Neiva, 08/05/2023

Señores

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD SURCOLOMBIANA

Ciudad Neiva

El (Los) suscrito(s):

Andrés Felipe Vega Tique, con C.C. No.1003815365,

Autor(es) de la tesis y/o trabajo de grado o Pasantía

Titulado "DESARROLLO DE MÓDULO DIDÁCTICO DE INTEGRACIÓN DE ROS CON ROBOT OPENBOTV V1 PARA FINES ACADÉMICOS E INVESTIGACIÓN" presentado y aprobado en el año 2023 como requisito para optar al título de Ingeniero Electrónico;

Autorizo (amos) al CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN de la Universidad Surcolombiana para que, con fines académicos, muestre al país y el exterior la producción intelectual de la Universidad Surcolombiana, a través de la visibilidad de su contenido de la siguiente manera:

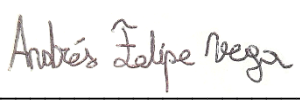
- Los usuarios puedan consultar el contenido de este trabajo de grado en los sitios web que administra la Universidad, en bases de datos, repositorio digital, catálogos y en otros sitios web, redes y sistemas de información nacionales e internacionales "open access" y en las redes de información con las cuales tenga convenio la Institución.

- Permita la consulta, la reproducción y préstamo a los usuarios interesados en el contenido de este trabajo, para todos los usos que tengan finalidad académica, ya sea en formato Cd-Rom o digital desde internet, intranet, etc., y en general para cualquier formato conocido o por conocer, dentro de los términos establecidos en la Ley 23 de 1982, Ley 44 de 1993, Decisión Andina 351 de 1993, Decreto 460 de 1995 y demás normas generales sobre la materia.

- Continúo conservando los correspondientes derechos sin modificación o restricción alguna; puesto que, de acuerdo con la legislación colombiana aplicable, el presente es un acuerdo jurídico que en ningún caso conlleva la enajenación del derecho de autor y sus conexos.

De conformidad con lo establecido en el artículo 30 de la Ley 23 de 1982 y el artículo 11 de la Decisión Andina 351 de 1993, "Los derechos morales sobre el trabajo son propiedad de los autores", los cuales son irrenunciables, imprescriptibles, inembargables e inalienables.

EL AUTOR/ESTUDIANTE:

Firma: 

TÍTULO COMPLETO DEL TRABAJO: "DESARROLLO DE MÓDULO DIDÁCTICO DE INTEGRACIÓN DE ROS CON ROBOT OPENBOTV V1 PARA FINES ACADÉMICOS E INVESTIGACIÓN".

AUTOR O AUTORES:

Primero y Segundo Apellido	Primero y Segundo Nombre
Vega Tique	Andrés Felipe

DIRECTOR Y CODIRECTOR TESIS:

Primero y Segundo Apellido	Primero y Segundo Nombre
Sendoya Losada	Diego Fernando

ASESOR (ES):

Primero y Segundo Apellido	Primero y Segundo Nombre
X	X

PARA OPTAR AL TÍTULO DE: Ingeniero Electrónico

FACULTAD: Ingeniería

PROGRAMA O POSGRADO: Electrónica

CIUDAD: Neiva **AÑO DE PRESENTACIÓN:2023** **NÚMERO DE PÁGINAS: 75**

TIPO DE ILUSTRACIONES (Marcar con una **X**):

Diagramas___ Fotografías___ Grabaciones en discos___ Ilustraciones en general__x_ Grabados___ Láminas___ Litografías___ Mapas___ Música impresa___ Planos___ Retratos___ Sin ilustraciones___ Tablas o Cuadros_x_

SOFTWARE requerido y/o especializado para la lectura del documento: Ninguno

MATERIAL ANEXO: Ninguno

PREMIO O DISTINCIÓN *(En caso de ser LAUREADAS o Meritoria)*: Ninguno

PALABRAS CLAVES EN ESPAÑOL E INGLÉS:

	Español	Inglés		Español	Inglés
1.	ROS	ROS	6.	Dynamixel	Dynamixel
2.	Cinemática	Kinematics	7.	Brazo	Arm
3.	Python	Python			
4.	Robótica	Robotics			
5.	OpenBotV v1	OpenBotV v1			

RESUMEN DEL CONTENIDO: (Máximo 250 palabras)

En este trabajo de grado en modalidad de pasantía se presentan los resultados obtenidos del proyecto "Desarrollo de módulo didáctico de integración de ROS con robot OpenBotv v1 para fines académicos e investigación", el cual se plantea con el objetivo de desarrollar un módulo didáctico para la empresa Robotics 4.0 S.A.S, para que entusiastas en temáticas de robótica hispano hablantes cuenten con una mayor oferta en el mercado local y nacional, en lo que respecta a formas de instruirse en el manejo de softwares especializados en robótica como lo es ROS y robots académicos compatibles con el software como OpenBotv v1.

El pasante que llevó a cabo la construcción del módulo didáctico lo realizó en distintas fases. Inicialmente, se centró en realizar el proceso de programación, simulación y manejo del robot académico "OpenBotv v1" desde el software ROS. Posteriormente, teniendo un control total sobre el robot se aplicaron modelos cinemáticos mediante el uso del software y, por último, se documentó el proceso realizado en forma de guías y videos para formar el material que conformaría el módulo didáctico.

Actualmente, el material desarrollado por el pasante se encuentra publicado por Robotics 4.0 S.A.S en la plataforma de Udemy en forma del curso "Robótica Antropomórfica Básica en ROS".

ABSTRACT: (Máximo 250 palabras)

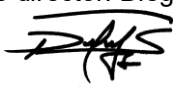
This degree work in internship form presents the results obtained from the project "Development of a didactic module for the integration of ROS with OpenBotv v1 robot for academic and research purposes", which aims to develop a didactic module for the company Robotics 4. 0 S.A.S., so that Spanish-speaking robotics enthusiasts have a greater offer in the local and national market on ways to be instructed in the management of specialized robotics software such as ROS and academic robots compatible with the software as OpenBotv v1.

The intern who built the didactic module did it in different phases. Initially, he focused on the process of programming, simulation and management of the academic robot "OpenBotv v1" using ROS software. Subsequently, having full control over the robot, kinematic models were applied using the software and, finally, the process was documented in the form of guides and videos to form the material that would conform the didactic module.

Currently, the material developed by the intern is published by Robotics 4.0 S.A.S. on the Udemy platform in the form of the course "Basic Anthropomorphic Robotics in ROS".

APROBACION DE LA TESIS

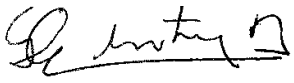
Nombre director: Diego Fernando Sendoya Losada

Firma: 

Nombre Jurado: José de Jesús Salgado Patrón

Firma: 

Nombre Jurado: German Eduardo Martínez Barreto

Firma: 

DESARROLLO DE MÓDULO DIDÁCTICO DE INTEGRACIÓN DE ROS CON ROBOT OPENBOTV V1 PARA FINES ACADÉMICOS E INVESTIGACIÓN

ANDRÉS FELIPE VEGA TIQUE
CÓDIGO: 20181164490

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
NEIVA – HUILA
2023

DESARROLLO DE MÓDULO DIDÁCTICO DE INTEGRACIÓN DE ROS CON ROBOT OPENBOTV V1 PARA FINES ACADÉMICOS E INVESTIGACIÓN

Andrés Felipe Vega Tique
Código: 20181164490

Línea de la propuesta:
Robótica

Director:
Ing. Diego Fernando Sendoya Losada

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
NEIVA – HUILA
2023

AGRADECIMIENTOS

Agradezco a mis padres Robert Vega y Norha Tique por permitir formarme como ingeniero electrónico en la universidad Surcolombiana y por el apoyo incondicional, tanto emocional y financiero, que me brindaron durante mi proceso de aprendizaje.

Agradezco al ingeniero Mario Arbulú, por la gran oportunidad que me brindó al realizar un proceso de pasantía en la empresa Robotics 4.0 S.A.S. Debido a que no existen ofertas similares en la región que permitan ganar experiencia y familiarizarse con el área de la robótica.

Agradezco a mis docentes por guiarme y los conocimientos compartidos durante mi trayectoria universitaria, en especial, al ingeniero Diego Sendoya por asesorarme en la construcción de este trabajo de grado en modalidad de pasantía.

TABLA DE CONTENIDO

pág.

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

En este trabajo de grado en modalidad de pasantía se presentan los resultados obtenidos del proyecto “Desarrollo de módulo didáctico de integración de ROS con robot OpenBotv v1 para fines académicos e investigación”, el cual se plantea con el objetivo de desarrollar un módulo didáctico para la empresa Robotics 4.0 S.A.S, para que entusiastas en temáticas de robótica hispano hablantes cuenten con una mayor oferta en el mercado local y nacional, en lo que respecta a formas de instruirse en el manejo de softwares especializados en robótica como lo es ROS y robots académicos compatibles con el software como OpenBotv v1.

El pasante que llevó a cabo la construcción del módulo didáctico lo realizó en distintas fases. Inicialmente, se centró en realizar el proceso de programación, simulación y manejo del robot académico “OpenBotv v1” desde el software ROS. Posteriormente, teniendo un control total sobre el robot se aplicaron modelos cinemáticos mediante el uso del software y, por último, se documentó el proceso realizado en forma de guías y videos para formar el material que conformaría el módulo didáctico.

Actualmente, el material desarrollado por el pasante se encuentra publicado por Robotics 4.0 S.A.S en la plataforma de Udemy en forma del curso “Robótica Antropomórfica Básica en ROS”[1].

[1] Robotics 4.0. (16 de febrero de 2023). Udemy. Obtenido de Robótica Antropomórfica Básica en ROS: https://www.udemy.com/course/robotica-antropomorfica-basica-en-ros/?src=sac&kw=robotica%2Bantropomorf

INTRODUCCIÓN

El campo de la robótica es una rama fascinante que combina tanto los campos de hardware y software para diseñar robots que han sido programados para facilitar la ejecución de una tarea específica. Un detalle que no debe omitirse es que entre más difícil o compleja sea la tarea a realizar del robot, más arduo es el proceso de codificación.

Esta es una de las razones por la cual la Universidad de Stanford en 2007 desarrolló el software conocido como Robot Operating System o comúnmente conocido como ROS (Quigley, Gerkey, & Smart, 2015, p.24). La literatura disponible lo define como "Un entorno de trabajo flexible, con una amplia variedad de herramientas, librerías y paquetes que busca la creación de un software complejo para tener robots robustos y con un comportamiento variado". Igualmente, posee la ventaja de ser un software con libertad para uso comercial e investigación, con el apoyo de una comunidad fuerte.

La principal desventaja de ROS, respecto a entusiastas en temáticas de robótica hispano hablantes, es que en el mercado local y nacional existe poca oferta de oportunidades que permitan instruirse fácilmente en el manejo de esta herramienta. ROBOTICS 4.0 S.A.S una empresa enfocada en diseñar, transferir e integrar tecnología robótica en el ámbito académico, servicios e industria; en conjunto con su robot académico OpenBotv v1 se propone a avanzar en la integración de éste con el software ROS y de manera paralela crear un módulo didáctico que genere un impacto social y académico para la comunidad interesada en temáticas de robótica.

Basado en lo anterior, en este trabajo de grado en modalidad de pasantía se describe el proceso que llevó a cabo el pasante para realizar la programación y manejo del robot académico "OpenBotv v1" desde el software ROS, la implementación de modelos cinemáticos mediante el uso de esta herramienta y la posterior condensación de lo aprendido en un módulo didáctico.

1. PLANTEAMIENTO DEL PROBLEMA

ROBOTICS 4.0 S.A.S es una empresa que ofrece soluciones robotizadas en el marco de los estándares de la cuarta revolución industrial para el área de servicios e industria. De manera paralela la empresa desde sus inicios ha intentado promover el aprendizaje de la rama de la robótica en la región a través de módulos didácticos y cursos en este campo para distintos niveles académicos.

A pesar de haber interactuado con el software ROS (Robot Operating System) en el pasado, la empresa dentro sus contenidos académicos desarrollados y publicados no posee nada referente al uso de esta herramienta. Con la iniciativa de suplementar esta carencia, Robotics 4.0 S.A.S se propone a profundizar en el uso de este software para poder acoplarlo a su robot académico “OpenBotv v1” y aplicar modelos cinemáticos programados desde éste. Adicionalmente, con el objetivo de plasmar todo el procedimiento en un módulo didáctico.

Para la anterior tarea se requiere de un recurso humano, cuyo perfil incluye un estudiante de ingeniería electrónica con experiencia en programación en Python, dispuesto a adquirir conocimientos básicos de robótica y el software ROS. Además, que vea en este campo una oportunidad futura de desarrollo profesional.

2. JUSTIFICACIÓN

Como se ha mencionado, Robot Operating System es una herramienta usada a nivel industrial por distintas empresas en el área de la robótica por las distintas ventajas que ofrece a la hora de desarrollar proyectos. Tal es su popularidad que según ABI Research[2], en su informe de "Proyectos de robótica de código abierto", se espera que casi el 55% de los robots de todo el mundo incluyan al menos un paquete de ROS en 2024. Convirtiendo el uso de esta herramienta en el lenguaje común de automatización para proyectos de robótica.

Por tal razón, la realización de esta propuesta no solo representa para el pasante una gran oportunidad de aprendizaje, permitiéndole familiarizarse con herramientas (software y hardware) relacionadas a la programación y manejo de robots como lo son ROS y el robot académico OpenBotv v1, sino también, que la construcción de un contenido académico respecto al proceso que realice el pasante, representa un gran impacto para la región para aquellas personas que vean un futuro desarrollo profesional en el campo de la robótica, debido a que contarían con un material de calidad, escrito en su lengua materna (Español), que les permitiera introducirse en un elemento tan potente en el área de la robótica como lo es ROS.

[2] ABI Research. (19 de febrero de 2023). Obtenido de The rise of ROS: Nearly 55% of total commercial robots shipped in 2024 will have at least one robot operating system package installed: https://www.abiresearch.com/press/rise-ros-nearly-55-total-commercial-robots-shipped-2024-will-have-least-one-robot-operating-system-package-installed/

3. ANTECEDENTES

ROBOTICS 4.0 S.A.S es una empresa fundada a finales de 2018 que se dedica a desarrollar soluciones robotizadas a medida de la cuarta revolución industrial para el sector de servicios e industria. De manera paralela la empresa desde sus inicios ha intentado promover el aprendizaje de la rama de Robótica en la región a través de módulos didácticos y cursos en este campo para distintos niveles académicos.

Todos los módulos académicos que maneja la empresa se complementan con el uso del robot OpenBotv v1 y sus derivados.

Estos robots se caracterizan por ser reconfigurables y permitirle al estudiante tener la posibilidad de diseñar y probar distintas configuraciones o tareas. Además, como su nombre lo indica le brinda la facilidad al usuario de controlarse y programarse en distintos entornos como lo son Matlab, Labview, C++, Java, Pythony ROS. Esta versatilidad que ofrece al ser una plataforma abierta es lo que motiva también a usarse en distintas aplicaciones de Robótica industrial.

Desde la fundación de la empresa se pueden resaltar los siguientes proyectos[3]:

- Teleoperación inalámbrica mio-eléctrica inteligente (2018): Proyecto desarrollado en colaboración con estudiantes de la Escuela Colombiana de ingeniería 'Julio Garavito'.

- Robot para localización Estereotáxica (2018-2020): PMV para FACOSEME SAC cofinanciado por INNOVATE / Perú.

- Robot para limpieza de paneles solares (2020-2021): PMV implementado para la empresa SUNNYAPP SAS, cofinanciado por Minciencias y Tecnnova.

[3] Robotics 4.0. (16 de febrero de 2023). Youtube. Obtenido de HuilaFEST 4.0: https://youtu.be/iPjFRZ1tPg0

4. OBJETIVOS

4.1 OBJETIVO GENERAL

Desarrollar para ROBOTICS 4.0 S.A.S un módulo didáctico sobre la integración de ROS con el robot OpenBotv v1, para interactuar con hardware real y mediante modelos de simulación programados en ROS.

4.2 OBJETIVOS ESPECÍFICOS

- Aprender el funcionamiento de la arquitectura ROS para implementar algoritmos de modelado cinemático en robot OpenBotv v1 mediante el estudio de tutoriales y aplicación de simulaciones desarrolladas por la empresa.

- Implementar e integrar modelos cinemáticos desarrollados en el robot OpenBotv v1 a través de ROS.

- Validar la implementación de los modelos cinemáticos mediante simulaciones en ROS y su desempeño en el robot real.

- Desarrollar material académico (guías de trabajo, videos, etc.) para explicar paso a paso la integración de ROS con OpenBotv v1 y la aplicación de los modelos cinemáticos.

5. MARCO TEÓRICO

5.1 ¿QUÉ ES ROS?

Robot Operating System, comúnmente llamado como ROS, es un framework de robótica de carácter gratuito y código abierto con posibilidad de uso para fines comerciales e investigación. Actualmente, se le considera la plataforma por defecto para el desarrollo de aplicaciones robotizadas debido a las ventajas que ofrece a sus usuarios, como:

- Interfaz de paso de mensajes entre procesos (Nodos).
- Funcionalidades similares a las de un sistema operativo (Workspace, Packages).
- Soporte a lenguajes de programación de alto nivel (Python, C++)
- Herramientas de simulación (Gazebo, Rviz)
- Soporte por parte de la comunidad. (Lentin y Aleena, 2018, p.132)

Figura 1. Logotipo de Robot Operating System

Fuente: Open Robotics. (16 de febrero de 2023). ROS: Home. Obtenido de https://www.ros.org

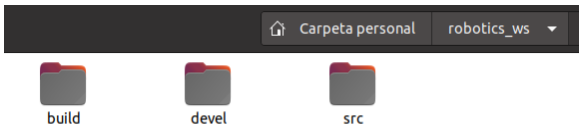
5.2 ¿CÓMO FUNCIONA ROS?

5.2.1 Workspace

Es una carpeta donde se introducen distintos paquetes para ejecutar una aplicación (Simulaciones, códigos de Python, etc.).

La estructura de un Workspace está conformada por una carpeta src, devel y build.

Figura 2. Estructura de espacio de trabajo en ROS

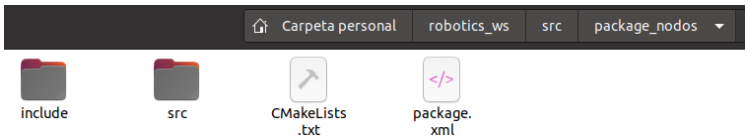


Fuente: Propia

5.2.2 Paquete

Es una subcarpeta que se diseña para que cumpla una funcionalidad especifica al ser llamada por ROS. Posee una carpeta src donde se colocan los archivos a emplear durante la ejecución y dos archivos de configuración (CmakeLists.txt y package.xml) para definir parámetros del paquete.

Figura 3. Estructura de un paquete en ROS

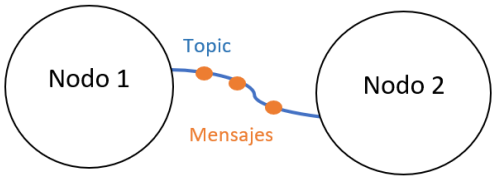


Fuente: Propia

5.2.3 Nodos

Según Fairchild & Harman (2017), básicamente son programas que realizan algún cálculo o tarea específica. Cada nodo está pensado para poder operar independientemente ejecutando un script, pero poseen la capacidad de comunicarse entre ellos mediante la estructura de comunicación de ROS.

Figura 4. Forma de comunicación entre nodos



Fuente: Propia

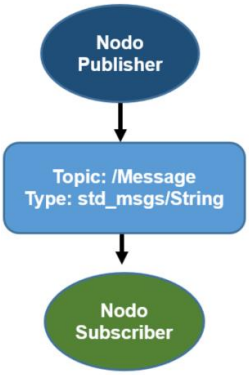
A los canales de comunicación que existen entre los nodos se les denomina Topics y la manera como se envía información es por medio de unas estructuras conocidas como mensajes.

Para el caso específico de Python, Rospy es la librería para la configuración e inicialización de un nodo mediante un script (Rospy - ROS Wiki, 2017).

5.2.4 Nodo Publisher y Subscriber

Se define como una relación emisor-receptor que existe entre los nodos para el intercambio de información. Es una de las maneras de intercambio de datos que ofrece ROS.

Figura 5. Comunicación entre nodo Publisher y Subscriber

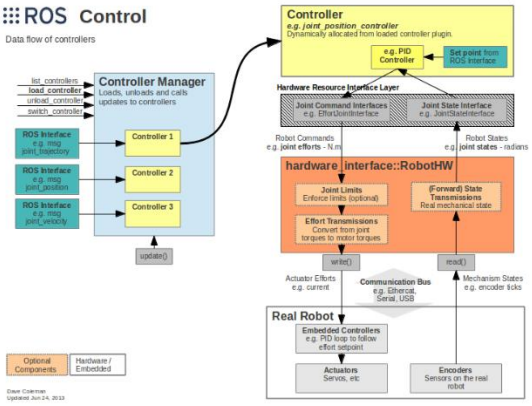


Fuente: Propia

5.2.5 ROS control

Es una librería de ROS que permite establecer controladores (generalmente PID) con distintos parámetros de entrada para el control de la articulación de un robot. Su implementación pide la configuración del propio controlador en un archivo .yaml y añadir unas estructuras conocidas como transmisores al archivo Xacro o URDF donde se encuentra descrito el modelo del robot. (ROS control,2023).

Figura 6. Diagrama de funcionamiento ROS Control



Fuente: Apache 2.0. (16 de febrero de 2023). Gazebo : Tutorial : ROS control. Obtenido de https://classic.gazebosim.org/tutorials?tut=ros_control#Aboutros_control

5.2.6 Herramientas de simulación de ROS

Gazebo: Es un entorno de simulación de robots gratuito y de código abierto desarrollado por Willow Garage. Como herramienta multifuncional para desarrolladores de robots en ROS, Gazebo soporta lo siguiente:

- Diseño de modelos de robots
- Creación rápida de prototipos y pruebas de algoritmos
- Simulación de ambientes (Interiores y exteriores)
- Simulación de datos de sensores
- Motores de física de alto rendimiento como: Object-Oriented Graphics Rendering Engine (OGRE), Open Dynamics Engine (ODE), Bullet, Simbody y Dynamic Animation and Robotics Toolkit (DART).

(Fairchild & Harman, 2017, p.60)

Figura 7. Logotipo de software de simulación Gazebo

Fuente: Apache 2.0. (16 de febrero de 2023). Gazebo : Media. Obtenido de https://classic.gazebosim.org/media

RVIZ: Son las siglas para ROS Visualization. Es un entorno 3D generalmente empleado para la visualización de robots, sensores y algoritmos. Como la mayoría de las herramientas de ROS, se puede utilizar para cualquier robot y rápidamente configurarlo para una aplicación particular (Quigley, Gerkey, & Smart, 2015, p.126).

Figura 8. Logotipo de software de simulación RVIZ

Fuente: ros visualization . GitHub. (16 de febrero de 2023). Obtenido de https://github.com/ros-visualization

La principal diferencia con Gazebo, es la manera como cada programa realiza su proceso de simulación. “RVIZ muestra lo que el robot piensa que está pasando mientras que Gazebo muestra lo que de verdad está pasando” (Quigley, Gerkey, & Smart, 2015, p.300).

5.2.7 Componentes de una Simulación de ROS

URDF: Son las siglas de (Unified Robot Description Format). Corresponde a un formato de lenguaje utilizado para describir robots empleando la gramática XML. A través de un formato URDF se puede modelar la estructura de un robot, sus dimensiones, masa, articulaciones, actuadores, etc. (Lentin y Aleena, 2018, p.132).

Figura 9. Ejemplo de estructura de archivo URDF

```
<?xml version="1.0"?>
<robot name="myfirst">
  <link name="base_link">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
    </visual>
  </link>
</robot>
```

Fuente: Model with URDF from Scratch: http://wiki.ros.org/urdf/Tutorials/Building%20a%20Visual%20Robot%20Model%20with%20URDF%20from%20Scratch

Xacro: Para Fairchild & Harman (2017, p.42), Xacro es un lenguaje de macros XML creado para hacer los archivos de descripción de robots URDF más fáciles de leer, reduciendo la duplicación de información dentro del archivo.

Figura 10. Estructura de archivo Xacro de OpenBotv v1

Abrir — openbot_v1.xacro — ~/demostracion_ws/src/openbot_v1_description/urdf

```
<?xml version="1.0" ?>
<robot name="openbot_v1" xmlns:xacro="http://www.ros.org/wiki/xacro">

<xacro:include filename="$(find openbot_v1_description)/urdf/materials.xacro" />
<xacro:include filename="$(find openbot_v1_description)/urdf/openbot_v1.trans" />
<xacro:include filename="$(find openbot_v1_description)/urdf/openbot_v1.gazebo" />

<link name="world"/>
<joint name="world_joint" type="fixed">
  <parent link="world"/>
  <child link="base_link"/>
</joint>

<link name="base_link">
  <inertial>
    <origin rpy="0 0 0" xyz="0.0005943128623696434 -0.00075374611477290 55 0.01585867561565245"/>
    <mass value="8.934333178369254"/>
    <inertia ixx="0.063131" ixy="-2e-06" ixz="-0.000496" iyy="0.061573" iyz="0.000637" izz="0.111897"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://openbot_v1_description/meshes/visual/base_link.stl" scale="0.001 0.001 0.001"/>
    </geometry>
    <material name="silver"/>
  </visual>
  <collision>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://openbot_v1_description/meshes/collision/base_link_collision.stl" scale="0.0001 0.0001 0.0001"/>
    </geometry>
  </collision>
</link>
```

Fuente: Propia

Roslaunch: Es una herramienta para arrancar fácilmente varios nodos de ROS e inicializar parámetros. Los archivos de configuración de Roslaunch se escriben en XML y utilizan la extensión .launch. (Fairchild & Harman,2017, p.44).

Figura 11. Ejemplo de estructura de archivo .launch

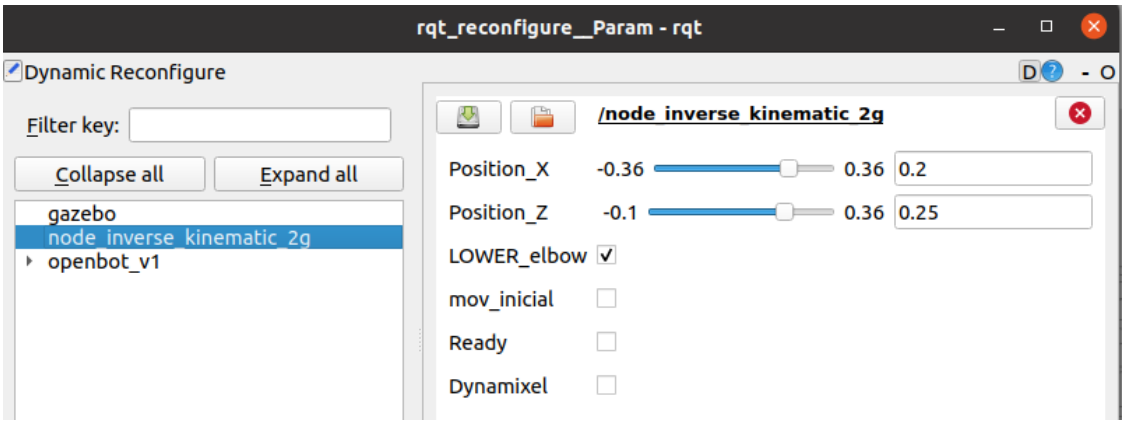
```
<launch>
        <include file="$(find openbot_gazebo)/launch/gazebo.launch"/>
        <include file="$(find openbot_control)/launch/controller.launch"/>
        <!--Launch the inverse kinematic -->
        <node pkg="modelos_cinematicos"
         type="Demo_cinematica_Inversa_2G.py"
         name="node_inverse_kinematic_2g"
         output="screen"
         >
        </node>
        <!--RQT_reconfigure for the dynamic configure-->
        <node name="rqt_reconfigure" pkg="rqt_reconfigure" type="rqt_reconfigure"/>
        <!--TF frame to know the Position of the effector-->
        <node pkg="tf2_ros" type="static_transform_publisher" name="effector_link_tf" args="0 0 0.15 0 0 0 1 clamp_arm_link_1 effector_link_tf" />

</launch>
```

Fuente: Propia

Rqt-configure y archivos .cfg: Las herramientas rqt que forman parte del framework permiten representaciones gráficas de nodos, temas, mensajes y otro tipo de información (rqt/Plugins - ROS Wiki). La wiki de ROS enumera muchas de las posibles herramientas que se pueden usar como complementos, pero la usada durante el proyecto fue rqt_reconfigure

Esta herramienta permite a nodos que hayan sido programados usando la rqt_reconfigure API ser visibles dentro la interfaz y que dentro del GUI aparezcan sus parámetros con los valores actuales y límites. (Fairchild & Harman,2017, p.198).

Figura 12. Ejemplo de interfaz empleando rqt_configure



Fuente: Propia

Estos parámetros se definen empleando un archivo .cfg, en el cual se crean las variables que se quiere que aparezcan en la interfaz, su rango de trabajo y valor inicial.

Figura 13. Ejemplo de estructura de archivo .cfg

```
#!/usr/bin/env python
PACKAGE = "modelos_cinematicos"

from dynamic_reconfigure.parameter_generator_catkin import *

gen = ParameterGenerator()
gen.add("Position_X",    double_t,    0, "Valor de X en el plano", 0.2,  -0.36, 0.36)
gen.add("Position_Z",    double_t,    0, "Valor de Z en el plano", 0.25,  -0.1,0.36)
gen.add("LOWER_elbow", bool_t , 0, "Orientaciòn deseada de codo",True)
gen.add("mov_inicial", bool_t , 0, "Marcar para enviar a la posición inicial",False)
gen.add("Ready", bool_t , 0, "Marcar para habilitar movimiento en simulaciòn",False)
gen.add("Dynamixel", bool_t , 0, "Marcar para habilitar movimiento robot real",False)

exit(gen.generate(PACKAGE, "InverseKinematics_2dof", "IK_2G"))
```

Fuente: Propia

Cmakelist: Este archivo contiene todos los comandos para construir el código fuente de ROS dentro del paquete y crear el ejecutable. (Lentin y Aleena, 2018, p.179).

Package.xml: Es un archivo XML que contiene principalmente las dependencias del paquete, información, etc. (Lentin y Aleena, 2018, p.180).

5.3 OPENBOTV V1

5.3.1 Definición

Es un robot bioinspirado, didáctico y reconfigurable con 6 grados de libertad, especializado en resolver problemas de desplazamiento de objetos y movimiento espacial. Mediante un accesorio de intercambio de datos, es capaz de controlarse y programarse desde un PC, mediante distintos entornos como Matlab, LabView, C++, Java, Python, ROS. (E-Robotics 4.0, 2023).

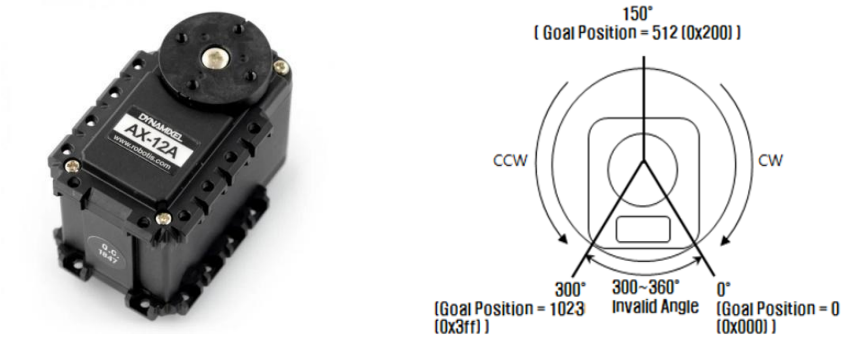
Figura 14. Robot académico OpenBotv v1

Fuente: Robotics 4.0. (15 de Marzo de 2023). Robotics 4.0. Obtenido de E-Robotics 4.0 I Robotics 4.0: https://robotics40.com/wp-content/uploads/2019/04/OpenBotvv1.jpg

5.3.2 Motores

El OpenBotv v1 está compuesto por 6 motores de la marca Dynamixel serie AX-12. Los servomotores de esta referencia operan en un rango de trabajo de 0 a 1023 posiciones. (AX-12A, 2023).

Figura 15. Rango de posiciones de motor Dynamixel AX-12A



Fuente: ROBOTIS. (15 de febrero de 2023). ROBOTIS e-Manual. Obtenido de AX-12A: https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/

Todos los motores de la marca Dynamixel son manipulables desde distintos lenguajes de programación y para el caso específico de ROS poseen una librería denominada Dynamixel_Workbench para hacer más fácil el acople con el software. (DYNAMIXEL Workbench, 2023).

5.4 ELEMENTOS USADOS EN EL MONTAJE

5.4.1 Convertidor U2D2

“Es un convertidor de comunicación USB de pequeño tamaño que permite controlar y operar DYNAMIXEL desde un PC” (Robotis e-Manual U2D2,2023).

Figura 16. Convertidor U2D2

Fuente: ROBOTIS. (13 de febrero de 2023). ROBOTIS e-Manual. Obtenido de DYNAMIXEL Workbench: https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_workbench/

5.4.2 SMPS2Dynamixel

Este dispositivo proporciona energía a un Dynamixel desde un SMPS. Posee conectores de 3 pines para la serie AX y conectores de 4 pines para la serie DX/RX/EX. Las líneas de alimentación y comunicación están conectadas, lo que permite desempeñar el papel de bus de expansión del Dynamixel. (Robotis SMPS2Dynamixel, 2023).

Figura 17. SMPS2 Dynamixel

Fuente: ROBOTIS. (15 de febrero de 2023). ROBOTIS. Obtenido de SMPS2Dynamixel - ROBOTIS: https://www.robotis.us/smps2dynamixel/

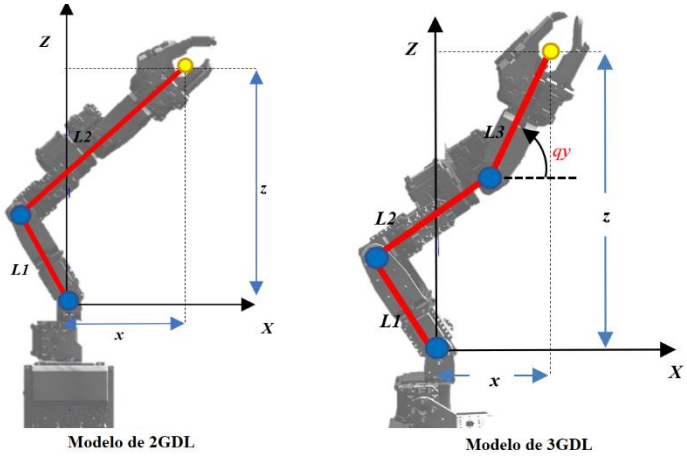
5.5 DESARROLLO TEÓRICO DE LOS MODELOS CINEMÁTICOS

5.5.1 Aclaraciones

La construcción de los modelos cinemáticos que se aplican sobre el OpenBotv v1 en conjunto con ROS, se implementan según los siguientes aspectos:

- El modelo se define para un movimiento planar respecto al plano **XZ** considerando una configuración de 2 y 3 grados de libertad.

Figura 18. Diagrama de modelo cinemático de 2GDL y 3GDL

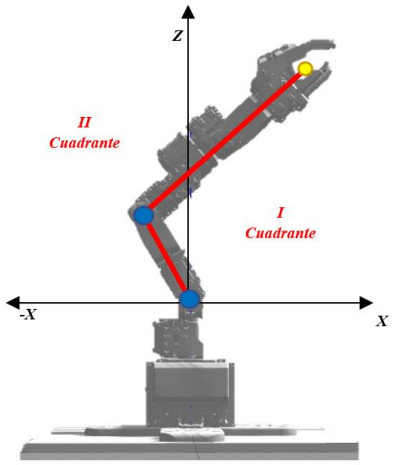


Fuente: Propia

La diferencia entre ambas configuraciones es como el modelo de 3 grados de libertad considera la orientación o ángulo del efector final (***qy***) al realizar el movimiento planar.

- El desarrollo teórico respecto al plano **XZ** se realiza exclusivamente para el primer y segundo cuadrante.

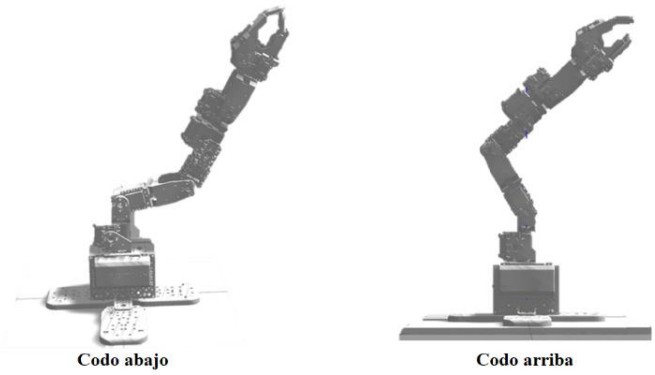
Figura 19. Cuadrantes del plano (x, z) empleados en el modelo



Fuente: Propia

- Adicionalmente, el análisis del modelo 2 y 3 grados de libertad se plantea respecto a dos orientaciones de codo (arriba y abajo) para representar el comportamiento de un brazo robótico.

Figura 20. Orientaciones de codo abajo y arriba



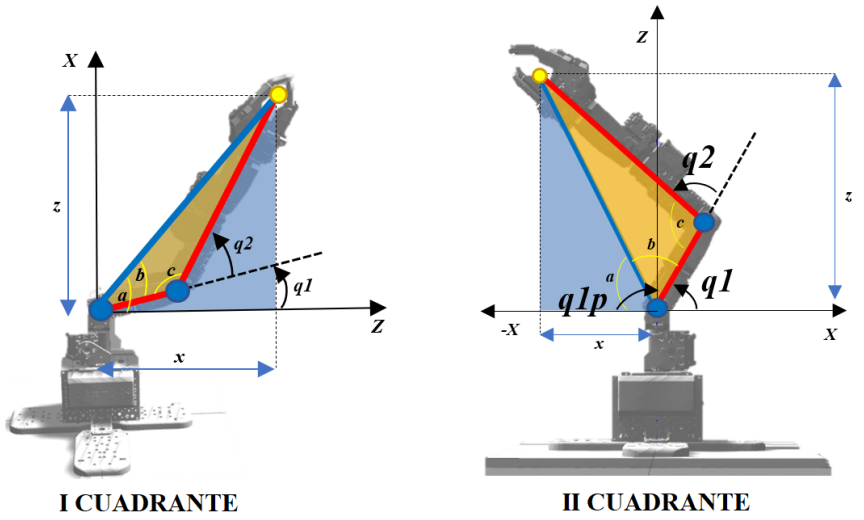
Fuente: Propia

5.5.2 Cinemática inversa

Es una técnica que permite calcular los giros de las articulaciones del robot para llevar su efector final a unas coordenadas específicas en el plano.

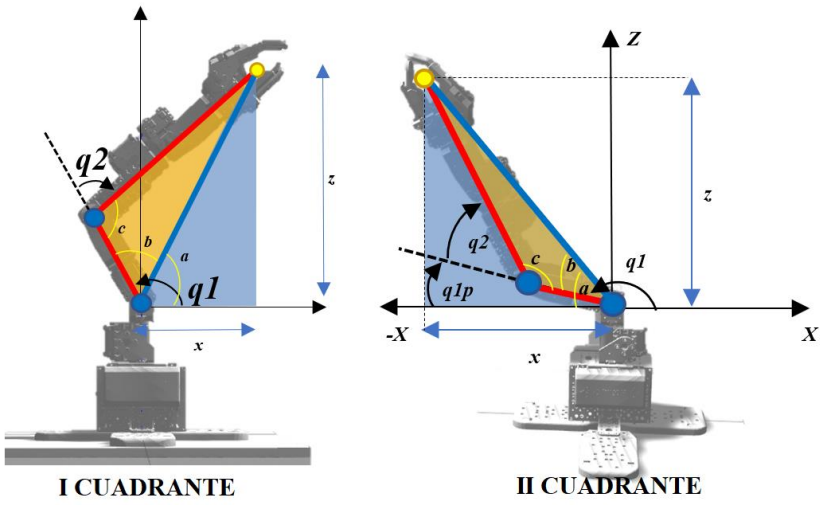
Para 2GDL, se planteó el modelo a través de un sistema de triángulos que permite calcular fácilmente los ángulos de interés ($\boldsymbol{q1}$, $\boldsymbol{q2}$) en ambas orientaciones de codo (arriba y abajo).

Figura 21. Diagrama de cinemática inversa 2GDL Codo abajo



Fuente: Propia

Figura 22. Diagrama de cinemática inversa 2GDL Codo arriba

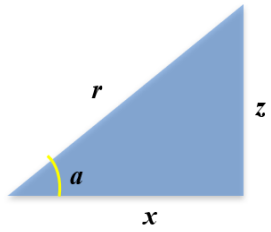


Fuente: Propia

- $L1$ es la longitud del eslabón 1.
- $L2$ es la longitud del eslabón 2.
- $q1$ es el ángulo que se forma entre el eslabón 1 y el eje x.
- $q2$ es el ángulo que se forma entre la prolongación del eslabón 1 y el eslabón 2.
- $q1p$ es un ángulo auxiliar para relacionar q1 en el segundo cuadrante.

El sistema de triángulos está conformado por un elemento de color azul y uno naranja. El primero relaciona el ángulo a para obtener una distancia r y el segundo a los ángulos internos b y c.

Figura 23. Análisis triángulo azul C. inversa 2GDL



Fuente: Propia

Por trigonometría se tiene que la expresión para a es:

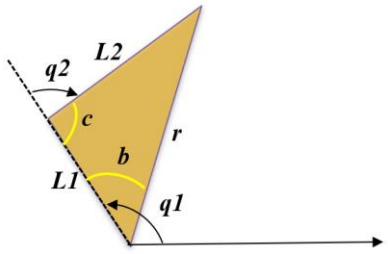
$$a = \text{atan}\left(\frac{z}{x}\right) (1)$$

Por teorema de Pitágoras se halla la expresión para r:

$$r = \sqrt{x^2 + z^2}\ (2)$$

Conociendo el valor de r, se emplea la ley del coseno como un método que permite relacionar los valores de b y c.

Figura 24. Análisis naranja C. inversa 2GDL



Fuente: Propia

$$\mathbf{L2^2 = L1^2 + r^2 - 2 * L1 * r * \cos(b)\ (3)}$$

Despejando ***b:***

$$\boldsymbol{b = acos\left(\frac{L1^2 + r^2 - L2^2}{2 * L1 * r}\right)(4)}$$

Para *c* se hace el mismo proceso:

$$\boldsymbol{c = acos\left(\frac{L1^2 - r^2 + L2^2}{2 * L1 * L2}\right)\ (5)}$$

Independientemente de la orientación codo y el cuadrante, el sistema de triángulos siempre se resuelve de la misma manera empleando las ecuaciones (1) al (5). Lo único que varía al obtener $\boldsymbol{q1}$ y $\boldsymbol{q2}$ mediante $\boldsymbol{a}$, $\boldsymbol{b}$ y $\boldsymbol{c}$ son las relaciones que se plantean entre estos.

Figura 25. Ecuaciones para hallar q1 y q2 con a, b y c

Codo abajo:

Cuadrante 1	Cuadrante 2
$q1 = a - b$	$q1 = 180° + q1p$
$q2 = 180° - c$	$q1p = b - a$
	$q2 = c - 180°$

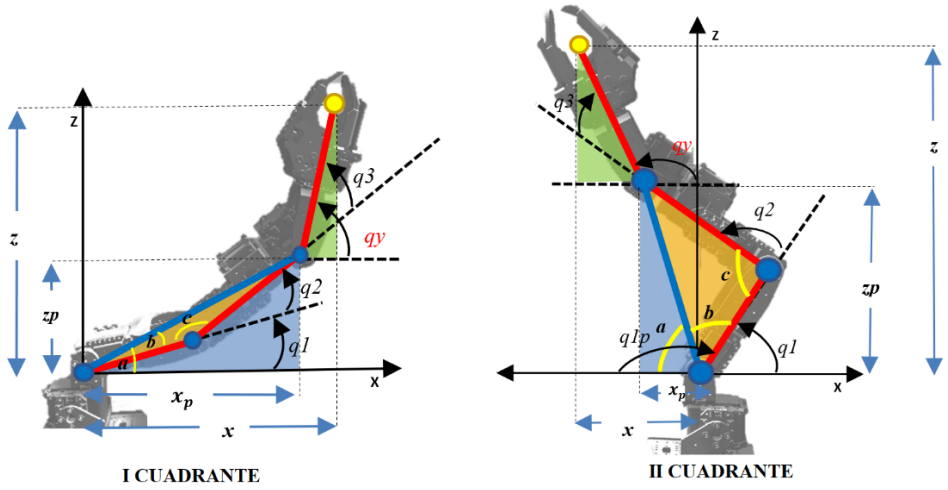
Codo arriba:

Cuadrante 1	Cuadrante 2
$q1 = a + b$	$q1 = 180° + q1p$
$q2 = c - 180°$	$q1p = -(a + b)$
	$q2 = 180° - c$

Fuente: Propia

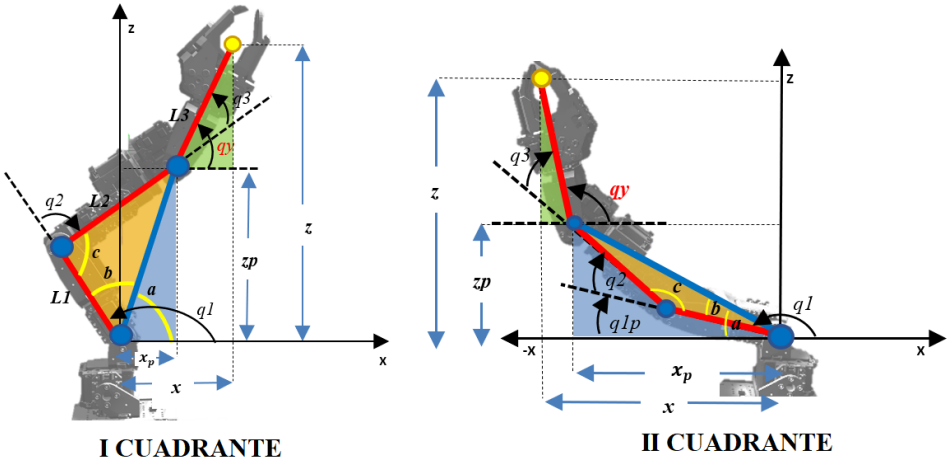
Para 3GDL, se aplica nuevamente el sistema de triángulos con el añadido de considerar el ángulo de orientación del efector final ($\boldsymbol{qy}$). El modelo resultante permite calcular fácilmente los ángulos de interés $\boldsymbol{q1}, \boldsymbol{q2}$ y $\boldsymbol{q3}$ en ambas configuraciones de codo (arriba y abajo).

Figura 26. Diagrama de cinemática inversa 3GDL Codo abajo



Fuente: Propia

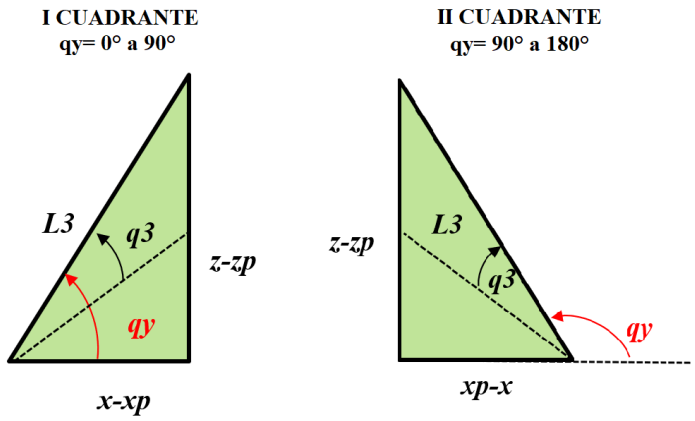
Figura 27. Diagrama de cinemática inversa 3GDL Codo arriba



Fuente: Propia

El ángulo de orientación del efector (***qy***), dependiendo del cuadrante donde se encuentre varia su forma de obtener las coordenadas (***xp, zp***) mediante el triángulo de color verde.

Figura 28. Análisis triángulo verde C. inversa 3GDL



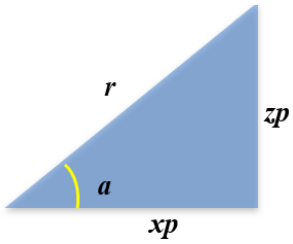
Fuente: Propia

$$xp = x - L3 * cos(qy) \ (6) \qquad xp = x - L3 * cos(180 - qy) \ (8)$$

$$zp = z - L3 * sen(qy) \ (7) \qquad zp = z - L3 * sen(180 - qy) \ (9)$$

Posteriormente, conociendo los valores de ***(xp, zp)***, se calculan los ángulos internos a, b y c para determinar los valores de ***q1***, ***q2*** y ***q3.***

Figura 29. Análisis triángulo azul C. inversa 3GDL



Fuente: Propia

Por trigonometría se tiene que la expresión para a es:

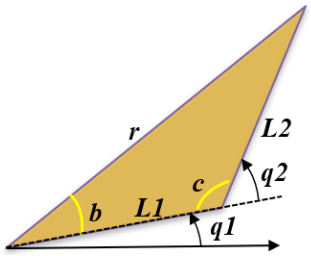
$$\mathbf{a = atan\left(\frac{zp}{xp}\right)} \ (10)$$

Por teorema de Pitágoras se halla la expresión para r:

$$\mathbf{r = \sqrt{xp^2 + zp^2}} \ (11)$$

Conociendo el valor de r, se emplea la ley del coseno como un método para relacionar los valores de b y c.

Figura 30. Análisis triángulo naranja C. inversa 3GDL



Fuente: Propia

$$\mathbf{L2^2 = L1^2 + r^2 - 2 * L1 * r * \cos(b)} \quad (12)$$

Despejando b:

$$b = acos\left(\frac{L1^2 + r^2 - L2^2}{2 * L1 * r}\right) \quad (13)$$

Para c se hace el mismo proceso:

$$c = acos\left(\frac{L1^2 - r^2 + L2^2}{2 * L1 * L2}\right) \quad (14)$$

Finalmente, $q1$ y $q2$ se obtienen operando los ángulos a, b y c hallados. Respecto a $q3$, mediante una relación que tienen los 3 ángulos de las articulaciones con qy es posible calcularlo. En la Figura 32, se traza una línea paralela al eslabón 1 para comprobar como qy es la suma de estos componentes.

Figura 31. Ecuaciones para hallar q1 y q2 con a, b y c

Codo abajo:

Cuadrante 1

$q1 = a - b$

$q2 = 180 - c$

$q3 = qy - q1 - q2$

Cuadrante 2

$q1 = 180 + q1p$ $\quad q1p = b - a$

$q2 = c - 180$

$q3 = qy - q1 - q2$

Codo arriba:

Cuadrante 1

$q1 = a + b$

$q2 = c - 180$

$q3 = qy - q1 - q2$

Cuadrante 2

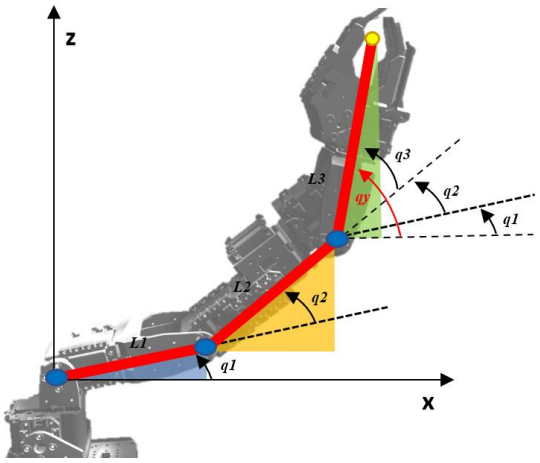
$q1 = 180 + q1p$ $\quad q1p = -(a + b)$

$q2 = 180 - c$

$q3 = qy - q1p - q2$

Fuente: Propia

Figura 32. Relación de qy con los ángulos de las articulaciones del sistema (q1, q2 y q3)



Fuente: Propia

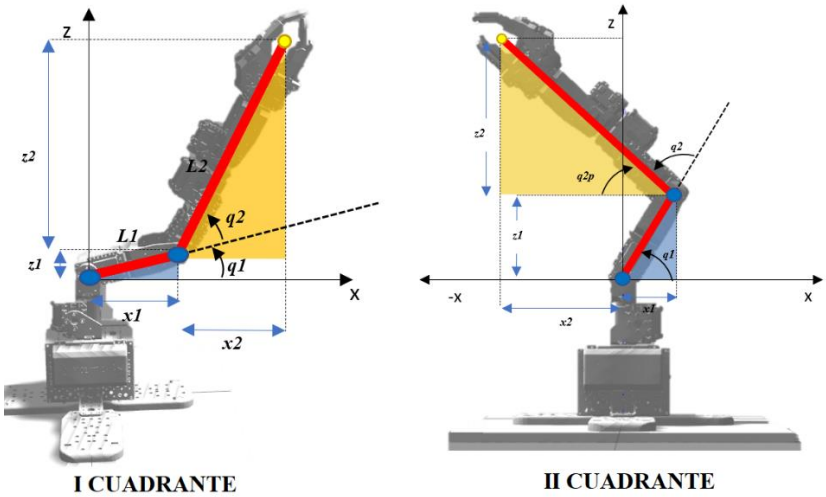
$$qy = q1 + q2 + q3 \ (15)$$

5.5.3 Cinemática directa

Es un método para calcular las coordenadas del efector final a partir de los ángulos de rotación de las articulaciones del robot, es decir, según los ángulos ***q1, q2*** y ***q3*** dependiendo de cuántos grados de libertad tenga el modelo (2GDL o 3GDL).

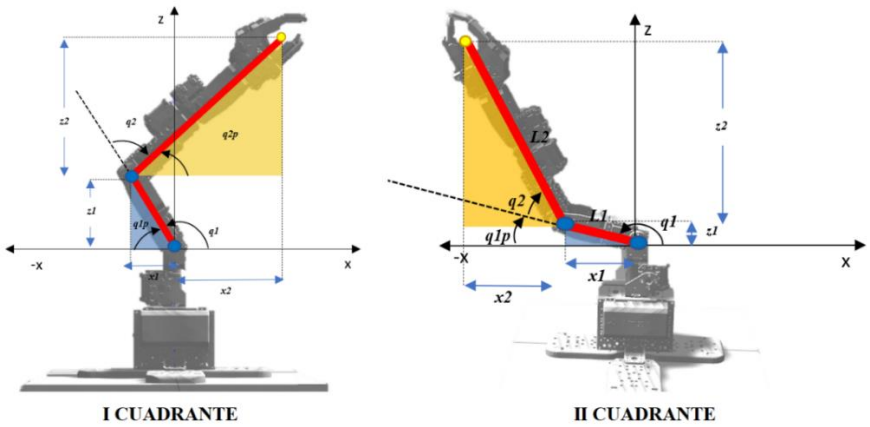
Para 2GDL, se plantea el modelo a través de un sistema de triángulos que permite calcular fácilmente la coordenada del efector final (x, z) en ambas configuraciones de codo (arriba y abajo).

Figura 33. Diagrama de cinemática directa 2GDL Codo abajo



Fuente: Propia

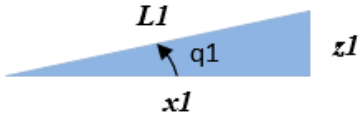
Figura 34. Diagrama de cinemática directa 2GDL Codo arriba



Fuente: Propia

Como se conoce el valor de ***q1*** y ***q2***, independientemente del cuadrante y orientación de codo, se hallan las coordenadas de cada triángulo rectángulo por medio de relaciones trigonométricas y posteriormente, se suman los segmentos de cada eje para hallar la coordenada (***x, z***). Por ejemplo, para el primer cuadrante en codo abajo el procedimiento es el siguiente:

Figura 35. Análisis triángulo azul C. directa 2GDL



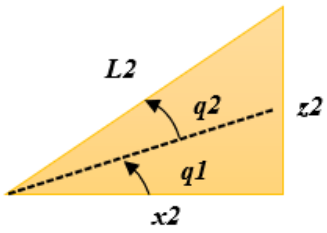
Fuente: Propia

Para $x1$ y $z1$:

$$x1 = L1 * \cos(q1) \ (\mathbf{16})$$

$$z1 = L1 * \text{sen}\ (q1) \ (\mathbf{17})$$

Figura 36. Análisis triángulo naranjal C. directa 2GDL



Fuente: Propia

Para $x2$ y $z2$:

$$x2 = L2 * \cos(q1 + q2)\ (\mathbf{18})$$

$$z2 = L2 * \text{sen}(q1 + q2)\ (\mathbf{19})$$

Se suman los segmentos de cada eje para obtener $\boldsymbol{x}$ y $\boldsymbol{z}$:

Para $\boldsymbol{x}$:

$$\boldsymbol{x} = x1 + x2\ (\mathbf{20})$$

$$x = L1 * \cos\ (q1) + L2 * \cos\ (q1 + q2)$$

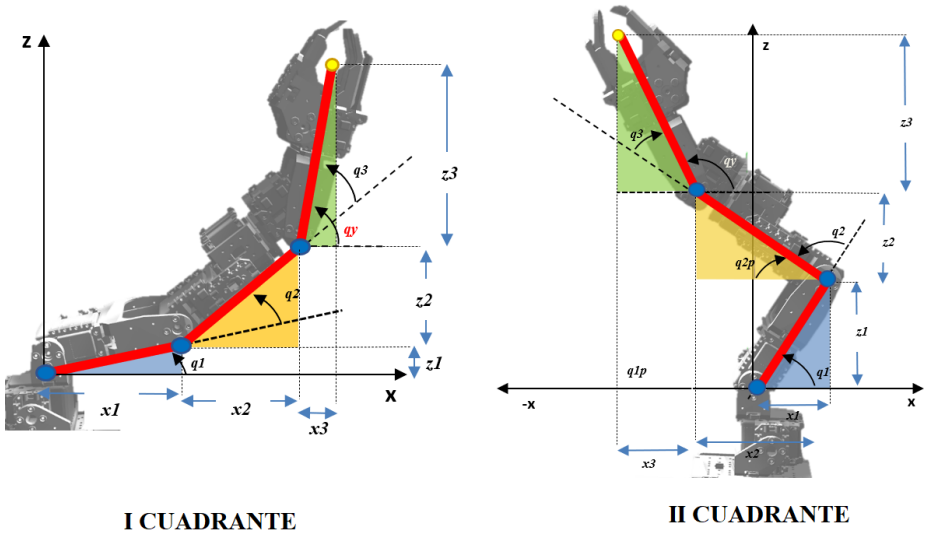
Para $\boldsymbol{z}$:

$$\boldsymbol{z} = z1 + z2\ (\mathbf{21})$$

$$z = L1 * \text{sen}\ (q1) + L2 * \text{sen}\ (q1 + q2)$$

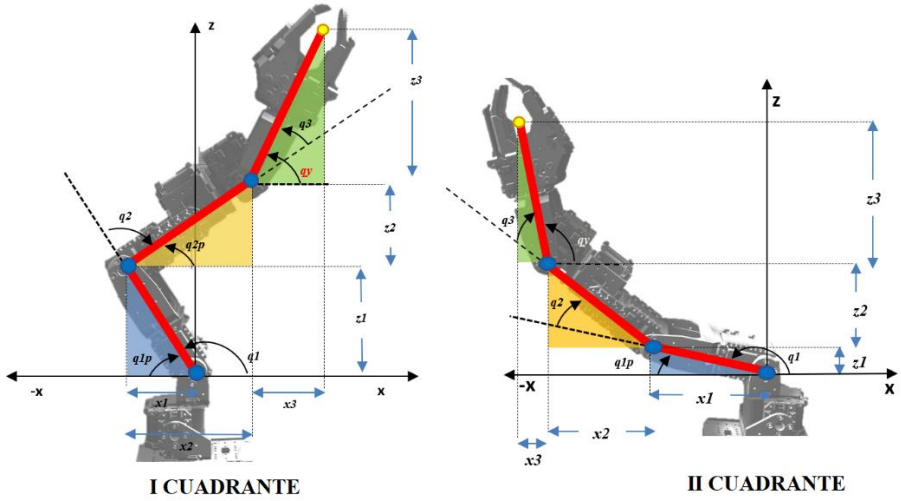
Para 3GDL, se aplica el mismo método, pero considerando que son 3 segmentos los que deben sumarse para hallar las coordenadas del efector final ($\boldsymbol{x}$, $\boldsymbol{z}$).

Figura 37. Diagrama de cinemática directa 3GDL Codo abajo



Fuente: Propia

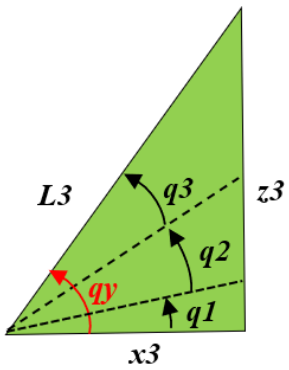
Figura 38. Diagrama de cinemática directa 3GDL Codo arriba



Fuente: Propia

El triángulo de color azul y naranja en el modelo de 3GDL se resuelven de igual manera que en el de 2GDL, empleando las ecuaciones del (16) al (21). Esto se debe a que la única variación entre ambos sistemas es el valor de las distancias ***L1*** y ***L2.*** Para el triángulo de color verde, similar a la demostración en la Figura 32, se realiza un paralelo al eslabón 1 y se dibujan los ángulos ***q1***, ***q2*** y ***q3*** para ver su relación respecto a ***qy***.

Figura 39. Análisis triángulo verde C. directa 3GDL



Fuente: Propia

$$x3 = L3 * Cos(q1 + q2 + q3) \ (\mathbf{23})$$

$$z3 = L3 * Sen(q1 + q2 + q3) \ (\mathbf{24})$$

Para $\boldsymbol{x}$:

$$x = x1 + x2 + x3 \ (\mathbf{25})$$

$$x = L1 * \cos(q1) + L2 * \cos(q1 + q2) + L3 * cos(q1 + q2 + q3)$$

Para $\boldsymbol{z}$:

$$z = z1 + z2 + z3 \ (\mathbf{26})$$

$$z = L1 * \text{sen}\ (q1) + L2 * \text{sen}(q1 + q2) + L3 * sen(q1 + q2 + q3)$$

Para $\boldsymbol{qy}$:

$$qy = \boldsymbol{q1} + \boldsymbol{q2} + \boldsymbol{q3} \ (\mathbf{27})$$

5.5.4 Espacio de trabajo

Se define como el conjunto de puntos que puede alcanzar el robot alrededor de sí mismo considerando su configuración (2GDL-3GDL), el tamaño de sus eslabones y los límites de sus articulaciones. En otras palabras, el espacio de trabajo es el rango de movilidad del robot.

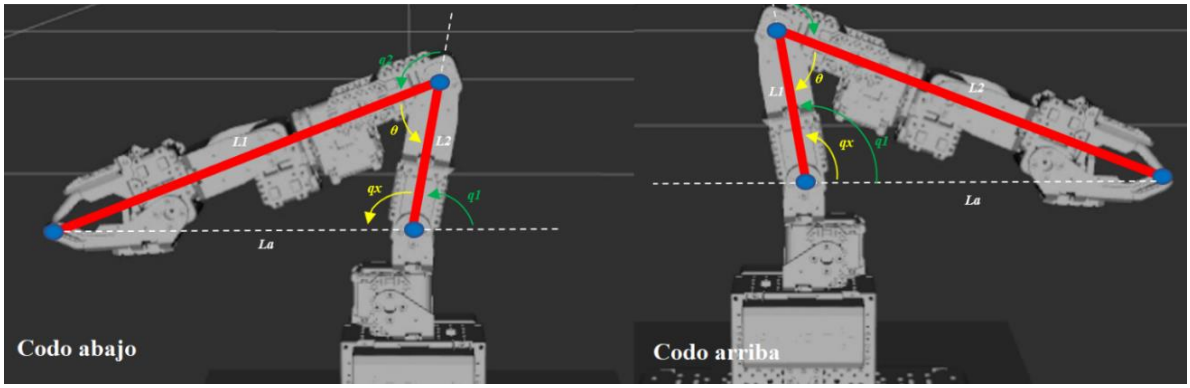
Su construcción respecto a 2GDL, se hace asumiendo intervalos de giro para las articulaciones de interés. Estos rangos de operación que se asumen para $(\boldsymbol{q1}, \boldsymbol{q2})$ a través del método de cinemática directa permiten obtener las coordenadas en el plano para realizar el trazo del espacio de trabajo.

Tabla 1. Límites para trazar el espacio de trabajo 2GDL

		Limites		
Orientación codo	Ángulo de articulación	Barrido general de ***q1*** manteniendo constante ***q2***	***q1*** constante y ***q2*** varía hasta el valor soportado por su articulación (120°)	***q2*** constante y ***q1*** varía hasta un valor calculado
Abajo	***q1***	0° a 180 °	0 °	0° a 80.73°
	q2	0°	0° a 120°	120°
Arriba	***q1***	180 ° a 0°	180 °	180° a 99.26°
	q2	0°	0° a -120°	-120°

Para la construcción del tercer límite presente en la Tabla 1, como se desconoce el valor hasta el que puede girar ***q1*** manteniendo un valor de ***q2***. Se plantea el siguiente sistema de triángulos:

Figura 40. Análisis para calcular ultimo límite de Espacio de trabajo 2GDL



Fuente: Propia

$$C.Abajo \rightarrow \theta = 180° + q2 \ (\mathbf{28})$$

$$C.Arriba \rightarrow \theta = 180° - q2 \ (\mathbf{29})$$

A través de la ley del coseno:

$$La = \sqrt{L1^2 + L2^2 - (2L1 * L2 * \cos(\theta))} \ (\mathbf{30})$$

A través de la ley del coseno:

$$qx = \text{acos}\left(\frac{La^2 + L1^2 - L2^2}{2 * L1 * La}\right) (\mathbf{31})$$

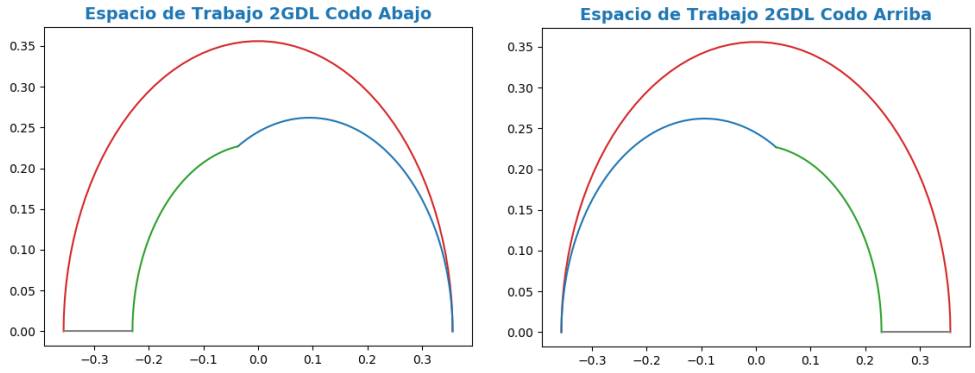
Finalmente, conociendo qx, q1 manteniendo q2 variará hasta:

$$C.Abajo \rightarrow \boldsymbol{q1} = 180° - \text{qx} \ (\mathbf{32})$$

$$C.Arriba \rightarrow \boldsymbol{q1} = \text{qx} \ (\mathbf{33})$$

Finalmente, tras calcular los 3 límites, la gráfica equivalente al área de trabajo para el robot con una configuración de 2 grados de libertad, es la siguiente:

Figura 41. Gráfica de Espacio de trabajo 2GDL



Nota: Cada color de trazo del espacio de trabajo se corresponde con el límite que lo produce de la Tabla 1

Fuente: Propia

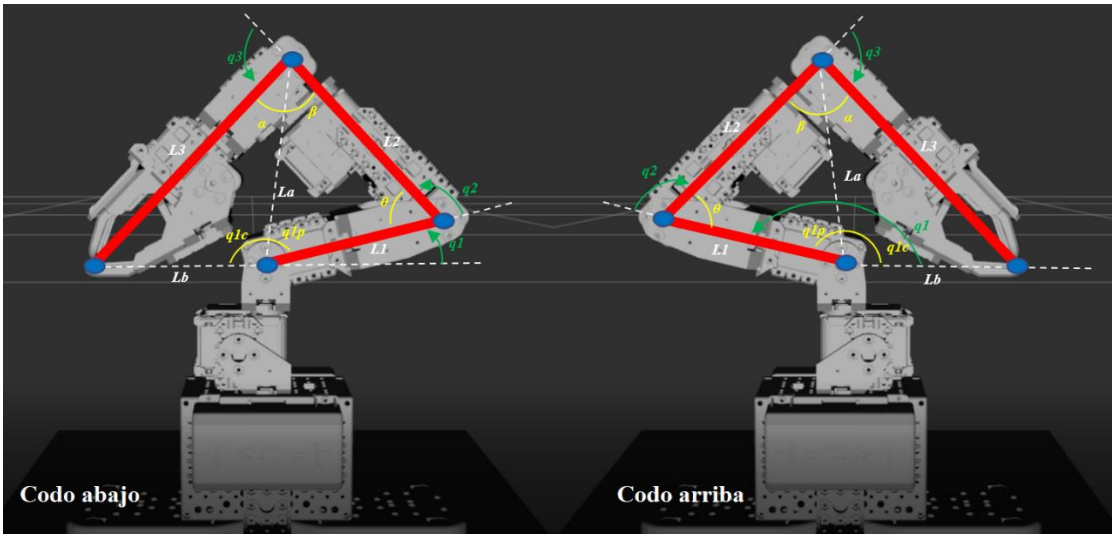
Respecto a 3GDL, se trazan dos espacios de trabajo. Uno general, aplicando la misma lógica que en 2GDL, asumiendo distintos intervalos de giro considerando los límites de las articulaciones. Resultando en rangos de operación para $(\boldsymbol{q1}, \boldsymbol{q2}, \boldsymbol{q3})$ a los cuales se les aplica el método de cinemática directa para obtener las coordenadas del espacio de trabajo.

Tabla 2. Límites para trazar el espacio de trabajo 3GDL

		Limites			
Orientación codo	Ángulo de articulación	Barrido de ***q1*** manteniendo constante ***q2*** y ***q3***	***q1*** y ***q2*** constante y ***q3*** varía hasta su límite máximo (120°)	***q1*** y ***q3*** constante y ***q2*** varía hasta su límite máximo (90°)	***q2*** y ***q3*** constante y ***q1*** varía hasta un valor calculado
Abajo	***q1***	0° a 180 °	0 °	0 °	0° a 13.45°
	q2	0°	0°	0° a 120°	120°
	q3	0°	0° a 90°	90°	90°
Arriba	***q1***	180 ° a 0°	180°	180 °	180° a 166.54°
	q2	0°	0°	0° a -120°	-120°
	q3	0°	0° a -90°	0° a 120°	90°

Para la construcción del último límite, como se desconoce el valor hasta el que puede girar ***q1*** para mantener el valor de ***q2*** y ***q3***. Se plantea el siguiente sistema de triángulos:

Figura 42. Análisis para calcular ultimo límite de Espacio de trabajo 3GDL



Fuente: Propia

θ *se obtiene empleando la ecuación* (**28**) *o* (**29**) *según la orientaciópn de codo.*

La se obtiene empleando la ecuación (**30**)

Se halla q1p a través de la ley del coseno:

$$q1p = \text{acos}\left(\frac{L1^2 + La^2 - L2^2}{2 * L1 * La}\right) (\mathbf{34})$$

Finalmente, conociendo q1p:

$$C.Abajo \rightarrow \boldsymbol{\alpha} = 180° - \text{q3} - \beta \ (\mathbf{35})$$

$$C.Arriba \rightarrow \boldsymbol{\alpha} = 180° + \text{q3} - \beta \ (\mathbf{36})$$

Se halla q1c a través de la ley del coseno:

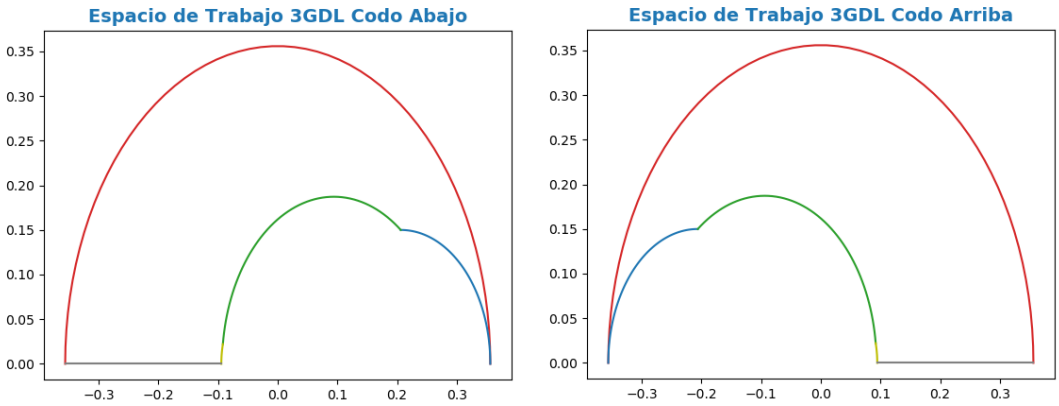
$$q1c = \text{acos}\left(\frac{La^2 + Lb^2 - L3^2}{2 * La * Lb}\right) (\mathbf{37})$$

Finalmente, conociendo q1p y q1c:

$$C.Abajo \rightarrow \boldsymbol{q1} = 180° - \text{q1p} - \text{q1c} \ (\mathbf{38})$$

$$C.Arriba \rightarrow \boldsymbol{q1} = \text{q1p} + \text{q1c} \ (\mathbf{39})$$

Figura 43. Gráfica de Espacio de trabajo 3GDL

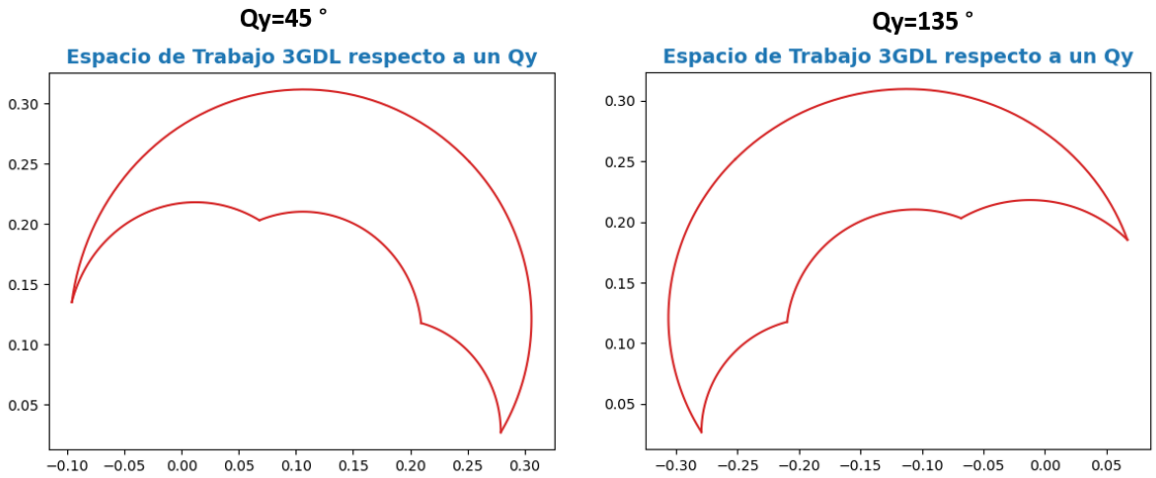


Nota: Cada color de trazo del espacio de trabajo se corresponde con el límite que lo produce de la Tabla 2

Fuente: Propia

El segundo espacio de trabajo respecto a 3GDL, aplica la misma lógica anterior, pero considerando que los rangos de operación para $(\boldsymbol{q1}, \boldsymbol{q2}, \boldsymbol{q3})$ deben garantizar que se mantenga el ángulo de orientación del efector final $\boldsymbol{qy}$. Resultando en un espacio de trabajo exclusivo para ese valor de ángulo.

Figura 44. Gráfica de Espacio de trabajo 3GDL respecto a un valor de qy



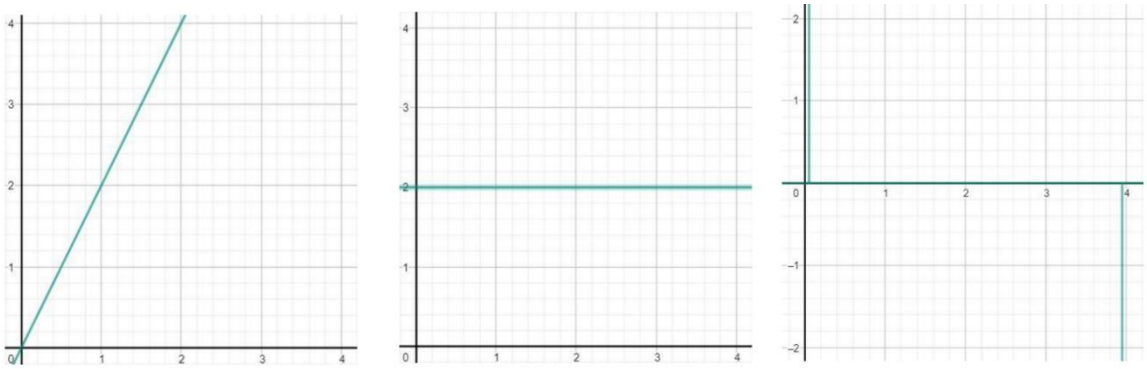
Fuente: Propia

5.5.5 Trayectorias

Inicialmente para el trazo de la trayectoria se pensó usar un movimiento lineal debido a que idealmente la velocidad sería constante y la aceleración nula. Pero este planteamiento aplicado en la realidad no resulta cierto, debido a que un movimiento realiza una aceleración al inicio y una desaceleración al final. Esto se

vería representado con dos picos respecto a la aceleración (Figura 45), trayendo consigo un riesgo que puede afectar los dispositivos físicos involucrados.

Figura 45. Gráficas de posición, velocidad y aceleración para un movimiento lineal



Fuente: Propia

Tomando como base teórica lo expuesto sobre interpoladores cúbicos en el libro "FUNDAMENTOS DE ROBÓTICA" (Barrientos, Penín, Balaguer, & Aracil, 1997), la alternativa que se planteó para esta problemática fue utilizar un Spline cúbico en lugar de una línea recta para la construcción de la trayectoria. El polinomio planteado es el siguiente:

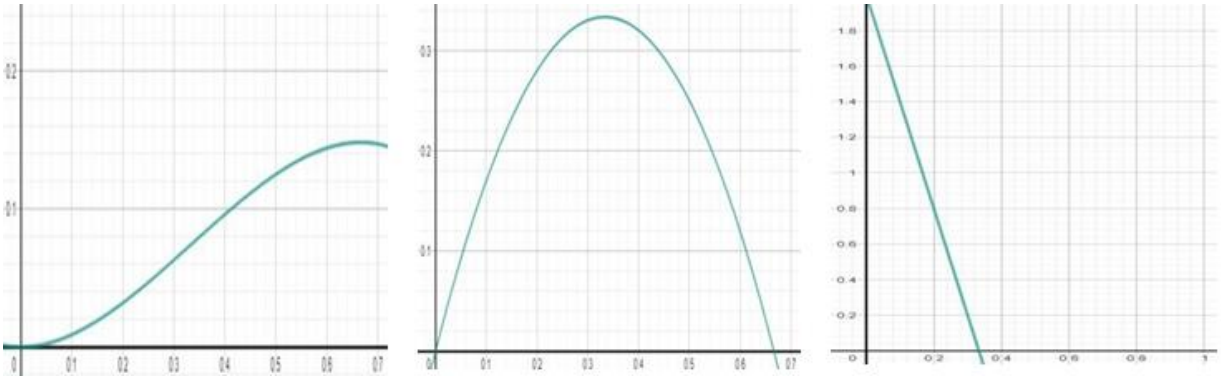
$$\boldsymbol{f(t) = a * t^3 + b * t^2 + c * t + d} \ \boldsymbol{(40)}$$

Válido para posicionamiento espacial y movimiento articular

Este polinomio permite:

- Un movimiento suave y en general, un ajuste a los puntos tabulados.
- La velocidad cumple la función de acelerar (al inicio) y desacelerar (al final).
- Corregir el error de los picos de aceleración.

Figura 46. Gráficas de posición, velocidad y aceleración para un movimiento empleando un Spline cúbico



Fuente: Propia

A partir de la ecuación del Spline Cúbico planteada y sus derivadas se obtiene la ecuación de posición, velocidad y aceleración para trayectorias:

$$\boldsymbol{f(t) = a \cdot t^3 + b \cdot t^2 + c \cdot t + d} \ \boldsymbol{(41)}$$

$$\boldsymbol{f'(t) = v(t) = 3 \cdot a \cdot t^2 + 2 \cdot b \cdot t + c} \ \boldsymbol{(42)}$$

$$\boldsymbol{f''(t) = a(t) = 6 \cdot a \cdot t + 2 \cdot b} \ \boldsymbol{(43)}$$

6. METODOLOGÍA

Para la realización de este proyecto el pasante se encontraba bajo la guía y dirección del ingeniero Mario Arbulú, supervisor en la empresa ROBOTICS 4.0 S.A.S de este proyecto.

Las actividades se realizaron de manera presencial en las instalaciones de Tecnoparque SENA en la ciudad de Neiva, cumpliendo con la intensidad horaria definida por la Universidad Surcolombiana de 20 horas semanales durante un periodo mínimo de 6 meses.

Con base en los objetivos general y específicos descritos previamente (numeral 4) la metodología a utilizar durante la pasantía incluye procedimientos y técnicas de carácter investigativo y/o práctico con el fin de entregar el módulo didáctico planteado en el objetivo general.

En términos generales, este proyecto de pasantía se divide en 4 fases:

- **Fase 1: Aprender el funcionamiento de la arquitectura ROS para implementar algoritmos de modelado cinemático en robot OpenBotv v1.**

Previo a la implementación de los algoritmos de modelado cinemático. Durante esta fase, el pasante se dedicó a estudiar el funcionamiento de ROS: Su estructura, las herramientas de simulación que posee (Gazebo y Rviz) y la manera de manipular tanto el robot en el entorno simulado como real empleando este software.

El aprendizaje se realizó mediante el estudio de tutoriales, material proporcionado por Robotics 4.0 S.A.S (Simulaciones, Guías, avances previos de la empresa en ROS, etc.) y asesorías del supervisor a cargo del proyecto.

Por último, la implementación de ROS durante la pasantía se realizó en un equipo con S.O. Ubuntu y la programación de los códigos se realizó a través de Python 3.

- **Fase 2: Implementar e integrar modelos cinemáticos desarrollados en el robot Openbotv1 a través de ROS.**

Durante esta fase, en conjunto con el robot OpenBotv v1, el pasante realizó en el software ROS la aplicación de distintos conceptos básicos de robótica, como: cinemática inversa, cinemática directa, espacio de trabajo y trayectorias.

Los resultados de este proceso, se resumen en la creación de códigos en Python y demostraciones tanto para el robot simulado y real que permiten la verificación de las temáticas mencionadas anteriormente.

- **Fase 3: Desarrollar un material académico (guías de trabajo, videos, etc.) para explicar paso a paso la integración de ROS con Openbotv1 y la aplicación de los modelos cinemáticos.**

Una vez alcanzadas las metas de la fase 2, el pasante basándose en los resultados obtenidos en el proceso de integración de ROS con OpenBotv v1 y la aplicación de los modelos cinemáticos, documentó en forma de guías de trabajo y videos el paso a paso que llevó a cabo. Este material realizado por el pasante será el contenido para el módulo didáctico que conservará Robotics 4.0 S.A.S.

- **Fase 4: Análisis y Documentación de los resultados finales del proceso de pasantía.**

Por último, el documento final de la pasantía contiene un informe detallado de los cálculos matemáticos, las simulaciones, las pruebas realizadas con el robot real y los códigos que se utilizaron para la realización de las distintas fases del proyecto.

7. DESARROLLO DE LA PASANTÍA Y ANÁLISIS DE RESULTADOS

En esta sección, se mostrarán los resultados obtenidos durante el proceso que llevó a cabo el pasante para el desarrollo del módulo didáctico sobre la integración de OpenBotv v1 con el software ROS. Esto se resume en la socialización de:

- Los ajustes que tuvo que realizar el pasante para el acople del hardware (OpenBotv v1) y el software (ROS).
- Las demostraciones o demos realizadas para comprobar la implementación de los modelos cinemáticos en ROS.
- El estado actual del módulo didáctico tras la finalización del proceso de pasantía.

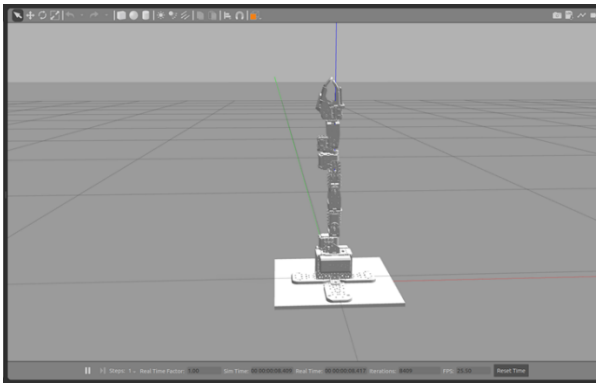
Previo a mostrar los resultados, se mostrará el material que fue suministrado al pasante por Robotics 4.0 S.A.S, para así, comprender el punto de inicio del proyecto. Inicialmente, el pasante contaba con lo siguiente:

Figura 47. Workspace de ROS suministrado por Robotics 4.0 S.A.S

Fuente: Propia

- Un Workspace en ROS que incluye paquetes que definen las características (Dimensiones, controladores, articulaciones, etc.) del modelo de OpenBotv v1 en las herramientas de simulación: Gazebo y Rviz (Ver sección 5.26).
- Un ejemplo de una simulación en Gazebo que permitía simplemente visualizar el robot en un entorno 3D.

Figura 48. Ejemplo de simulación suministrado por Robotics 4.0 S.A.S



Fuente: Propia

7.1 AJUSTES REALIZADOS PARA EL ACOPLE DEL HARDWARE (OPENBOTV V1) Y EL SOFTWARE (ROS)

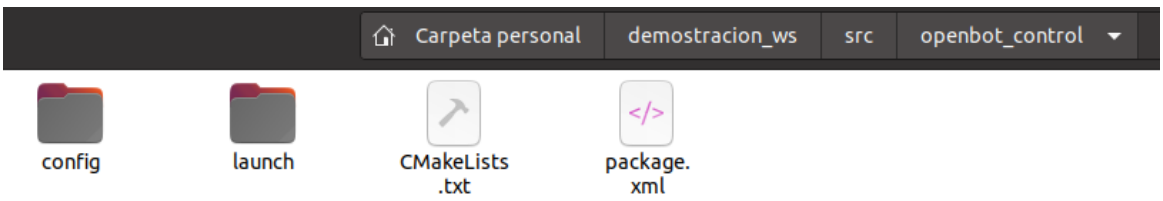
Inicialmente solo se contaba con una simulación básica en la cual podía visualizarse el robot en un entorno 3D (Gazebo). Por lo que una de las primeras tareas que realizó el pasante fue ser capaz de manipular tanto el robot OpenBotv v1 simulado y real desde ROS. A continuación, se describe el proceso que llevó a cabo para realizar esta tarea.

- **Control sobre una articulación de OpenBotv v1 en entorno simulado:**

Para ser capaces de manipular las articulaciones del OpenBotv v1 en el entorno simulado fue necesario la implementación de la librería ROS Control (Ver sección 5.25) que permite usar Effort_Controllers/Joint_position_controllers. Estos son controladores PID que permiten introducir un valor deseado en radianes de 0 a 2π y generar en cada articulación un valor deseado de torque equivalente para llegar a esa posición. Para la correcta implementación de esta librería en el Workspace, fue necesario aplicar los siguientes cambios:

Creación de un paquete de Control: Como ROS opera mediante el uso de paquetes, se crea uno relacionado a la librería ROS Control y se le denomina como “openbot_control”.

Figura 49. Estructura del paquete openbot_control



Fuente: Propia

Creación de un archivo de configuración: En el paquete openbot_control se crea un archivo llamado controller.yaml. Dentro de este se definen las articulaciones en las cuales se va a ejercer control, el valor de las constantes PID y el tipo de controlador a usar, que se decidió que fueran los **effort_controllers/ JointPositionController** debido a que la configuración es para servomotores.

Figura 50. Estructura de archivo controller.yaml

```yaml
openbot_v1:
  # Publish all joint states -----------------------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  # Position Controllers ---------------------------------------
  arm1_arm2_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: arm1_arm2_joint
    pid: {p: 450.0, i: 100.0, d: 0.8}
  arm2_arm3_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: arm2_arm3_joint
    pid: {p: 300, i: 100.0, d: 0.01}
  arm3_arm4_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: arm3_arm4_joint
    pid: {p: 50.0, i: 1, d: 0.06}
  arm4_clamp1_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: arm4_clamp1_joint
    pid: {p: 500, i: 500, d: 0.9}
  clamp1_clamp2_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: clamp1_clamp2_joint
    pid: {p: 70.0, i: 1.0, d: 0.01}
  base_arm1_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: base_arm1_joint
    pid: {p: 50.0, i: 1.0, d: 0.5}
```

Fuente: Propia

Modificación de archivo Xacro para introducir transmisores: El uso de los controladores implica definir dentro del modelo de OpenBotv v1 la relación entre cada actuador y la articulación. Esto se hace con una estructura llamada transmisor[4] que se define para cada una de las articulaciones del OpenBotv v1 y la pinza.

Estos cambios se realizan en el paquete openbot_v1_description, específicamente, el archivo openbot_v1.xacro, debido a que es el encargado de definir las características del modelo simulado del robot.

[4] Apache 2.0. (16 de febrero de 2023). Gazebo : Tutorial : ROS control. Obtenido de https://classic.gazebosim.org/tutorials?tut=ros_control#Aboutros_control

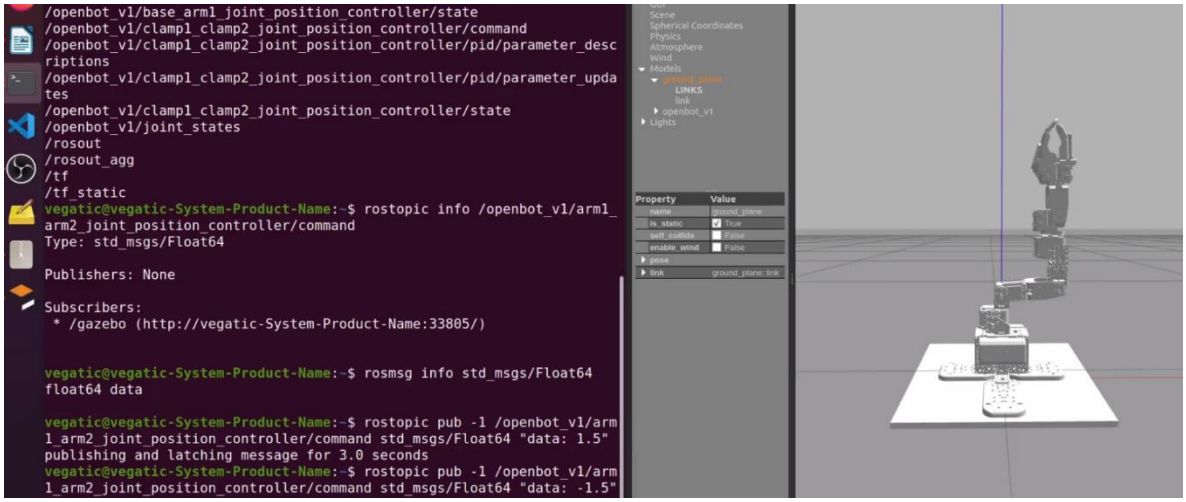
Figura 51. Definición de transmisores en Openbot_v1.xacro

```
<transmission name="clamp1_clamp2_joint_tran">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="clamp1_clamp2_joint">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
  </joint>
  <actuator name="clamp1_clamp2_joint_actr">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

Fuente: Propia

Posterior a esto, se procedió a comprobar el funcionamiento de las articulaciones en el OpenBotv v1 simulado enviando ángulos en radianes a distintas articulaciones. Este proceso de manipulación del robot se realizó desde el terminal mediante comandos y a través de un script en Python usando la librería Rospy[5]. Para ambos casos, se aplican los conceptos de nodo Publisher y Subscriber (Ver sección 5.24), funcionando el código y el terminal como emisor y la articulación del robot que recibe el ángulo a mover en radianes como el receptor.

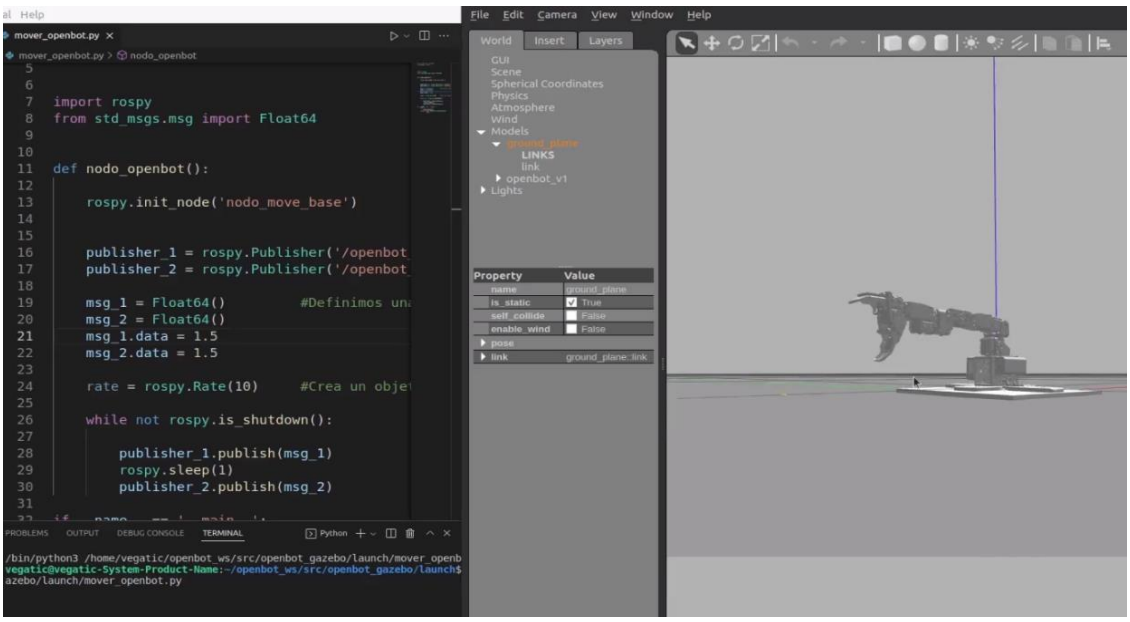
Figura 52. OpenBotv v1 manipulado a través de comandos en terminal



Fuente: Propia

[5] ROS Wiki. (12 de febrero de 2023). Obtenido de rospy: http://wiki.ros.org/rospy

Figura 53. OpenBotv v1 manipulado desde script en Python empleando Rospy



Fuente: Propia

Tras la sesión de pruebas, se observa que es más fácil manipular el robot desde Python en conjunto con ROS debido a que pueden definirse y controlarse varios canales de comunicación (Topics) a la vez, relegando el método por terminal como una forma de comprobar que existe comunicación con la articulación. Por esta razón, la manipulación desde Python será la forma por defecto en la que se operará el robot simulado al implementar los modelos cinemáticos.

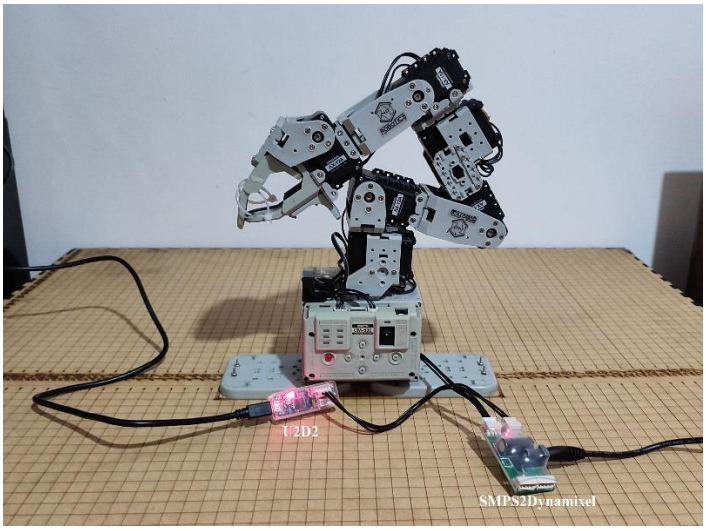
- **Control sobre una articulación de OpenBotv v1 físico:**

Como se mencionó en el literal 6.3.3, el OpenBotv v1 emplea 6 motores de la referencia Dynamixel Ax-12. Estos cuentan con una librería especializada para su integración con ROS y que facilita el movimiento de todas las articulaciones del robot a la vez. Se denomina Dynamixel Workbench[6] y es la que se usó para el desarrollo del proyecto.

Adicionalmente, el montaje empleado para la comunicación con el OpenBotv v1 incluye una convertidor USB U2D2 (Ver sección 5.4.1), un cable USB, un SMPS2Dynamixel (Ver sección 5.4.2) y una fuente de alimentación para los motores (12 V DC).

[6] ROBOTIS. (13 de febrero de 2023). ROBOTIS e-Manual. Obtenido de DYNAMIXEL Workbench: https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_workbench/

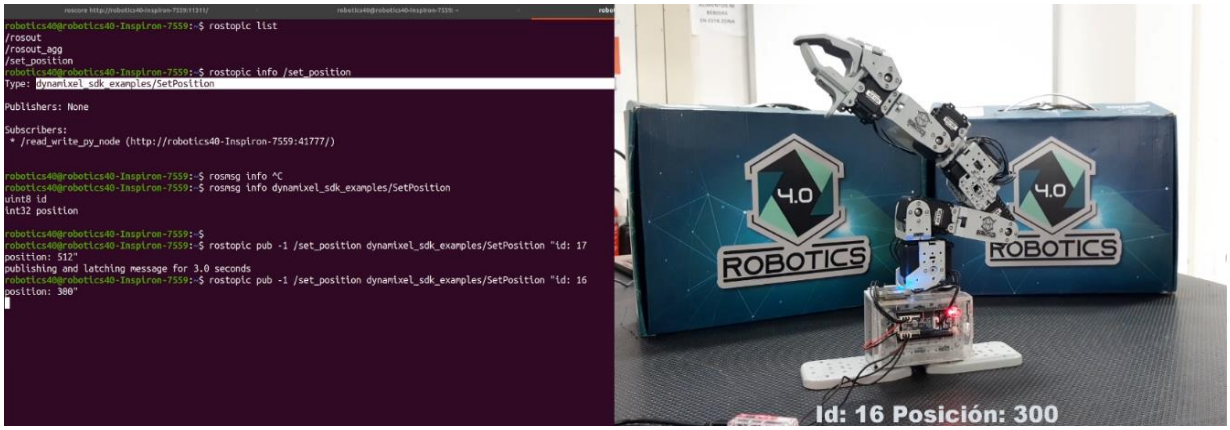
Figura 54. Montaje para manipular OpenBotv v1



Fuente: Propia

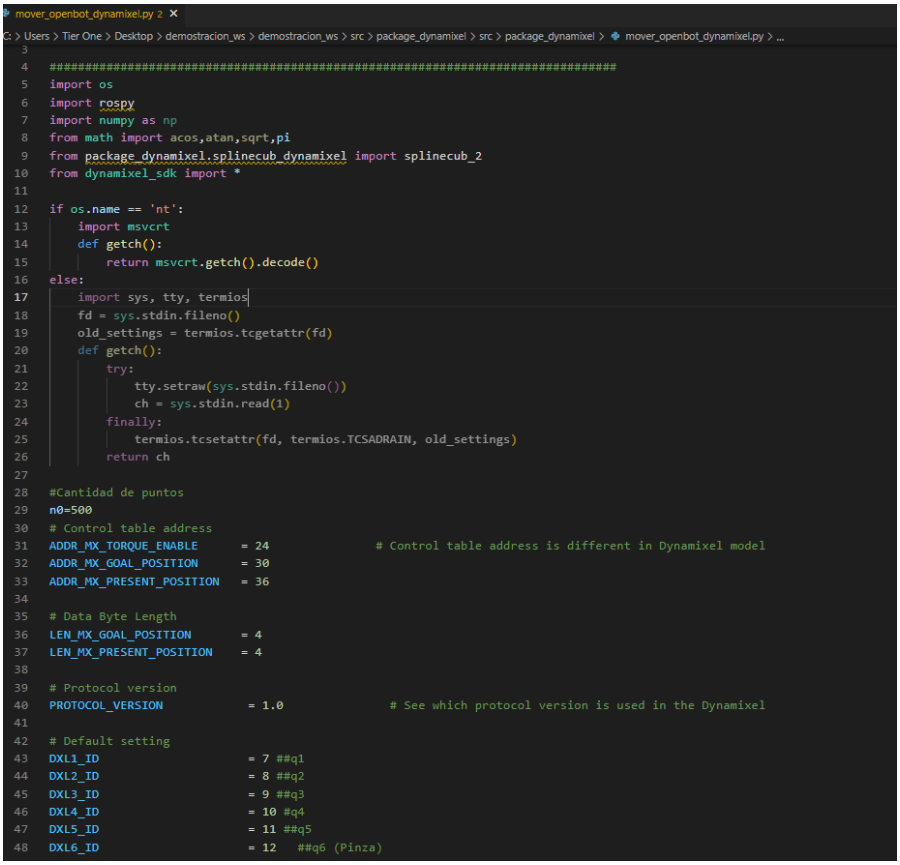
Para comprobar el funcionamiento del control de las articulaciones del robot, se realiza el envío de posiciones de 0 a 1023 (Rango soportado por la referencia de Dynamixel AX-12) tanto por comandos en el terminal como la manipulación por medio de un script de Python empleando las librerías de Dynamixel_workbench.

Figura 55. OpenBotv v1 físico manipulado a través de comandos en terminal



Fuente: Propia

Figura 56. Script empleado para manipular OpenBotv v1 desde Python



```
mover_openbot_dynamixel.py 2 ×
C: > Users > Tier One > Desktop > demostracion_ws > demostracion_ws > src > package_dynamixel > src > package_dynamixel > mover_openbot_dynamixel.py > ...
3
4   ###############################################################################
5   import os
6   import rospy
7   import numpy as np
8   from math import acos,atan,sqrt,pi
9   from package_dynamixel.splinecub_dynamixel import splinecub_2
10  from dynamixel_sdk import *
11
12  if os.name == 'nt':
13      import msvcrt
14      def getch():
15          return msvcrt.getch().decode()
16  else:
17      import sys, tty, termios
18      fd = sys.stdin.fileno()
19      old_settings = termios.tcgetattr(fd)
20      def getch():
21          try:
22              tty.setraw(sys.stdin.fileno())
23              ch = sys.stdin.read(1)
24          finally:
25              termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
26          return ch
27
28  #Cantidad de puntos
29  n0=500
30  # Control table address
31  ADDR_MX_TORQUE_ENABLE      = 24               # Control table address is different in Dynamixel model
32  ADDR_MX_GOAL_POSITION      = 30
33  ADDR_MX_PRESENT_POSITION   = 36
34
35  # Data Byte Length
36  LEN_MX_GOAL_POSITION       = 4
37  LEN_MX_PRESENT_POSITION    = 4
38
39  # Protocol version
40  PROTOCOL_VERSION            = 1.0              # See which protocol version is used in the Dynamixel
41
42  # Default setting
43  DXL1_ID                     = 7 ##q1
44  DXL2_ID                     = 8 ##q2
45  DXL3_ID                     = 9 ##q3
46  DXL4_ID                     = 10 #q4
47  DXL5_ID                     = 11 ##q5
48  DXL6_ID                     = 12   ##q6 (Pinza)
```

Fuente: Propia

Tras la sesión de pruebas, se observa que es más fácil manipular el robot desde Python en conjunto con ROS debido a que pueden definirse y controlarse varias ID de motores a la vez, relegando el método de comunicación por terminal como una forma rápida de verificar la conexión con el Dynamixel. Por esta razón, esta será la forma por defecto en la que se manipulará el robot real al implementar los modelos cinemáticos.

Implementación de un Spline Cúbico: Tras comprobar que era posible el control de las articulaciones del robot físico empleando la librería, fue necesario el uso de un método matemático que permitiera planear movimientos que involucran el recorrido de una serie de puntos de una coordenada espacial a otra.

El método que se decidió usar fue un Spline cúbico por las razones que se especifican en la sección 5.5.5. Este se encuentra representado mediante la siguiente función de Python que considera una posición inicial (Po), una posición final (Pf), el tiempo de muestreo (Tf) y la cantidad de puntos (n).

Figura 57. Función desarrollada para aplicar método de Spline cúbico

```
import numpy as np
from math import trunc

def splinecub_2 (Po, Pf, Tf, n):

    dt=(Tf-0)/(n-1)
    t=np.arange(0.0, Tf+0.00001, dt)


    C=[ [0,0,0,1],
        [0,0,1,0],
        [Tf**3 , Tf**2 ,Tf,1],
        [3*(Tf**2), 2*Tf,  1, 0]
        ]

    B= [[Po], [0], [Pf], [0]]



    A= np.dot(np.linalg.inv(C),B)

    a=A[0]
    b=A[1]
    c=A[2]
    d=A[3]
    t_3= t**3
    t_2=t**2
    P=a*t_3+b*t_2+c*t+d
    P=P.astype(int)
    return P
```

Fuente: Propia

La función de Splinecub se integra al código previamente mostrado en la Figura 57. Permitiendo realizar movimientos más complejos (Figura 59) sin poner en riesgo la estructura del robot.

Figura 58. Uso de la función Splinecub en código de control de Dynamixel

```
if q1!='no':
    p1=int( 512-q1*1023/(5/3*pi) )#q1
    dxl1_present_position0 = packetHandler.read2ByteTxRx(portHandler, DXL1_ID, ADDR_MX_PRESENT_POSITION)
    th10 = splinecub_2(dxl1_present_position0[0],p1,10,n0)
if q2!='no':
    p2=int( 512-q2*1023/(5/3*pi) ) #Q2
    dxl2_present_position0 = packetHandler.read2ByteTxRx(portHandler, DXL2_ID, ADDR_MX_PRESENT_POSITION)
    th20 = splinecub_2(dxl2_present_position0[0],p2,10,n0)
if q3!='no':
    p3=int( 512-q3*1023/(5/3*pi) ) #Q3
    dxl3_present_position0 = packetHandler.read2ByteTxRx(portHandler, DXL3_ID, ADDR_MX_PRESENT_POSITION)
    th30 = splinecub_2(dxl3_present_position0[0],p3,10,n0)
if q4!='no':
    p4=int( 512-q4 *1023/(5/3*pi))  #Q4
    dxl4_present_position0 = packetHandler.read2ByteTxRx(portHandler, DXL4_ID, ADDR_MX_PRESENT_POSITION)
    th40 = splinecub_2(dxl4_present_position0[0],p4,10,n0)
if q5!='no':
    p5=int( 512+q5*1023/(5/3*pi) )
    dxl5_present_position0 = packetHandler.read2ByteTxRx(portHandler, DXL5_ID, ADDR_MX_PRESENT_POSITION)
    th50 = splinecub_2(dxl5_present_position0[0],p5,10,n0)
if q6!='no':
    p6=int( 512-q6*1023/(5/3*pi) )
    dxl6_present_position0 = packetHandler.read2ByteTxRx(portHandler, DXL6_ID, ADDR_MX_PRESENT_POSITION)
    th60 = splinecub_2(dxl6_present_position0[0],p6,10,n0)
```

Fuente: Propia

Figura 59. Movimiento en OpenBotv v1 aplicando el método de Spline Cúbico

Fuente: Propia

- **Conversión de ángulos a posiciones:**

Conociendo de antemano que el modelo cinemático planteado entrega sus resultados en forma de ángulos en radianes, es necesario convertir este resultado a un valor de posición dentro del rango de operación del Dynamixel AX-12. El proceso de conversión se realiza mediante el uso de una ecuación definida para cada articulación y se integran en un script de Python como puede observarse en la Figura 60.

Figura 60. Ecuaciones para convertir de ángulos a posiciones (0 a 1023)

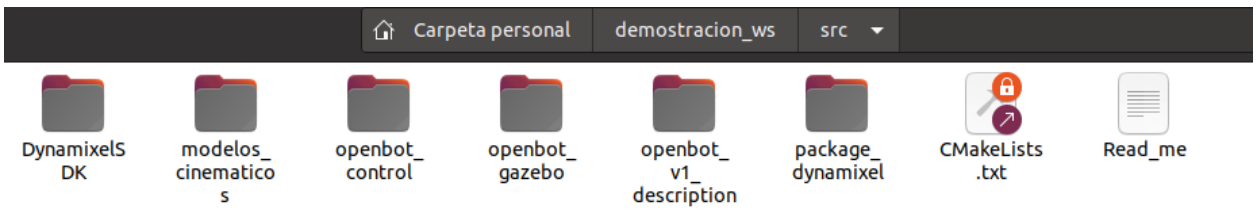
```
p1=int( 512-q1*1023/(5/3*pi) ) #Articulación 1
p2=int( 512-q2*1023/(5/3*pi) ) #Articulación 2
p3=int( 512-q3*1023/(5/3*pi) ) #Articulación 3
p4=int( 512-q4 *1023/(5/3*pi)) #Articulación 4
p5=int( 512+q5*1023/(5/3*pi) ) #Articulación 5
p6=int( 512-q6*1023/(5/3*pi) ) #Articulación 6
```

Fuente: Propia

- **Incorporación de estos cambios en el Workspace:**

Tomando como base el Workspace entregado por Robotics 4.0 S.A.S (Figura 49), los cambios realizados se reflejan en la introducción de nuevos paquetes y cambios en los existentes.

Figura 61. Estructura de Workspace en ROS desarrollado por el pasante



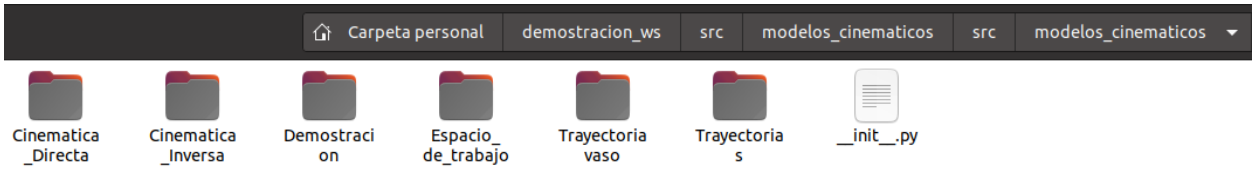
Fuente: Propia

- **DynamixelSDK:** Contiene los archivos de la librería Dynamixel Workbench.
- **package_dynamixel:** Contiene los archivos relacionados para manipular el robot físico.
- **Openbot_gazebo, openbot_v1_description, openbot_control:** Contiene los archivos relacionados al control, la definición de los parámetros y la manipulación del robot simulado.
- **Modelos_cinematicos:** Contendrá los archivos relacionados a estas temáticas.

7.2 DEMOSTRACIONES DE LOS MODELOS CINEMÁTICOS APLICADOS EN ROS

Tras tener un control total de las articulaciones simuladas y reales del robot se procedió a la implementación de los modelos cinemáticos. Para cada una de las temáticas se creó una subcarpeta dentro del paquete “modelos_cinematicos”.

Figura 62. Estructura de package modelos_cinematicos

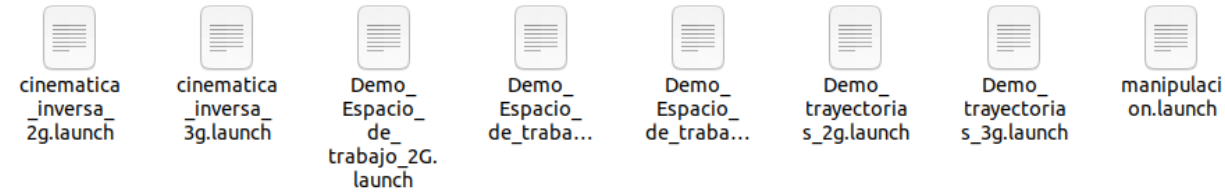


Fuente: Propia

Adicionalmente, cada tema que se trató posee una demostración o “Demo” que permite condensar y comprobar lo aprendido respecto a estas.

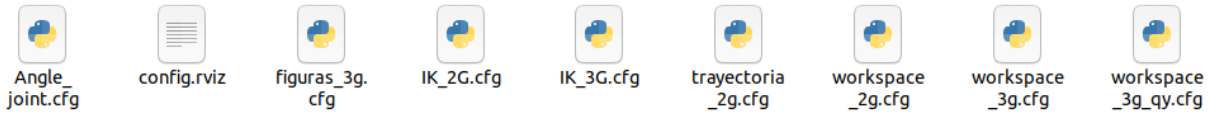
Estructura de las demos: La creación de cada demo involucra el uso de un archivo .launch y un archivo de configuración .cfg para que en conjunto con la herramienta de ROS “rqt_reconfigure” se tenga una interfaz de usuario. (Ver sección 5.2.7)

Figura 63. Archivos .launch desarrollados por el pasante



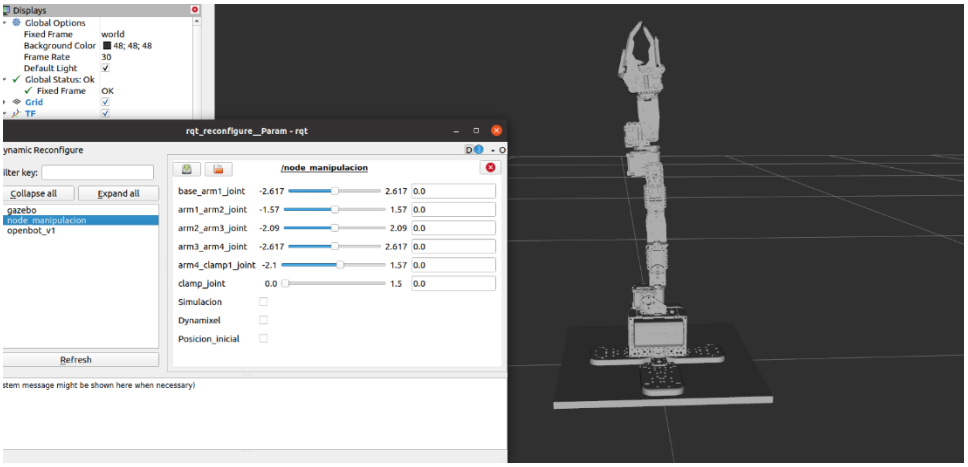
Fuente: Propia

Figura 64. Archivos .cfg desarrollados por el pasante



Fuente: Propia

Figura 65. Ejemplo de demostración desarrollada en ROS



Fuente: Propia

- **Demostración de aplicación de Cinemática inversa en Openbotv1 desde ROS:**

Los cálculos para el método de cinemática inversa se realizaron para una configuración de 2 y 3 grados de libertad (Ver sección 5.5.2). Estos dentro del Workspace, se encuentran definidos en una clase en Python denominada “cálculos_cinematica_inversa” que posee distintas funciones para representar las configuraciones que se consideran teóricamente. Cada función toma como entrada las coordenadas del efector final y retorna los ángulos producto del cálculo.

Figura 66. Script desarrollado para cálculos de cinemática inversa

cinematica_inversa_calculos.py

C: > Users > Tier One > Desktop > demostracion_ws > demostracion_ws > src > modelos_cinematicos > src > mode

```python
#! /usr/bin/env python
# Computing
from math import acos,atan,sqrt,pi,cos,sin
from numpy import array

class calculos_cinematica_inversa():

    def cal_2GDL_arri(x,z,L1, L2):
        r=sqrt(x*x+z*z)
        a=atan(abs(z)/abs(x))
        b=acos((L1*L1+r*r-L2*L2)/(2*L1*r))
        c=acos((L1*L1-r*r+L2*L2)/(2*L1*L2))

        if x==0 and z==0:
                print("Sin coordenadas")

        ## condiciones para cada cuadrante del robot
        #Cuadrante 1
        elif x>0 and z>0:
                q1p=0
                q1=a+b
                q2=c-pi
        #Cuadrante 2
        elif x<0 and z>0 :
                q1p=b-a
                q1=pi+q1p
                q2=c-pi
```

```python
#Retornamos el resultado
result=array([q1,-q2])
print(" ")
print('Codo Arriba')
print('Q1 vale en grados: ',q1g)
print('Q2 vale en grados  ',q2g)
print('El resultado en radianes es: ',-q1,q2)
print("""


 """)
return result
```

Fuente: Propia

En otro script principal, se llama esta clase en Python para obtener el resultado de los cálculos y aplicar los métodos desarrollados previamente para mover el robot simulado y real.

Figura 67. Aplicación de función en Python cálculos_cinematica_inversa

```python
def calculos(self,parameters):
    x=parameters[0]
    z=parameters[1]
    L1=parameters[2]
    L2=parameters[3]
    codo=parameters[4]
    try:
        if codo == "True":
            result=calculos_cinematica_inversa.cal_2GDL_aba(x,z,L1,L2)
        if codo ==  "False":
            result=calculos_cinematica_inversa.cal_2GDL_arri(x,z,L1,L2)
    except ValueError :
        print("Error Matematico, Configuraciòn Inalcanzable")
        result="Error"
    return result
```

Fuente: Propia

Figura 68. Script para mover robot simulado y real en base a los cálculos de cinemática inversa

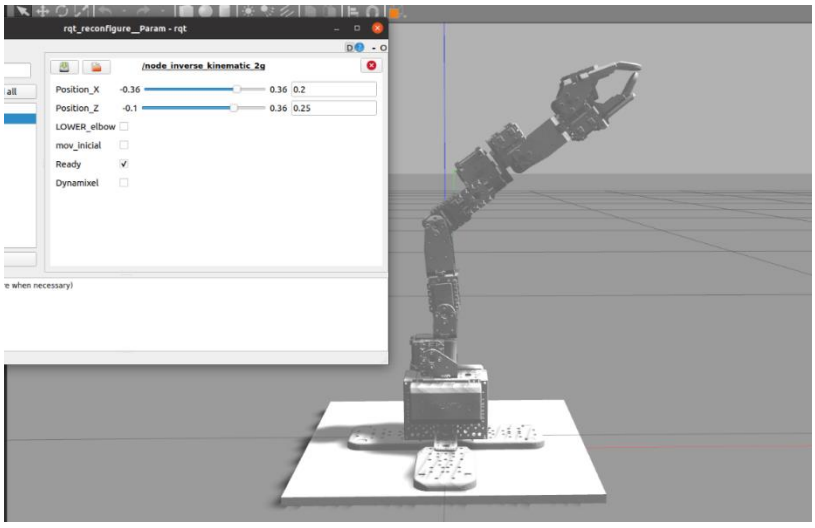
```
if "{Ready}".format(**Ik_2g)=="True" and "{mov_inicial}".format(**Ik_2g)=="False":
        parameters=[float("{Position_X}".format(**Ik_2g)),float("{Position_Z}".format(**Ik_2g)),L1,L2,"{LOWER_elbow}".format(**Ik_2g)]
        result=openbot_object.calculos(parameters) #Calculos de Cinematica Inversa
        if result == "Error":
            print("El robot no se moverà")
        else:
            openbot_object.move_openbot_toposition(result[0], result[1] ) #El robot se mueve segun q1 y q2
            if "{Dynamixel}".format(**Ik_2g)=="True" and "{mov_inicial}".format(**Ik_2g)=="False":
                print("Moving Dynamixel....")
                dynamixel.main()
                if float("{Position_X}".format(**Ik_2g)) > 0 and float("{Position_X}".format(**Ik_2g)) >= 0.15:
                    q1=-result[0]+0.17
                    q2=-result[1]+0.05
                elif float("{Position_X}".format(**Ik_2g)) < 0 and float("{Position_X}".format(**Ik_2g)) <= -0.15:
                    q1=-result[0]-0.17
                    q2=-result[1]-0.05
                else:
                    q1=-result[0]
                    q2=-result[1]
                dynamixel.movimiento_inicial('no', -q1 ,-q2 ,'no','no','no')
                print("Desfase = ",result[0]-q1,result[1]-q2)
```

Fuente: Propia

Explicación de las demos desarrolladas:

Demo Cinemática inversa 2GDL: La demo desarrollada para 2GDL permite ingresar la coordenada deseada del efector final y la orientación de codo (arriba o abajo). Con estos datos, se realiza el cálculo a través del método de cinemática inversa y se mueve las articulaciones 2 y 3 del robot para colocarlo en la posición indicada.

Figura 69. Demo de cinemática inversa para 2GDL

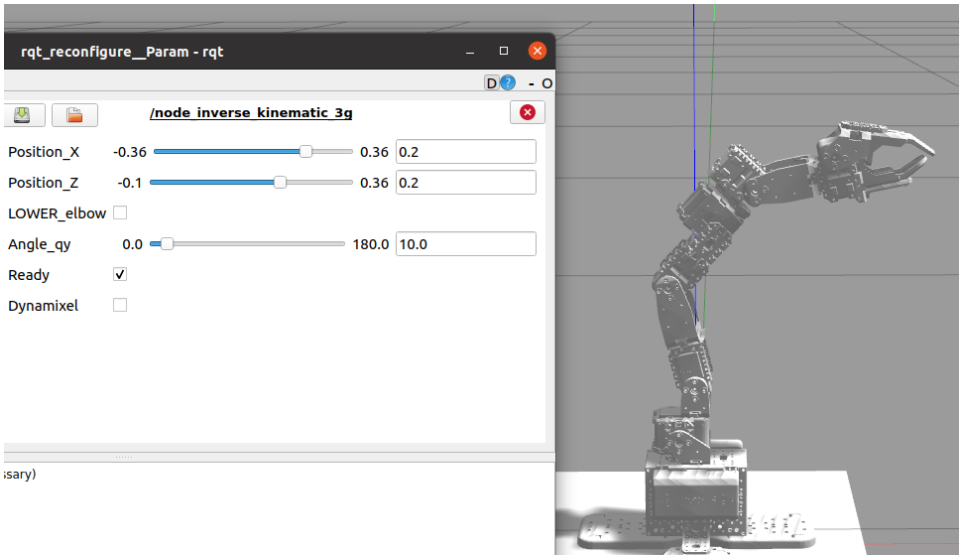


Fuente: Propia

Demo Cinemática inversa 3GDL: La demo desarrollada para 3GDL permite al usuario ingresar la coordenada deseada del efector final, su orientación (**qy**) y la orientación del codo (arriba o abajo). Posteriormente, aplica el método de

cinemática inversa y mueve las articulaciones 2, 3 y 5 del robot para colocarlo en la posición indicada.

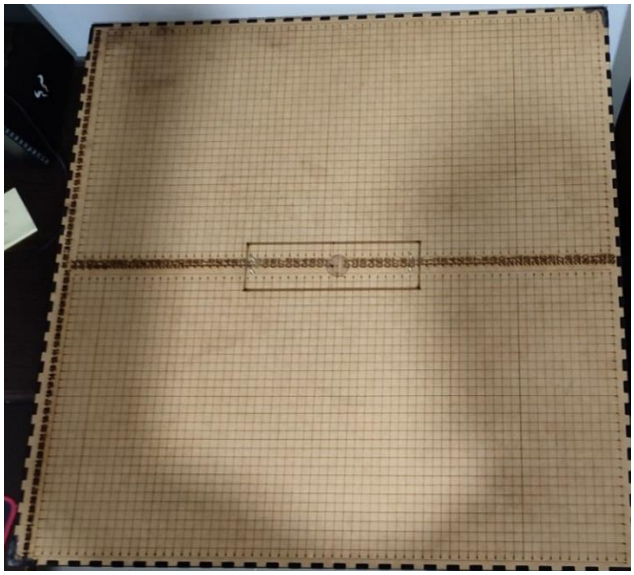
Figura 70. Demo de cinemática inversa para 3GDL



Fuente: Propia

Comprobación y exactitud del método de Cinemática inversa: Para comprobar la exactitud del movimiento del robot real aplicando las demos desarrolladas para cinemática inversa, se empleó una cuadrícula y un metro para realizar las mediciones.

Figura 71. Plano cartesiano en madera (35 cm X 35 cm)



Fuente: Propia

Tabla 3. Prueba de exactitud para 2GDL

	Valor Teórico (m)		Valor Medido con un metro(m)		Porcentaje de error	
	Coordenada X	Coordenada Z	Coordenada X	Coordenada Z	Eje X	Eje Z
1	0,2	0,25	0,198	0,263	1,0	-5,200
2	-0,2	0,26	-0,198	0,261	1,0	-0,385
3	0,05	0,25	0,05	0,248	0,0	0,800
4	0,1	0,25	0,1	0,245	0,0	2,000
5	-0,23	0,23	-0,23	0,233	0,0	-1,304
6	0,07	0,32	0,068	0,318	2,9	0,625
7	0,18	0,28	0,17	0,29	5,6	-3,571
8	0,3	0,03	0,295	0,028	1,7	6,667
9	0,24	0,04	0,24	0,038	0,0	5,000
10	0,24	0,2	0,24	0,198	0,0	1,000
				Total	1,2079	2,6552

Tabla 4. Prueba de exactitud para 3GDL

	Valor Teórico (m)		Valor Medido con un metro(m)		Porcentaje de error	
	Coordenada X	Coordenada Z	Coordenada X	Coordenada Z	Eje X	Eje Z
1	0,2	0,2	0,205	0,190	-2,5	5,000
2	0,22	0,2	0,22	0,198	0,0	1,000
3	0,26	0,2	0,265	0,198	-1,9	1,000
4	0,24	0,18	0,24	0,188	0,0	-4,444
5	0,24	0,15	0,238	0,145	0,8	3,333
6	0,04	0,27	0,035	0,26	12,5	3,704
7	0,04	0,2	0,05	0,18	-25,0	10,000
8	0,14	0,22	0,14	0,217	0,0	1,364
9	0,18	0,22	0,18	0,223	0,0	-1,364
10	0,1	0,24	0,105	0,228	-5,0	5,000
				Total	4,7756	3,6208

Tras realizar las mediciones para distintas configuraciones, se obtiene que para movimientos en 2GDL hay un porcentaje de error promedio de 1.2% en el eje x y 2.65% en el eje z. Para 3GDL, se tiene 4.77% en el eje x y 3.62% para el eje z.

Basado en los datos anteriores, el supervisor del proceso de pasantía decidió que el desempeño del método de cinemática inversa era aceptable para la aplicación de trayectorias que se haría posteriormente con este.

- **Demostración de trazo de espacio de trabajo en Openbotv1 desde ROS:**

Los cálculos desarrollados para cinemática directa (Ver sección 5.5.3) son la base para la construcción del espacio de trabajo. Estos de manera similar, al método de cinemática inversa se implementaron en una clase en Python denominada como "cálculos_cinematica_directa" que posee distintas funciones para representar las configuraciones que se consideraron teóricamente. Cada función toma como entrada los ángulos de las articulaciones y retorna la coordenada del efector final.

Figura 72. Script desarrollado para cálculos de cinemática directa

```
import numpy as np
from math import acos,atan,sqrt,pi,cos,asin,sin

class calculos_cinematica_directa():
    def cal_2GDL_abajo(q1,q2,L1,L2):
        #ROBOT CODO ABAJO
        #conversion de los angulos de grados a radianes
        q1 = (q1*pi)/180
        q2 = (q2*pi)/180

        # Condiciones para cada cuadrante

        # Primer cuadrante
        if(q1+q2 >=0 and q1+q2 <=(pi/2)):
            x = L1*cos(q1)+L2*cos(q1+q2)
            z= L1*sin(q1)+L2*sin(q1+q2)
            #print("1")

        #segundo cuadrante
        elif(q1+q2 > pi/2 and q1+q2 <= pi):
            # angulo auxiliar
            q2p = q1 + q2 - pi

            x = L1*cos(q1)-L2*cos(q2p)
            z = L1*sin(q1)-L2*sin(q2p)
            #print("2")
        #tercer cuadrante
        elif(q1+q2 > pi and q1+q2 <= ((3/2)*pi)):
            #angulo auxiliar
            q1p = q1 - pi
            x = -L1*cos(q1p)-L2*cos(q1p+q2)
            z = -L1*sin(q1p)-L2*sin(q1p+q2)
            #print("3")
        #cuarto cuadrante
        elif(q1+q2> ((3/2)*pi) and q1+q2 <= (2*pi)):
            #angulo auxiliar
            q1p = q1 - pi
            q2p = q1 + q2 - (2*pi)

            x = -L1*cos(q1p)+L2*cos(q2p)
            z = -L1*sin(q1p)+L2*sin(q2p)
            #print("4")
        return x,z
    #ROBOT CODO ARRIBA
    def cal_2GDL_arriba(q1,q2,L1,L2):
```

Fuente: Propia

Respecto al trazo del espacio de trabajo, los cálculos definidos para este (Ver sección 5.5.4) se encuentran en un script que contiene el gráfico equivalente para las configuraciones de 2GDL, 3GDL y 3GDL para una orientación específica del efector final.

Figura 73. Script desarrollado para cálculos de Espacio de trabajo

```
import numpy as np
from math import acos,atan,sqrt,pi,cos,asin,sin
from modelos_cinematicos.Cinematica_Directa.calculos_cinematica_directa import calculos_cinematica_directa

class Espacio_de_trabajo():

    def workspace_2g_abajo(self):
        #Dimensiones
        L1=0.094
        L2=0.262
        #Numero de muestras
        n=60
        #Inicializacion de vectores:
        x=np.zeros((n))
        z=np.zeros((n))
        x1=np.zeros((n))
        z1=np.zeros((n))
        x2=np.zeros((n))
        z2=np.zeros((n))


        #Primer limite o limite exterior
        q1=np.linspace(180,0,num=n,endpoint=True)
        q2= np.zeros(n)

        for i in range(n):
            x[i],z[i]= calculos_cinematica_directa.cal_2GDL_abajo(q1[i],q2[i],L1,L2,)
```

Fuente: Propia

Trazo del Espacio de trabajo en Rviz: Esta tarea se realiza en conjunto de la librería MarkerArray[7] que dibuja una serie de marcadores respecto a un punto de referencia. Para el trazo del espacio de trabajo, se escoge la articulación 2 del OpenBotv v1 debido a que al realizar los cálculos se asumió este punto como la coordenada (0,0) en el plano **XZ**.

Adicionalmente, dentro del mismo script se crea una función en Python (Figura 75) que toma los intervalos de giro definidos en la Tabla 1 y hace que el robot se mueva respecto a estos. De esta manera, se puede comprobar que la gráfica de espacio de trabajo producida es correcta.

[7] Open Robotics. (3 de marzo de 2023). ROS Wiki. Obtenido de rviz/DisplayTypes/Marker: http://wiki.ros.org/rviz/DisplayTypes/Marker

Figura 74. Función desarrollada para trazar el Espacio de trabajo en Rviz

```python
def Rviz_workspace_completo(self,coordenadas):
    self.Delete_marker()
    markerArray = MarkerArray()
    marker=Marker()
    marker.header.frame_id = "arm2_link_1"
    marker.type = marker.SPHERE
    marker.action = marker.ADD
    marker.scale.x = -0.02
    marker.scale.y = 0.01
    marker.scale.z = 0.01
    marker.scale.z = 0.01
    marker.color.a = 0.6
    marker.color.r = 0.4
    marker.color.g = 0.1
    marker.color.b = 0.5
    marker.pose.orientation.w = 1.0
    marker.pose.position.y = 0
    marker.lifetime=rospy.Duration(1000)
    N=0
    rate=rospy.Rate(1)
    id=0
    while not rospy.is_shutdown() and N<5:
        for par in coordenadas:
            for i in range(0,len(par[0])):
                marker.pose.position.x = -par[0][i]
                marker.pose.position.z = par[1][i]
                marker.id=id
                #print(markerArray.markers)
                markerArray.markers.append(marker)
                self.pub_marker.publish(markerArray)
                id += 1
        N=N+1
        rate.sleep()
```

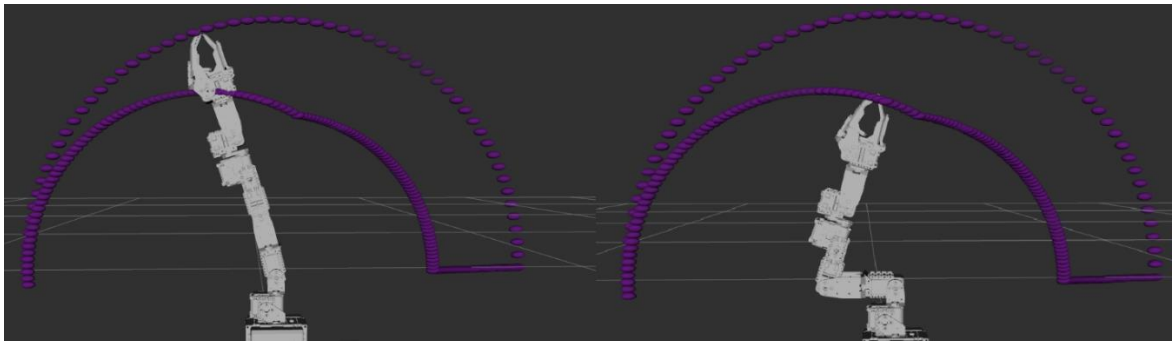
Fuente: Propia

Figura 75. Función desarrollada para mover OpenBotv v1 según los límites del Espacio de trabajo

```python
def Openbot_trazo_espacio_de_trabajo_abajo(self):
    n=10
    #Primer limite
    limite_1=np.linspace(1.57,-1.57,num=n,endpoint=True)

    for i in limite_1:
        self.mover_demo(i, 0)

    #Segundo limite
    limite_2=np.linspace(0,2.09,num=n,endpoint=True)
    for i in limite_2:
        self.mover_demo(-1.57, i )

    #Tercer limite
    limite_3=np.linspace(-1.57,-0.18,num=n,endpoint=True)
    for i in limite_3:
        self.mover_demo(i ,2.09 )

def Openbot_trazo_espacio_de_trabajo_arriba(self):
    n=10
    #Primer limite
    limite_1=np.linspace(-1.57,1.57,num=n,endpoint=True)

    for i in limite_1:
        self.mover_demo(i, 0)

    #Segundo limite
    limite_2=np.linspace(0,-2.09,num=n,endpoint=True)
    for i in limite_2:
        self.mover_demo(1.57, i )

    #Tercer limite
    limite_3=np.linspace(1.57,0.18,num=n,endpoint=True)
    for i in limite_3:
        self.mover_demo(i ,-2.09 )
```

Fuente: Propia

Figura 76. OpenBotv v1 simulado recorriendo el Espacio de trabajo trazado.

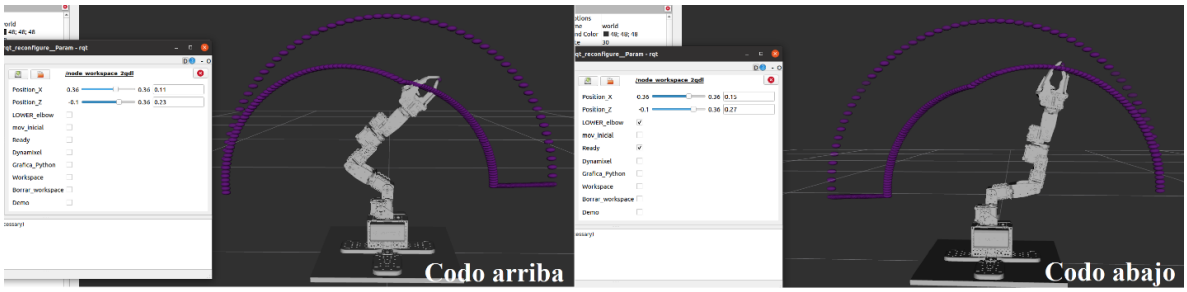


Fuente: Propia

Explicación de las demos desarrolladas:

Espacio de trabajo 2GDL: La demo desarrollada para 2GDL realiza un barrido de las articulaciones 2 y 3 para conocer todos los puntos en el plano **XZ** que puede alcanzar el efector final del robot.

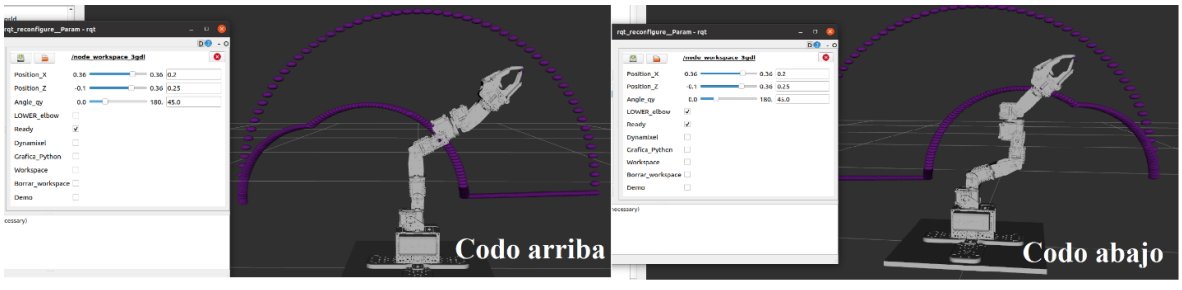
Figura 77. Demo desarrollada para Espacio de trabajo 2GDL



Fuente: Propia

Espacio de trabajo 3GDL: La demo desarrollada para 3GDL realiza un barrido de las articulaciones 2, 3 y 5 para conocer todos los puntos en plano **XZ** que puede alcanzar el efector final del robot.

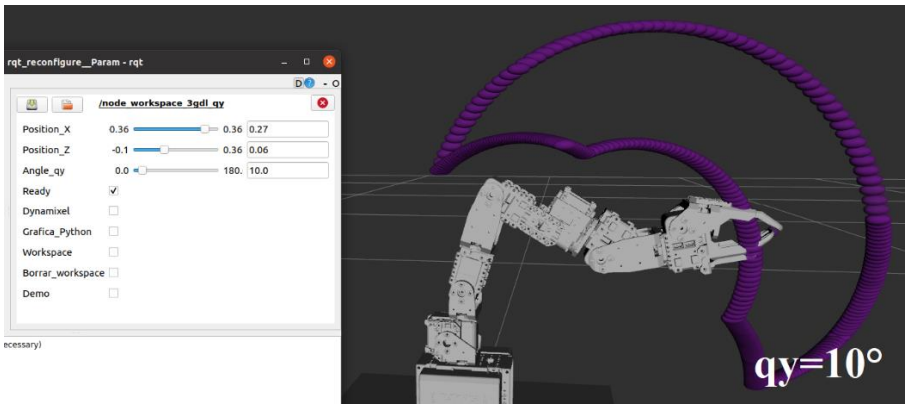
Figura 78. Demo desarrollada para Espacio de trabajo 3GDL



Fuente: Propia

Espacio de trabajo 3GDL respecto a una orientación específica: La demo desarrollada para 3GDL realiza un barrido de las articulaciones 2, 3 y 5 pero considerando que debe mantener una orientación específica del efector durante todo el recorrido. De esta manera se pueden conocer todos los puntos en el plano **XZ** que puede alcanzar el efector final del robot respecto a esa configuración.

Figura 79. Demo desarrollada para 3GDL respecto a una orientación especifica.

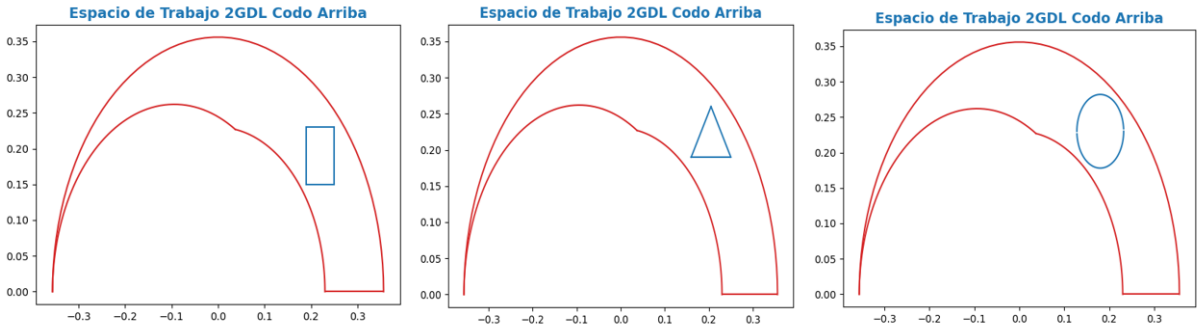


Fuente: Propia

- **Demostración de trazo de trayectorias en Openbotv1 desde ROS:**

La aplicación de trayectorias se realizó mediante el trazo de 3 figuras geométricas (Un cuadrado, un triángulo y un círculo) dentro de los espacios de trabajo previamente definidos para las configuraciones de 2GDL Y 3GDL.

Figura 80. Ejemplo de trazo de figuras geométricas dentro del espacio de trabajo



Fuente: Propia

La serie de puntos para trazar la trayectoria de cada una de las figuras geométricas se genera a través de un script que aplica el método de Spline Cúbico (Ver sección 5.5.5).

Figura 81. Uso de la función Splinecub para cada figura geométrica.

```python
def trazo_cuadrado(self,n,tf,codo):
    #Parametros del cuadrado
    if codo =="True":
        corrimientox=-0.01
        corriemientoz=0.25
        base=-0.06
        altura=0.06

    else:
        corrimientox=0.01
        corriemientoz=0.25
        base=0.06
        altura=0.06
    #Rango de X
    x1=splinecub(corrimientox,corrimientox+base,tf,n)
    x2=splinecub(corrimientox+base,corrimientox+base,tf,n)
    x3=splinecub(corrimientox+base,corrimientox,tf,n)
    x4=splinecub(corrimientox,corrimientox,tf,n)
    #Rango de z
    z1=splinecub(altura+corriemientoz,altura+corriemientoz,tf,n)
    z2=splinecub(altura+corriemientoz,corriemientoz,tf,n)
    z3=splinecub(corriemientoz,corriemientoz,tf,n)
    z4=splinecub(corriemientoz,corriemientoz+altura,tf,n)
    coordenadas=[[x1,z1],[x2,z2],[x3,z3],[x4,z4]]
    return coordenadas
```

```python
def trazo_triangulo(self,n,tf,codo):
    #Parametros del triangulo
    if codo =="True":
        limiteiz=-0.01
        limiteinf=0.25
        base=-0.09
        altura=0.07

    else:
        limiteiz=0.01
        limiteinf=0.25
        base=0.09
        altura=0.07

    limitede=limiteiz+base

    x1=splinecub(limiteiz,(limitede+limiteiz)/2,tf,n)
    x2=splinecub((limiteiz+limitede)/2,limiteiz+base,tf,n)
    x3=splinecub(limiteiz+base,limiteiz,tf,n)

    z1=splinecub(limiteinf,altura+limiteinf,tf,n)
    z2=splinecub(altura+limiteinf,limiteinf,tf,n)
    z3=splinecub(limiteinf,limiteinf,tf,n)
    coordenadas=[[x1,z1],[x2,z2],[x3,z3]]
    return coordenadas
```

```python
def trazo_circulo(self,n,tf,codo):
    #Parametros del circulo
    if codo =="True":
        radio =0.04
        yc=0.28 #Valor desplazar el centro del circulo en y
        xc=-0.025 #Valor desplazar el centro del circulo en x

    else:
        radio =0.04
        yc=0.285 #Valor desplazar el centro del circulo en y
        xc=0.025 #Valor desplazar el centro del circulo en x

    trayectoria_x= np.array(splinecub(-radio+xc,radio+xc,tf,n))
    trayectoria_x2= np.array(splinecub(radio+xc,-radio+xc,tf,n))
    z_pos=np.sqrt(radio**2-np.power(trayectoria_x-xc,2))+yc
    z_neg=(-1*np.sqrt(radio**2-np.power(trayectoria_x-xc,2))+yc)
    coordenadas=[[trayectoria_x,z_pos],[trayectoria_x2,z_neg]]
    return coordenadas
```

Fuente: Propia

Posteriormente, a cada punto se le aplica el método de cinemática inversa y el robot se mueve por los métodos desarrollados previamente para el modelo simulado y real.

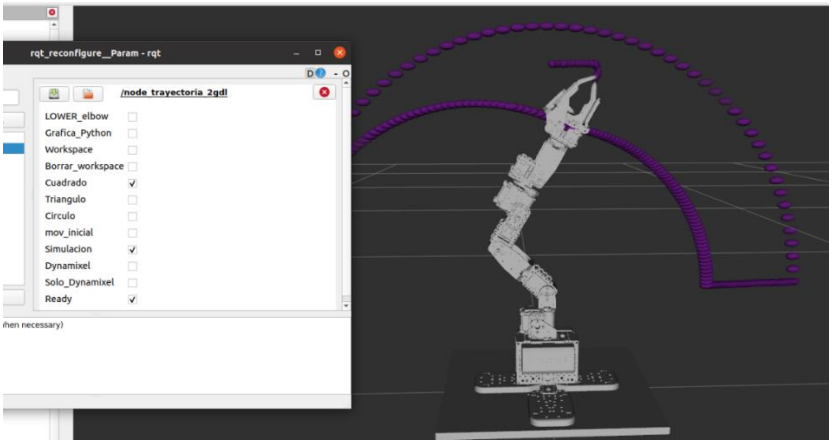
Trazo de figuras en Rviz: En conjunto con la librería MarkerArray se dibuja un punto en cada posición que recorre el efector final formando de esta manera la figura geométrica.

Figura 82. Función empleada para dibujar la posición del efector final en Rviz

```python
## Seccion de rviz
def Create_markers(header="effector_link_tf"):
        """Creat the marker for RVIZ"""
        markerArray = MarkerArray()
        marker=Marker()
        marker.header.frame_id = header
        marker.type = marker.SPHERE
        marker.action = marker.ADD
        marker.scale.x = 0.01
        marker.scale.y = 0.01
        marker.scale.z = 0.01
        marker.color.a = 0.6
        marker.color.r = 0.4
        marker.color.g = 0.1
        marker.color.b = 0.5
        marker.pose.orientation.w = 1.0
        marker.pose.position.x = 0
        marker.pose.position.y = 0
        marker.pose.position.z = 0
        return markerArray,marker
```

Fuente: Propia

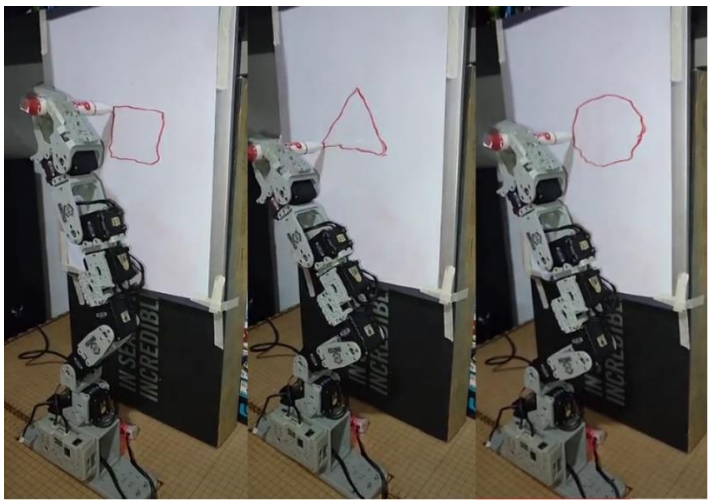
Figura 83. OpenBotv v1 trazando figuras dentro del espacio de trabajo trazado en Rviz.



Fuente: Propia

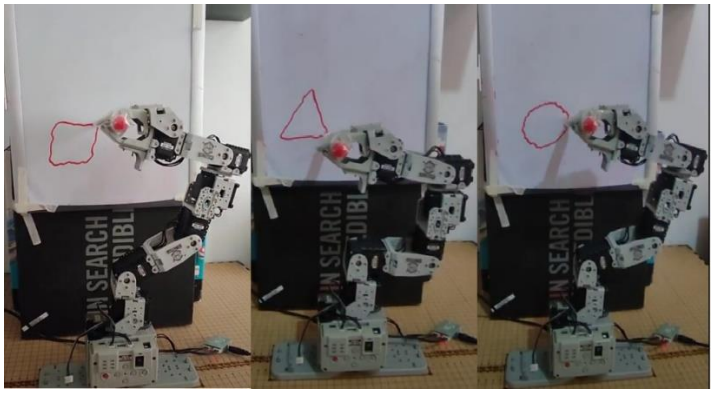
Trazo de figuras con robot real: Tomando los ángulos productos del método de cinemática inversa, el robot real simplemente se mueve siguiendo el recorrido de puntos de la trayectoria de cada una de las figuras geométricas.

Figura 84. Trazo de figuras con OpenBotv v1 respecto a 2GDL.



Fuente: Propia

Figura 85. Trazo de figuras con OpenBotv v1 respecto a 3GDL.

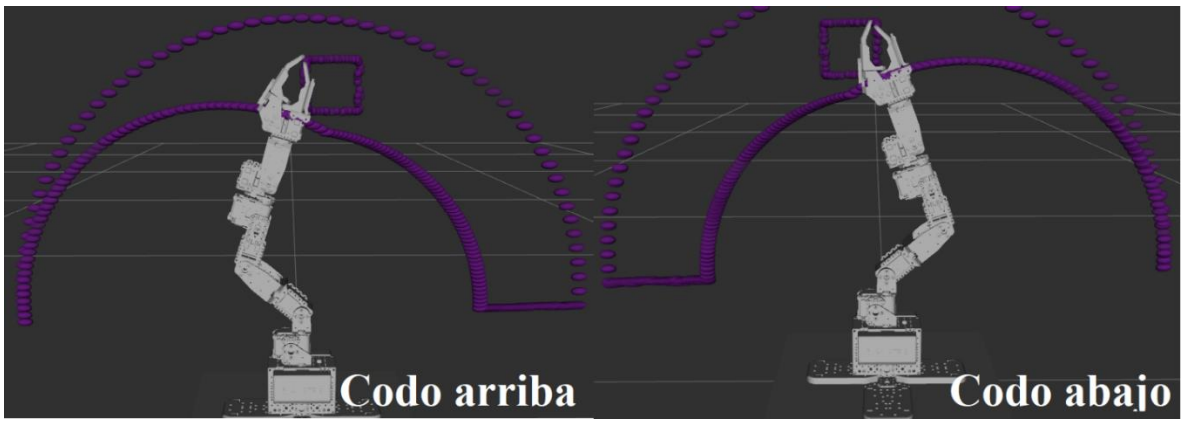


Fuente: Propia

Explicación de las demos desarrolladas:

Trazo de Figuras dentro de espacio de trabajo 2GDL: Para las configuraciones de codo abajo y arriba, considerando su espacio de trabajo equivalente, se planea el trazo 3 figuras geométricas dentro de este.

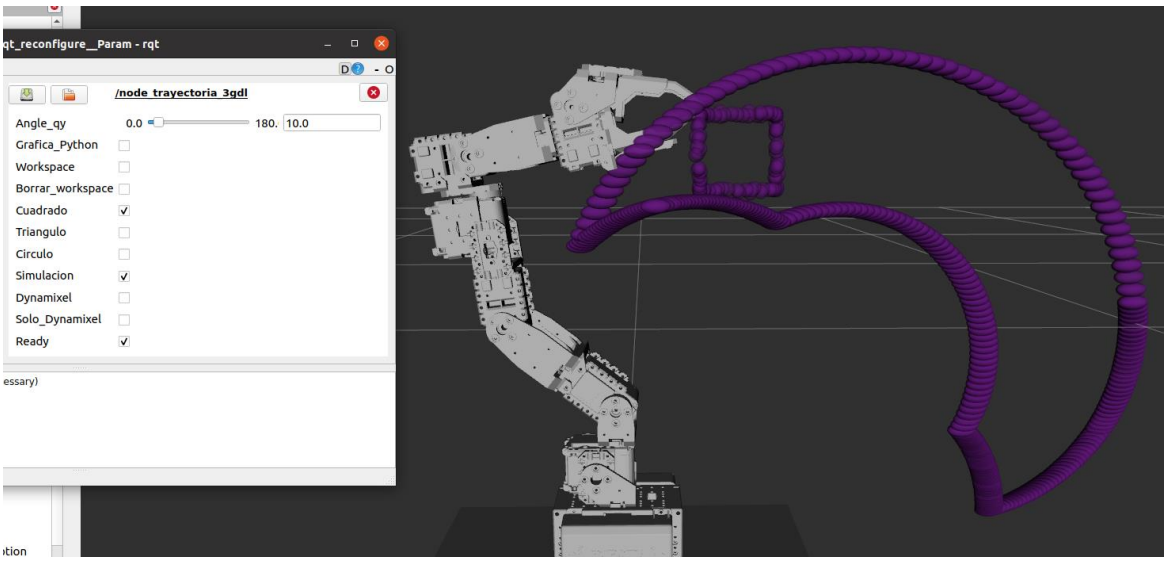
Figura 86. Demo desarrollada para trazo de figuras respecto 2GDL.



Fuente: Propia

Trazo de Figuras dentro de espacio de trabajo 3GDL: Considerando un ángulo de orientación del efector final de 10 grados, se realiza un recorrido de puntos manteniendo la configuración para trazar 3 figuras geométricas planeadas dentro del espacio de trabajo equivalente.

Figura 87. Demo desarrollada para trazo de figuras respecto 3GDL.



Fuente: Propia

7.3 ESTADO ACTUAL DEL MÓDULO DIDÁCTICO

Tras terminar la implementación de los modelos cinemáticos, el pasante documentó lo aprendido durante su proceso de pasantía en forma de guías y videos. El material desarrollado fue condensado y publicado por Robotics 4.0 S.A.S en la plataforma de Udemy en forma del curso de "Robótica Antropomórfica Básica en ROS"[8]. Contando a la fecha con un alcance de 60 estudiantes (Anexo1).

Figura 88. Curso de "Robótica Antropomórfica Básica en ROS" en la plataforma de Udemy

Fuente: Robotics 4.0. (16 de febrero de 2023). Udemy. Obtenido de Robótica Antropomórfica Básica en ROS: https://www.udemy.com/course/robotica-antropomorfica-basica-en-ros/?src=sac&kw=robotica%2Bantropomorf

[8] Robotics 4.0. (Open Robotics, 2023) (16 de febrero de 2023). Udemy. Obtenido de Robótica Antropomórfica Básica en ROS: https://www.udemy.com/course/robotica-antropomorfica-basica-en-ros/?src=sac&kw=robotica%2Bantropomorf

El curso consta de 8 unidades, en las cuales se explica el proceso detallado que realizó el pasante para la integración de OpenBotv v1 con ROS y la implementación de los modelos cinemáticos.

Figura 89. Contenido del curso publicado en Udemy

Fuente: Robotics 4.0. (16 de febrero de 2023). Udemy. Obtenido de Robótica Antropomórfica Básica en ROS: https://www.udemy.com/course/robotica-antropomorfica-basica-en-ros/?src=sac&kw=robotica%2Bantropomorf

8. CONCLUSIONES

Tras haber finalizado el proceso de pasantía para realizar el proyecto “Desarrollo de módulo didáctico de integración de ROS con robot OpenBotv v1 para fines académicos e investigación” se tienen las siguientes reflexiones:

- Como se ha descrito a lo largo del informe y lo certifica el Anexo A, se alcanzaron de forma exitosa los objetivos, tanto general como específico, que se plantearon en el anteproyecto de esta pasantía. Permitiendo que satisfactoriamente se desarrollara para Robotics 4.0 S.A.S un módulo didáctico sobre las temáticas tratadas durante el proceso, estando actualmente publicado en la plataforma de Udemy bajo el nombre del curso “Robótica Antropomórfica Básica en ROS”.

- La integración de OpenBotv v1 con ROS permitió de primera mano verificar las fortalezas de este software. Siendo un entorno en el cual puede interactuarse con modelos simulados y reales desde lenguajes de programación bastante conocidos como Python.

- El curso publicado por Robotics 4.0, posee un gran impacto social y académico para la región. Facilitando a los interesados en temáticas de robótica hispano hablantes adquirir conocimientos e instruirse en softwares bastante útiles en el área de la robótica como lo es ROS.

- Para alguien que ve su futura vida profesional en la rama de la robótica, el proceso de pasantía desarrollado en conjunto con Robotics 4.0 S.A.S fue una gran oportunidad. Permitiendo realizar un proceso de aprendizaje sobre ROS, un software que se utiliza a nivel industrial para el desarrollo de proyectos de robótica, y haber podido experimentar con un brazo robótico a pequeña escala como lo es el OpenBotv v1.

9. TRABAJO FUTURO

Como se observa en la Figura 65, las demostraciones de los modelos cinemáticos en ROS se implementaron empleando únicamente elementos nativos del software como: rqt_configure (Interfaz de usuario), archivo .cfg (menú) y archivo .launch (Ejecutable). Logrando de esta manera crear un método para interactuar con el robot real y simulado en tiempo real.

Si la empresa tuviera interés en ampliar la interfaz de usuario para hacerla más compleja y similar a trabajos previos que ha realizado en entornos como Matlab (Figura 90), se planteó la propuesta de implementarla nativamente en Python mientras los métodos de interacción con el Robot se siguen haciendo desde ROS. Esto debido a que herramientas como rqt_configure no están pensadas para ser interfaces modificables al gusto del usuario sino para ser entornos de prueba.

Figura 90. Interfaz de Usuario OpenBotv V1 en Matlab

Fuente: Toro Mendoza, S., & Nieto Solano, J. Desarrollo de una interfaz gráfica interactiva para el robot OpenBotv V1 en el entorno de MATLAB. Ingeniería Electrónica. BOGOTA D.C.: Facultad de Ingeniería (2022)

Adicionalmente, respecto al material académico publicado, por parte de Robotics 4.0 S.A.S, se espera que tomando como base el material desarrollado por el pasante, se continúe actualizando el curso de “Robótica Antropomórfica Básica en ROS” con el modelamiento de las mismas temáticas, pero desde una perspectiva más compleja como lo puede ser una configuración de 5 grados de libertad.

10. BIBILIOGRAFÍA

ABI Research. (19 de febrero de 2023). Obtenido de The rise of ROS: Nearly 55% of total commercial robots shipped in 2024 will have at least one robot operating system package installed: https://www.abiresearch.com/press/rise-ros-nearly-55-total-commercial-robots-shipped-2024-will-have-least-one-robot-operating-system-package-installed/

Quigley, M., Gerkey, B., & Smart, W. D. (2015). *Programming Robots With Ros: A practical introduction to the robot operating system.* Sebastopol: O'Reilly Media.

Toro Mendoza, S., & Nieto Solano, J. (2022). *Desarrollo de una interfaz gráfica interactiva para el robot OpenBotv V1 en el entorno de MATLAB.* BOGOTA D.C.

Apache 2.0. (16 de feberero de 2023). *Gazebo : Media*. Obtenido de https://classic.gazebosim.org/media

Apache 2.0. (16 de febrero de 2023). *Gazebo : Tutorial : ROS control*. Obtenido de https://classic.gazebosim.org/tutorials?tut=ros_control#Aboutros_control

Barrientos, A., Penín, L. F., Balaguer, C., & Aracil, R. (1997). *FUNDAMENTOS DE ROBÓTICA.* Madrid: McGRAW-HIILL.

Fairchild, C., & Harman, D. T. (2017). *Ros Robotics by example: Learning to control wheeled, limbed, and flying robots using Ros Kinetic Kame.* Birmingham: Packt Publishing Ltd.

GitHub. (16 de febrero de 2023). Obtenido de ros-visualization : https://github.com/ros-visualization

Lentin , J. (2018). *Robot Operating System (Ros) for absolute beginners: Robotics Programming made easy.* apress.

Open Robotics. (16 de febrero de 2023). *ROS wiki*. Obtenido de urdf/Tutorial/Building a Visual Robot Model with URDF from Scratch: http://wiki.ros.org/urdf/Tutorials/Building%20a%20Visual%20Robot%20Model%20with%20URDF%20from%20Scratch

Open Robotics. (13 de febrero de 2023). *ROS Wiki*. Obtenido de rqt/Plugins: http://wiki.ros.org/rqt/Plugins

Open Robotics. (13 de febrero de 2023). *ROS Wiki*. Obtenido de rqt/Plugins: http://wiki.ros.org/rqt/Plugins

Open Robotics. (3 de marzo de 2023). *ROS Wiki*. Obtenido de rviz/DisplayTypes/Marker: http://wiki.ros.org/rviz/DisplayTypes/Marker

Open Robotics. (16 de Febrero de 2023). *ROS: Home*. Obtenido de https://www.ros.org

Robotics 4.0. (12 de febrero de 2023). *Robotics 4.0*. Obtenido de E-Robotics 4.0: https://robotics40.com/e-robotics-4-0/

Robotics 4.0. (15 de Marzo de 2023). *Robotics 4.0*. Obtenido de E-Robotics 4.0: https://robotics40.com/wp-content/uploads/2019/04/OpenBotvv1.jpg

Robotics 4.0. (16 de febrero de 2023). *Udemy*. Obtenido de Robótica Antropomórfica Básica en ROS: https://www.udemy.com/course/robotica-antropomorfica-basica-en-ros/?src=sac&kw=robotica%2Bantropomorf

Robotics 4.0. (16 de febrero de 2023). *Youtube*. Obtenido de HuilaFEST 4.0: https://youtu.be/iPjFRZ1tPg0

ROBOTIS. (15 de febrero de 2023). *ROBOTIS*. Obtenido de SMPS2Dynamixel - ROBOTIS: https://www.robotis.us/smps2dynamixel/

ROBOTIS. (13 de febrero de 2023). *ROBOTIS e-Manual*. Obtenido de DYNAMIXEL Workbench: https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_workbench/

ROBOTIS. (15 de febrero de 2023). *ROBOTIS e-Manual*. Obtenido de AX-12A: https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/

ROS Wiki. (12 de febrero de 2023). Obtenido de rospy: http://wiki.ros.org/rospy

11. ANEXOS

- **Anexo: Certificado de finalización de proceso de pasantía en Robotics 4.0 S.A.S**

Yo, **MARIO RICARDO ARBULÚ SAAVEDRA** con cédula extranjera N° 386397 en calidad de Socio y CEO de la empresa **Robotics 4.0 SAS**, a petición de la parte interesada

CERTIFICO:

Que **ANDRES FELIPE VEGA TIQUE**, identificado con cedula de ciudadanía No. 1.003.815.365 de Neiva, Huila - Colombia, mayor de edad y domiciliado en Neiva, Huila – Colombia trabajó como pasante **CO-DESARROLLANDO 1 CURSO VIRTUAL** de Robótica Antropomórfica Básica en ROS, bajo la supervisión de la dirección Educativa de nuestra empresa, durante un periodo de 7 meses, comprendidos entre el diecisiete (17) de junio del 2022 y treinta y uno (31) de enero del 2023 demostrando ser una persona altamente responsable, dedicada y sobresaliente con sus deberes ante la empresa. Cabe recalcar además su proactividad, iniciativa y co-generación del mencionado producto digital, que está actualmente en el mercado con impacto en 4 países del mundo, y más 60 estudiantes inscritos en los 30 días de lanzamiento a la web.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento en lo que estimare conveniente. Cualquier información adicional pude contactarse con nosotros. La presente certificación se expide en la ciudad de Neiva, a los 16 días del mes de febrero del dos mil veintitrés (2023).

Neiva, 16 de febrero de 2023.

Atentamente:

MARIO RICARDO ARBULÚ SAAVEDRA, PhD
Socio y CEO
C.E 386397
Cel. +57 3134681095
Email: ceo@robotics40.com

ceo@robotics40.com
www.robotics40.com