)
173         td2["text"] = round(y, 4)
174         td3["text"] = round(z, 4)
175         ri = inverse(x, y, z)
176         td1r["text"] = round(ri[0], 4)
177         td2r["text"] = round(ri[1], 4)
178         td3r["text"] = round(ri[2], 4)
179         motores.x = x
180                 motores.y = y
181             motores.z = z
182             pub.publish(motores)
183     elif event.keysym == 'z':
184         z = z + deltaz
185         td1["text"] = round(x, 4)
186         td2["text"] = round(y, 4)
187         td3["text"] = round(z, 4)
```

```
188             ri = inverse(x, y, z)
189             td1r["text"] = round(ri[0], 4)
190             td2r["text"] = round(ri[1], 4)
191             td3r["text"] = round(ri[2], 4)
192             motores.x = x
193                     motores.y = y
194                 motores.z = z
195                 pub.publish(motores)
196         elif event.keysym == 'm':
197             z = z - deltaz
198             td1["text"] = round(x, 4)
199             td2["text"] = round(y, 4)
200             td3["text"] = round(z, 4)
201             ri = inverse(x, y, z)
202             td1r["text"] = round(ri[0], 4)
203             td2r["text"] = round(ri[1], 4)
204             td3r["text"] = round(ri[2], 4)
205             motores.x = x
206                     motores.y = y
207                 motores.z = z
208                 pub.publish(motores)
209     else:
210         tkMessageBox.showinfo(message="Fuera de Rango", title="Error")
211         z = -419.9
212 
213     canvas.bind_all('<KeyPress>', Capturarevento)
214 
215     while not rospy.is_shutdown():
216         try:
217         rospy.loginfo("%f, %f, %f ", motores.x, motores.y, motores.z)
218                 vp.update()
219         vp.update_idletasks()
220             rate.sleep()
221     except:
222         break
223 
224     rospy.loginfo("Finalizando nodo...")
225 
226 if __name__ == '__main__':
227     try:
228         principal()
229     except rospy.ROSInterruptException:
230     pass
```

A.4.5. Teleoperación Local - Nodo Subscriptor

Código A.7: Nodo Subscriptor - Robot Delta

```
#!/usr/bin/env python
# coding: utf-8
import rospy
from ik import *
import tkMessageBox
from Tkinter import *
from dynamixel_sdk import *
from dxl.dxlchain import DxlChain
from proyecto_delta.msg import posicion

global motores
motores = DxlChain("/dev/ttyUSB0", rate=1000000)
motors = motores.get_motor_list()
print motors

def callback_function(data):
    ri = inverse(data.x, data.y, data.z)
    motor1 = int(round(3800 - 10.8888*ri[0]))
    print motor1
    motor2 = int(round(3050 - 11.8888*ri[1]))
    motor3 = int(round(2350 - 11.4444*ri[2]))
    motores.goto(1, motor1, speed = 20, blocking = False)
    motores.goto(2, motor2, speed = 20, blocking = False)
    motores.goto(3, motor3, speed = 20, blocking = False)
    rospy.loginfo("He recibido: Px = %f, Py = %f y Pz = %f", data.x, data.y, data.z)
    return [motor1, motor2, motor3]

rospy.init_node('tele_local', anonymous=True)
rospy.Subscriber("coordenadas", posicion, callback_function)


rospy.spin()
```

A.4.6. Teleoperación Remota - Nodo Pulicador-Subscriptor

Código A.8: Nodo Subscriptor - Publicador Robot *falcon*

```
#!/usr/bin/env python
# coding: utf-8
import rospy
from ik import *
import tkMessageBox
from Tkinter import *
from dynamixel_sdk import *
from ros_falcon.msg import *

global motor1,motor2, motor3, X, Y, Z, ri, FX, FY, FZ

def callback_function(data):
    global motor1,motor2, motor3, X, Y, Z, ri, FX, FY, FZ
    X = -100+(data.X-(-52))*1.8518
    Y = -150+(data.Y-(-53))*3.173
    Z = -400 + (data.Z-(50))*-2.1
    FX = data.X
    FY = data.Y
    FZ = data.Z
```

```
    posicion = falconPos()
    ri = inverse(X, Y, Z)
    motor1 = int(round(3800 - 10.8888*ri[0]))
    motor2 = int(round(3050 - 11.8888*ri[1]))
    motor3 = int(round(2350 - 11.4444*ri[2]))
    posicion.X = motor1
    posicion.Y = motor2
    posicion.Z = motor3
    pub.publish(posicion)
    print motor1, motor2, motor3

rospy.init_node('tele_remota', anonymous=True)
rospy.Subscriber("/falconPos", falconPos, callback_function)
pub = rospy.Publisher('/ir_pos', falconPos, queue_size= 10)

vp = Tk()
vp.title("Teleoperación Remota")
vp.geometry("800x480+0+0")
vp.config(relief="sunken", bd=10)
imagen = PhotoImage(file="~/catkin_ws/src/proyecto_delta/src/esc.png")
fondo = Label(vp, image=imagen).place(x=580,y=0)
#vp.resizable(0, 0)
# Configuración de Frame1
frame1 = Frame(vp, width = 200, height = 250 )
frame1.pack(side = "left", anchor = "n")
frame1.config(bg="#f9a994", cursor="hand2")

frame2 = Frame(vp, width = 200, height = 250 )
frame2.pack(side = "left", anchor = "n")
frame2.config(bg="#80f669", cursor="hand2")

frame3 = Frame(vp, width = 200, height = 250 )
frame3.pack(side = "left", anchor = "n")
frame3.config(bg="#acb8ef", cursor="hand2")

tc4 = Label(frame1, bg = "#f9a994")
tc4.grid(row = 0, column = 0, padx = 2, pady = 1)
tc58 = Label(frame1, bg = "#f9a994")
tc58.grid(row = 8, column = 0, padx = 2, pady = 1)
tc59 = Label(frame2, bg = "#80f669")
tc59.grid(row = 8, column = 0, padx = 2, pady = 1)


tc3 = Label(frame1, text= "Falcon", bg = "#f9a994", font=("Comic Sans MS", 18))
tc3.grid(row = 1, column = 0, padx = 2, pady = 1)

tc1 = Label(frame2, bg = "#80f669")
tc1.grid(row = 0, column = 2, padx = 2, pady = 1)

tc2 = Label(frame2, text= "Motores", bg = "#80f669", font=("Comic Sans MS", 18))
tc2.grid(row = 1, column = 2, padx = 2, pady = 1)

tc5 = Label(frame3, bg = "#acb8ef")
tc5.grid(row = 0, column = 2, padx = 2, pady = 1)

tc6 = Label(frame3, text= "RPD", bg = "#acb8ef", font=("Comic Sans MS", 18))
tc6.grid(row = 1, column = 2, padx = 2, pady = 1)

# Falcon
te1 = Label(frame1, text = "Px", bg = "#f9a994", font=("Comic Sans MS", 10))
te1.grid(row = 2, column = 0, sticky = "w", pady = 0, padx = 20)
te1.config(cursor = "pirate")
td1 = Label(frame1, bg = "#ffffff", width = 20 )
td1.grid(row = 3, column = 0, sticky = "e", pady = 1, padx = 20)
```

```
td1.config(fg = "blue", justify = "center")

te1r = Label(frame2, text = "Motor 1", bg = "#80f669", font=("Comic Sans MS", 10))
te1r.grid(row = 2, column = 1, sticky = "w", pady = 0, padx = 20)
te1r.config(cursor = "pirate")
td1r = Label(frame2, bg = "#ffffff", width = 10 )
td1r.grid(row = 3, column = 1, sticky = "w", pady = 1, padx = 20)
td1r.config(fg = "blue", justify = "center")

te1r2 = Label(frame2, text = "theta1", bg = "#80f669", font=("Comic Sans MS", 10))
te1r2.grid(row = 2, column = 2, sticky = "w", pady = 0, padx = 20)
te1r2.config(cursor = "pirate")
td1r2 = Label(frame2, bg = "#ffffff", width = 10 )
td1r2.grid(row = 3, column = 2, sticky = "w", pady = 1, padx = 20)
td1r2.config(fg = "blue", justify = "center")

te1r2d = Label(frame3, text = "Px", bg = "#acb8ef", font=("Comic Sans MS", 10))
te1r2d.grid(row = 2, column = 2, sticky = "w", pady = 0, padx = 20)
te1r2d.config(cursor = "pirate")
td1r2d = Label(frame3, bg = "#ffffff", width = 10 )
td1r2d.grid(row = 3, column = 2, sticky = "w", pady = 1, padx = 20)
td1r2d.config(fg = "blue", justify = "center")

te2 = Label(frame1, text = "Py", bg = "#f9a994", font=("Comic Sans MS", 10))
te2.grid(row = 4, column = 0, sticky = "w", pady = 0, padx = 20)
te2.config(cursor = "pirate")
td2 = Label(frame1, bg = "#ffffff", width = 20 )
td2.grid(row = 5, column = 0, sticky = "e", pady = 1, padx = 20)
td2.config(fg = "blue", justify = "center")

te2r = Label(frame2, text = "Motor 2", bg = "#80f669", font=("Comic Sans MS", 10))
te2r.grid(row = 4, column = 1, sticky = "w", pady = 0, padx = 20)
te2r.config(cursor = "pirate")
td2r = Label(frame2, bg = "#ffffff", width = 10 )
td2r.grid(row = 5, column = 1, sticky = "w", pady = 1, padx = 20)
td2r.config(fg = "blue", justify = "center")

te2r2 = Label(frame2, text = "theta2", bg = "#80f669", font=("Comic Sans MS", 10))
te2r2.grid(row = 4, column = 2, sticky = "w", pady = 0, padx = 20)
te2r2.config(cursor = "pirate")
td2r2 = Label(frame2, bg = "#ffffff", width = 10 )
td2r2.grid(row = 5, column = 2, sticky = "w", pady = 1, padx = 20)
td2r2.config(fg = "blue", justify = "center")

te2r2d = Label(frame3, text = "Py", bg = "#acb8ef", font=("Comic Sans MS", 10))
te2r2d.grid(row = 4, column = 2, sticky = "w", pady = 0, padx = 20)
te2r2d.config(cursor = "pirate")
td2r2d = Label(frame3, bg = "#ffffff", width = 10 )
td2r2d.grid(row = 5, column = 2, sticky = "w", pady = 1, padx = 20)
td2r2d.config(fg = "blue", justify = "center")

te3 = Label(frame1, text = "Pz", bg = "#f9a994", font=("Comic Sans MS", 10))
te3.grid(row = 6, column = 0, sticky = "w", pady = 0, padx = 20)
te3.config(cursor = "pirate")
td3 = Label(frame1, bg = "#ffffff", width = 20 )
td3.grid(row = 7, column = 0, sticky = "e", pady = 1, padx = 20)
td3.config(fg = "blue", justify = "center")

tc58 = Label(frame1, bg = "#f9a994")
tc58.grid(row = 9, column = 0, padx = 2, pady = 1)

te3r = Label(frame2, text = "Motor 3", bg = "#80f669", font=("Comic Sans MS", 10))
te3r.grid(row = 6, column = 1, sticky = "w", pady = 0, padx = 20)
te3r.config(cursor = "pirate")
```

```
td3r = Label(frame2, bg = "#fffff", width = 10 )
td3r.grid(row = 7, column = 1, sticky = "w", pady = 1, padx = 20)
td3r.config(fg = "blue", justify = "center")

te3r3 = Label(frame2, text = "theta3", bg = "#80f669", font=("Comic Sans MS", 10))
te3r3.grid(row = 6, column = 2, sticky = "w", pady = 0, padx = 20)
te3r3.config(cursor = "pirate")
td3r3 = Label(frame2, bg = "#fffff", width = 10 )
td3r3.grid(row = 7, column = 2, sticky = "w", pady = 1, padx = 20)
td3r3.config(fg = "blue", justify = "center")

tc59 = Label(frame2, bg = "#80f669")
tc59.grid(row = 9, column = 0, padx = 2, pady = 1)

te3r2d = Label(frame3, text = "Pz", bg = "#acb8ef", font=("Comic Sans MS", 10))
te3r2d.grid(row = 6, column = 2, sticky = "w", pady = 0, padx = 20)
te3r2d.config(cursor = "pirate")
td3r2d = Label(frame3, bg = "#fffff", width = 10 )
td3r2d.grid(row = 7, column = 2, sticky = "w", pady = 1, padx = 20)
td3r2d.config(fg = "blue", justify = "center")

def codigoboton1():
        s = tkMessageBox.askyesno(message="¿Desea continuar?", title="Título")
        if s == False:
        vp.destroy()

boton1= Button(frame3, text = "Cerrar", command = codigoboton1, bg = "#f92103")
boton1.grid(row = 8, column = 2, pady = 8, padx = 4)


while not rospy.is_shutdown():
    try:
                td1r["text"] = motor1
                td2r["text"] = motor2
                td3r["text"] = motor3
                td1["text"] = FX
                td2["text"] = FY
                td3["text"] = FZ
                td1r2d["text"] = int(X)
                td2r2d["text"] = int(Y)
                td3r2d["text"] = int(Z)
                td1r2["text"] = int(ri[0])
                td2r2["text"] = int(ri[1])
                td3r3["text"] = int(ri[2])
                vp.update()
        vp.update_idletasks()
            #rate.sleep()
    except:
        break
rospy.spin()
```

A.4.7. Teleoperación Remota - Nodo Subscriptor

Código A.9: Nodo Subscriptor

```
#!/usr/bin/env python
import rospy
from ros_falcon.msg import *
from dxl.dxlchain import DxlChain

global motores, c
c = 0
motores = DxlChain("/dev/ttyUSB0", rate = 1000000)
motors = motores.get_motor_list()
#print motors

def callback_function(data):
    global c
    data1 = int(data.X)
    data2 = int(data.Y)
    data3 = int(data.Z)
    print c
    c = c + 1
    if c>40:
        motores.goto(1, data.X, speed = 40, blocking = False)
        motores.goto(2, data.Y, speed = 40, blocking = False)
        motores.goto(3, data.Z, speed = 40, blocking = False)
        c = 0
    print data.X, data.Y, data.Z, data1, data2, data3

def publisher():
    rospy.init_node('subs_falcon', anonymous=True)
    rospy.Subscriber("/ir_pos", falconPos, callback_function)
    rospy.spin()

if __name__ == '__main__':
    publisher()
```